Programación de servidores web con CGI, SSI e IDC

Sergio Luján Mora

Programación de servidores web con CGI, SSI e IDC

Sergio Luján Mora

Prefacio

Las aplicaciones web (*web-based application*) se clasifican dentro de las aplicaciones cliente/servidor. Por un lado, se tiene el navegador (*browser*) que hace el papel de cliente; por otro lado, se tiene el servidor web que representa la parte servidor. Para crear cada una de las partes, cliente y servidor, se emplean distintas tecnologías. Así, por ejemplo, para programar un cliente web se suele utilizar HTML, JavaScript o *applets* en Java, mientras que para programar un servidor web se emplea CGI, SSI, ASP o JSP.

En este libro se repasan las tecnologías que fueron esenciales en la programación de los servidores web durante los primeros años de la web. Las tres tecnologías que se presentan en este libro, CGI, SSI e IDC, permiten crear páginas web dinámicas.

Mientras que CGI y SSI aún se emplean muy a menudo, IDC ha sido superado por tecnologías que han aparecido posteriormente. Sin embargo, debido a la sencillez de IDC, he considerado que es un punto de inicio muy adecuado para afrontar el estudio de tecnologías más avanzadas pero a su vez más complicadas.

Para afrontar correctamente el estudio de los temas tratados en este libro, hace falta poseer unos mínimos conocimientos sobre HTML. Existen multitud de libros sobre HTML, pero recomiendo la consulta del libro *Programación en Internet: Clientes Web* que he publicado en Editorial Club Universitario. En él, se trata la programación de la parte cliente de las aplicaciones web y en particular se estudian HTML y JavaScript.

El contenido de este libro se ha dividido en tres capítulos y un apéndice. Los tres capítulos son independientes, por lo que se pueden leer en cualquier orden. El libro además posee una serie de índices que permiten su empleo como obra de referencia.

El capítulo primero trata sobre CGI: presenta el estándar CGI, describe

las distintas formas que existen de enviar información a un programa CGI, explica cómo emplear las variables de entorno y comenta algunos consejos que pueden ayudar a lograr programas CGI más seguros. El lenguaje empleado para programar los CGI de ejemplo que contiene este capítulo es *C*, por lo que es necesario poseer unos conocimientos mínimos de *C* o *C++* para comprenderlos.

El segundo capítulo está dedicado a SSI. Se explica su uso, los comandos más comunes (no todos los servidores web aceptan los mismos comandos) y se incluyen varios ejemplos.

El tercer capítulo explica la tecnología IDC de Microsoft y cómo generar páginas web dinámicas a partir de la información almacenada en una base de datos.

Por último, el único apéndice del libro complementa el capítulo tres, ya que explica como crear un DSN para acceder a una base de datos mediante ODBC.

Para finalizar, quisiera mandar un abrazo a mi familia y a Marisa, la gente que quiero; un saludo a los amigos y compañeros del Laboratorio Multimedia (mmlab), con los que trabajé y disfruté de buenos momentos, y otro saludo a los amigos y compañeros del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante, con los que trabajo (y espero seguir trabajando).

Alicante, 11 de noviembre de 2001 *Sergio Luján Mora*

Índice general

Índice de cuadros

Índice de figuras

Índice de acrónimos

API *Application Program Interface*
Interfaz de programación de aplicaciones. Conjunto de constantes, funciones y protocolos que permiten programar aplicaciones. Una buena API facilita la tarea de desarrollar aplicaciones, ya que facilita todas las piezas y el programador sólo tiene que unirlas para lograr el fin que desea.

ASP *Active Server Pages*
Páginas activas de servidor. Tecnología de MICROSOFT que permite crear páginas web dinámicas en el servidor. Se puede decir que las páginas **ASP** son similares a los programas **CGI**. Las páginas **ASP** suelen estar programadas en *VBScript*, aunque también se pueden programar en otros lenguajes.

ASCII *American Standard Code for Information Interchange*
Código binario utilizado para representar letras, números, símbolos, etc. A cada carácter se le asigna un número del 0 al 127 (7 bits). Por ejemplo, el código **ASCII** para la A mayúscula es 65. Existen códigos **ASCII** extendidos de 256 caracteres (8 bits), que permiten representar caracteres no ingleses como las vocales acentuadas o la eñe. Los caracteres de la parte superior (128 a 255) de estos códigos **ASCII** extendidos varían de uno a otro. Por ejemplo, uno de los más extendidos es **ISO** Latin-1 (oficialmente ISO-8859-1).

CGI *Common Gateway Interface*
Interfaz de pasarela común. Estándar que permite el intercambio de información entre un servidor y un programa externo al servidor. Un programa **CGI** es un programa preparado para recibir y enviar datos desde y hacia un servidor web según este estándar. Normalmente se programan

en *C* o en *Perl*, aunque se puede usar cualquier lenguaje de propósito general.

DLL *Dynamic Link Library*
Librería de enlace dinámico. Fichero que almacena funciones ejecutables o datos que pueden ser usados por una aplicación en Microsoft Windows. Una **DLL** puede ser usada por varios programas a la vez y se carga en tiempo de ejecución (no en tiempo de compilación).

DNS *Domain Name System*
Sistema de nombres de dominio. Servicio de Internet que traduce los nombres de dominio en direcciones **IP**. Cada vez que se emplea un nombre de dominio, un servidor de **DNS** tiene que traducir el nombre de dominio en su correspondiente dirección **IP**. Por ejemplo, el nombre de dominio `www.ua.es` se corresponde con la dirección **IP** `193.145.233.99`.

DSN *Data Source Name*
Nombre de origen de datos. Un **DSN** representa toda la información necesaria para conectar una aplicación con una base de datos mediante **ODBC**.

HTML *HyperText Markup Language*
Lenguaje de etiquetado de hipertexto. Lenguaje compuesto de una serie de etiquetas o marcas que permiten definir el contenido y la apariencia de las páginas web. Aunque se basa en **SGML**, no se puede considerar que sea un subconjunto. Existen cientos de etiquetas con diferentes atributos. **W3C** se encarga de su estandarización. El futuro sustituto de **HTML** es **XHTML**.

HTTP *HyperText Transfer Protocol*
Protocolo de transferencia de hipertexto. Es el protocolo que se emplea en **WWW**. Define como se tienen que crear y enviar los mensajes y que acciones debe tomar el servidor y el navegador en respuesta a un comando. Es un protocolo *stateless* (sin estado), porque cada comando se ejecuta independientemente de los anteriores o de los posteriores. Actualmente, la mayoría de los servidores soportan **HTTP** 1.1. Una de las principales ventajas de esta versión es que soporta conexiones persistentes: una vez que el navegador se conecta al servidor, puede recibir múltiples ficheros a través de la misma conexión, lo que aumenta el rendimiento de

la transmisión hasta en un 20 %. Se puede consultar el estándar en **RFC** 2616 (junio 1999).

IDC *Internet Database Connector*
Conector de bases de datos de Internet. Tecnología propietaria de MICROSOFT que permite generar páginas web dinámicas a partir de la información almacenada en una base de datos. Es el precursor de **ASP**.

IP *Internet Protocol*
Protocolo de Internet. Protocolo básico de Internet perteneciente a la familia **TCP/IP**. Especifica el formato de los paquetes (datagramas) y el esquema de direccionamiento.

ISAPI *Internet Server Application Program Interface*
Un API para el servidor Microsoft Internet Information Server. Permite programar aplicaciones web.

ISO *International Organization for Standards*
Organización fundada en 1946, cuyos miembros son las organizaciones nacionales de normalización (estandarización) correspondientes a los países miembros. Entre sus miembros se incluyen la ANSI (Estados Unidos), BSI (Gran Bretaña), AFNOR (Francia), DIN (Alemania) y UNE (España).

JSP *Java Server Pages*
Tecnología de SUN MICROSYSTEMS que permite crear páginas web dinámicas en el servidor. Equivale a la tecnología **ASP** de MICROSOFT. Se programan en *Java*.

MIME *Multipurpose Internet Mail Extensions*
Se usa en el correo electrónico desde 1992 para enviar y recibir ficheros de distinto tipo. Se puede consultar el estándar en **RFC** 1341, **RFC** 1521 y **RFC** 1522.

ODBC *Open Database Connectivity*
Conectividad abierta de bases de datos. **ODBC** es un estándar *de facto* para el acceso a base de datos en entornos cliente/servidor. Mediante **ODBC**, se puede cambiar la parte servidor (la base de datos) sin tener que cambiar el cliente.

RFC *Request for Comments*
Medio de publicar propuestas sobre Internet. Cada **RFC** recibe un número. Algunos se convierten en un estándar de Internet.

SGBD *Sistema Gestor de Bases de Datos*
Programa (o programas) que permite almacenar, modificar y extraer información contenida en una base de datos. Los **SGBD** se pueden clasificar según la forma que tienen de almacenar internamente los datos: modelo relacional, en red, jerárquico, etc.

SGML *Standard Generalized Markup Language*
Lenguaje que permite organizar y etiquetar los distintos elementos que componen un documento. Se emplea para manejar grandes documentos que sufren constantes revisiones y se imprimen en distintos formatos. Desarrollado y estandarizado por **ISO** en 1986 (ISO 8879:1986).

SQL *Structured Query Language*
Lenguaje de consulta estructurado. Lenguaje estandarizado de acceso a bases de datos. Basado en SEQUEL (*Structured English Query Language*), diseñado por IBM en 1974. Existen distintas versiones, siendo la más conocida SQL-92 y la última publicada y estandarizada SQL-1999.

SSI *Server Side Include*
Directivas de inclusión del servidor. Comandos que se incluyen en una página **HTML** y que son ejecutados por el servidor web antes de transmitir la página al cliente. Permite generar páginas web dinámicas.

TCP/IP *Transmission Control Protocol/Internet Protocol*
Familia de protocolos que se emplean en las comunicaciones de Internet.

URL *Universal Resource Locator*
También conocido como *Uniform Resource Locator*. Sistema de direccionamiento de máquinas y recursos en Internet. Es decir, se trata de una dirección que permite localizar cualquier máquina o documento que se encuentre accesible a través de Internet.

W3C *World Wide Web Consortium*
Consorcio internacional de compañías involucradas en el desarrollo de Internet y en especial de la **WWW**. Su propósito es desarrollar estándares y "poner orden" en Internet.

WWW *World Wide Web*
Sistema de servidores web conectados a Internet (no todos los ordenadores conectados a Internet forman parte de la **WWW**). Su protocolo de comunicación es **HTTP**, su lenguaje de creación de documentos **HTML** y su sistema de direccionamiento de los recursos **URL**. Los navegadores web (*browsers*) permiten navegar por la web.

XHTML *Extensible HyperText Markup Language*
HTML escrito según las normas que marca **XML**. Por tanto, se trata de una aplicación concreta de **XML** y no tienen que confundirse entre sí.

XML *Extensible Markup Language*
Metalenguaje de etiquetado basado en **SGML**. Diseñado específicamente para la **WWW** por **W3C**. Permite que un usuario diseñe sus propias etiquetas, con sus atributos y las reglas de construcción de documentos (sintaxis).

Capítulo 1

CGI

El interfaz CGI permite que un cliente web (un navegador) ejecute un programa en el servidor web. Por medio de CGI se pueden crear páginas web dinámicas. El programa CGI y el servidor web se comunican a través de la salida y entrada estándar. Los programas CGI pueden ser escritos mediante diferentes lenguajes de programación.

Índice General

1.1. Introducción

Common Gateway Interface (**CGI**) es un interfaz que permite transferir información entre un servidor web y un programa externo al servidor. ¿Por qué es necesario el estándar **CGI**? Si queremos acceder desde un servidor web a una aplicación externa, una primera solución puede ser incluir en el servidor web un interfaz para cada una de las aplicaciones externas que se quiera ejecutar. Pero está solución es claramente inviable: es difícil y laborioso programar un

servidor web para que pueda acceder a todas las posibles aplicaciones existentes y, además, mantenerlo "al día" según surjan nuevas aplicaciones. En vez de ello, mediante **CGI** se establece un conjunto de normas (protocolo) que deben de seguir los servidores web y las aplicaciones para poder interactuar entre sí.

En la Figura 1.1 está representado el funcionamiento básico de una aplicación web basada en **CGI**:

1. El cliente web (el navegador) lanza una petición nueva mediante *HyperText Transfer Protocol* (**HTTP**). Está petición puede ir acompañada de datos codificados por el navegador (por ejemplo, información introducida por el usuario en un formulario).

2. El servidor web recibe la petición, analiza la *Universal Resource Locator* (**URL**) y detecta que se trata de un programa **CGI**. Ejecuta el **CGI** y le pasa los datos codificados.

3. El **CGI** recibe los datos codificados, los descodifica y realiza su función (en algunos casos, es posible que un programa **CGI** no necesite recibir datos para cumplir su misión). La función que realiza el programa **CGI** se puede clasificar en procesamiento directo (el programa **CGI** realiza por sí mismo todo el procesamiento de los datos recibidos) y procesamiento indirecto (el programa **CGI** interactúa con otras aplicaciones que son las verdaderas destinatarias de los datos recibidos, como por ejemplo, un *Sistema Gestor de Bases de Datos* (**SGBD**)).

4. El programa **CGI** genera su resultado: una página *HyperText Markup Language* (**HTML**), una imagen, un archivo de sonido, etc. y lo envía al servidor web.

5. El servidor web procesa la información recibida del programa **CGI**: le añade el código necesario para formar un encabezado **HTTP** correcto[1].

6. El servidor web reenvía el resultado del programa **CGI** al cliente web.

7. El cliente web muestra la salida del programa **CGI**.

[1] Más adelante veremos que se puede evitar este procesamiento y "hablar" directamente al cliente.

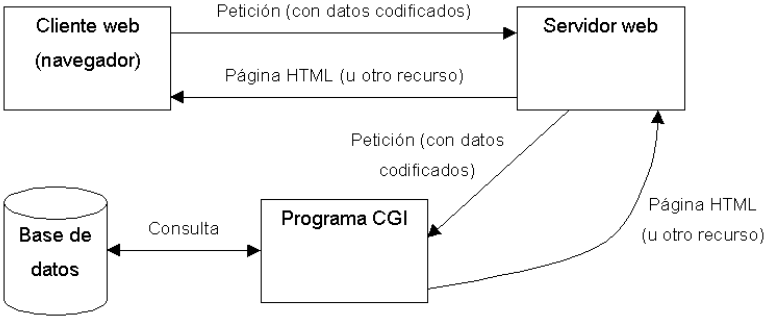

Figura 1.1: Esquema básico de una aplicación web basada en CGI

El uso de **CGI** supone un aumento en la complejidad de los sitios web, ya que se requieren conocimientos de programación y de administración de permisos de los sistemas operativos empleados en los servidores web.

La versión actual de este estándar es `CGI/1.1`. Las principales diferencias entre una página **HTML** normal y una página generada a partir de un programa **CGI** aparecen resumidas en el Cuadro 1.1.

Página HTML	**CGI**
El servidor web **recupera** la página El contenido es **estático**	El servidor web **ejecuta** el programa **CGI** El contenido puede ser **dinámico**

Cuadro 1.1: Diferencias entre una página HTML normal y una página generada a partir de un CGI

¿Qué se puede hacer con un **CGI**? En principio, no hay limitaciones. Pero siempre hay que tener en cuenta la siguiente recomendación: *cualquier cosa que haga un CGI, lo tiene que hacer rápidamente y empleando la menor cantidad posible de recursos.* Si no, el usuario se desesperará, se conectará a otra página y se prometerá a sí mismo no volver a visitar esa web donde las páginas tardaban una eternidad.

1.2. Un ejemplo

Cuando se introduce en el formulario de un buscador (por ejemplo, GOOGLE, ALTAVISTA o YAHOO!) un termino a buscar, el navegador (Microsoft Internet Explorer o Netscape Communicator) envía una petición al servidor web (Apache o Microsoft Internet Information Server[2]) en la que se solicita una página nueva y que se acompaña del término a buscar.

El servidor web recibe la solicitud, comprueba que la página que se solicita es un programa **CGI** y lo ejecuta pasándole el término a buscar. Además del término a buscar, le pasa información auxiliar en forma de variables de entorno, como por ejemplo la dirección *Internet Protocol* (**IP**) del cliente, método que ha empleado para enviar el término a buscar, etc.

El programa **CGI** realiza una búsqueda en una base de datos (o en un fichero plano) y localiza la información solicitada. El programa **CGI** genera de forma dinámica y en tiempo real una página **HTML** nueva a partir de la información encontrada y envía el resultado al servidor web.

El servidor web reenvía la página generada por el programa **CGI** al navegador.

1.3. Aplicaciones

El uso de programas **CGI** permite incorporar interacción en un sitio web: en vez de un sitio web estático se puede tener un sitio interactivo que se adapte a las necesidades de los distintos usuarios (páginas web dinámicas según el perfil de cada usuario). Las aplicaciones de los programas **CGI** son múltiples:

- Gestión de un libro de visitas o firmas (*guestbook*). Mediante un **CGI** se puede recuperar la información introducida en un formulario de un libro de visitas, almacenarla en un fichero y mostrar en una página web todas las visitas recibidas.

- Gestión de anuncios (*banners*). Mediante un **CGI** se pueden mostrar de forma aleatoria o de forma prefijada (por ejemplo, según la hora del día o según la dirección **IP** del cliente) distintos anuncios con distintas direcciones de enlace. Además, se puede controlar el número de pulsaciones (*clicks*) que recibe cada anuncio.

[2] Normalmente se conoce por sus siglas: IIS.

- Gestión de contadores (*hit counters*). Ya sea contadores en modo texto o modo gráfico (el programa **CGI** devuelve una imagen que contiene el valor del contador).

- Imágenes sensibles procesadas en el servidor web[3]. Las imágenes sensibles o mapas de imágenes son imágenes que contienen zonas activas que actúan como enlaces: en función de la zona de la imagen en la que pulse el usuario, se activa un enlace hacia un documento u otro. También se puede hacer zoom en una imagen mediante esta técnica.

- Acceso a bases de datos. Se puede emplear un programa **CGI** como pasarela (de ahí el nombre de *gateway*) para acceder a una base de datos. De este modo, se pueden crear aplicaciones como buscadores, comercio electrónico, etc.

1.4. Qué necesito para programar un CGI

Para poder programar un **CGI** y probarlo hacen falta los siguientes programas:

- Un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux para crear las páginas **HTML** que conectan con el programa **CGI** y para crear el propio código del programa **CGI**.

- Si se va a programar el **CGI** mediante un lenguaje compilado (*C*, *C++*, *Pascal*, etc.), hace falta el correspondiente compilador. Si se va a programar mediante un lenguaje interpretado (*Perl*, *shell* de Unix, etc.), hace falta el correspondiente intérprete.

- Un servidor web (ya sea local o remoto) en el que se puedan ejecutar programas **CGI**. Por ejemplo, Microsoft Personal Web Server, Microsoft Internet Information Server o Apache.

- Por último, un navegador como Netscape Communicator o Microsoft Internet Explorer para poder comprobar las páginas **HTML** y los programas **CGI**.

[3] También existen las imágenes sensibles procesadas en el cliente.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Lo que sí que es recomendable es utilizar un buen editor de textos, que sea cómodo, configurable, soporte macros, etc. y que sea *syntax highlight*. Esta última característica significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean variables, palabras reservadas, comentarios, etc.

1.5. Lenguaje de programación

Como un **CGI** es un programa que se ejecuta en el servidor, se puede programar en cualquier lenguaje que permita crear ejecutables para el sistema operativo del servidor. Lo único que se le exige al lenguaje de programación es que sea capaz de:

- Leer datos de la entrada estándar.
- Acceder a las variables de entorno.
- Escribir en la salida estándar.

Por tanto, la elección de un lenguaje se basa principalmente en qué lenguajes se conocen y qué lenguajes están disponibles en el sistema. Probablemente, *C* y *Perl* son los lenguajes más empleados a la hora de programar **CGI**.

Por razones históricas, a los programas **CGI** se les suele llamar también *scripts*[4], porque al principio se programaban con lenguajes de *script*. Mucha gente prefiere escribir los programas **CGI** con lenguajes de *script* en vez de lenguajes compilados, porque son más fáciles de depurar, modificar y mantener que un programa compilado. Sin embargo, los programas compilados son más rápidos a la hora de ejecutarse, ya que los *scripts* son interpretados.

Por tanto, la lista de lenguajes de programación que se pueden emplear no tiene límite; en el Cuadro 1.2 se muestran los más empleados en la programación de **CGI** (la lista no es excluyente: nada nos impide programar un **CGI** en *Python*, *Fortran*, *Pascal*, *TCL* o en nuestro lenguaje favorito).

Como la tecnología **CGI** se encuentra muy extendida en el mundo Internet, existen multitud de librerías en los distintos lenguajes de programación que

[4] Normalmente, se emplea la palabra programa para denotar aplicaciones y código "largo y compilado" mientras que *script* hace referencia a código "corto y no compilado".

Lenguaje	Sistema	Tipo
Cualquier shell de Unix	Unix	Interpretado
Perl	Unix, Windows, MacOS	Interpretado
C, C++	Unix, Windows, MacOS	Compilado
Visual Basic	Windows	Compilado
AppleScript	MacOS	Interpretado
REXX	OS2	Interpretado

Cuadro 1.2: Lenguajes de programación más comunes

facilitan la creación de programas **CGI**: cgi-lib (*Perl*), CGI-HTML (*C*), AHTML (*C++*), etc.

1.5.1. Independencia de plataforma

La independencia de plataforma implica la capacidad de ejecutar el código de un **CGI** en distinto *hardware* o *software* (sistema operativo) sin tener que modificarlo. La mejor forma de lograrlo es por medio de un "lenguaje universal" y no empleando código específico del sistema (llamadas al sistema operativo, por ejemplo).

Esto se traduce en el uso de lenguajes como *C* y *Perl*, que están disponibles prácticamente en cualquier plataforma. Si se tienen que emplear llamadas al sistema operativo, es conveniente aislar el código que las realiza en módulos independientes, de forma que al trasladar el código de una plataforma a otra se minimizan y facilitan los cambios necesarios.

1.5.2. Independencia de servidor

La independencia de servidor significa que el código se puede ejecutar en distintos servidores web sobre el mismo sistema operativo sin tener que modificarlo. Esta independencia es más sencilla de conseguir que la anterior, pero hay que observar una serie de recomendaciones:

- No asumir que el programa se ejecutará en un directorio concreto.
- No asumir que algunos directorios se hallan siempre en la misma ruta. Por ejemplo, suponer que el directorio temporal se encuentra siempre en

`C:\TEMP` o que el directorio principal del servidor web es `C:\INETPUB\-WWWROOT` es muy peligroso.

- No asumir que el programa se va a ejecutar con unos permisos (privilegios) concretos.
- No asumir la existencia de configuraciones de red concretas: direcciones **IP**, dominios, etc.
- No asumir la presencia de programas externos, como por ejemplo, suponer que está disponible en cualquier instalación el programa sendmail de Unix.

Si deseamos distribuir un programa **CGI** que hemos desarrollado, para evitar todos estos problemas, la mejor solución es proporcionar al usuario la posibilidad de configurar los valores dependientes del servidor mediante un fichero de configuración.

1.6. Razones para emplear CGI

En los primeros años de la era web (1992-1997), **CGI** era la única posibilidad que se tenía de añadir interactividad y dinamismo a los sitios web. Pero desde entonces han surgido distintas soluciones que sustituyen completamente este estándar. Entonces, ¿por qué seguir usando **CGI**? Existen diversas razones:

1. **CGI** es el método más rápido cuando se ejecuta mucho código. Sin embargo, cuando el código que se tiene que ejecutar es pequeño y poco complejo, las páginas activas como *Active Server Pages* (**ASP**), *Java Server Pages* (**JSP**) o PHP son la mejor solución, debido a la sobrecarga que supone ejecutar una aplicación externa al servidor web.

2. **CGI** es un estándar, compatible con la mayoría (por no decir la totalidad) de los servidores web. Podemos crear un programa **CGI** que se ejecute en distintos servidores web en distintas plataformas.

3. **CGI** es un estándar compatible con todos los clientes web.

4. Un programa **CGI** se puede escribir prácticamente en cualquier lenguaje. Por tanto, si se conoce un lenguaje de programación, se puede escribir un **CGI** desde el primer día.

5. Cómo es una tecnología establecida y probada (es decir, "antigua"), existen multitud de recursos, tales como tutoriales, programas **CGI** gratuitos, librerías, etc. La mayoría de los problemas que nos pueden surgir ya han sido resuelto y sólo hay que buscar qué soluciones se han planteado y cuál es la mejor.

1.7. Razones para no emplear CGI

Como se ha comentado en el apartado anterior, existen una serie de ventajas a la hora de emplear **CGI**. Sin embargo, el estándar **CGI** también tiene sus inconvenientes:

1. **CGI** es una tecnología obsoleta. Desde su nacimiento, han surgido otras posibilidades: *applets*, *servlets*, **ASP**, ColdFusion, **JSP**, PHP, etc.

2. **CGI** no mantiene el estado automáticamente[5]. Otras tecnologías (**ASP**, por ejemplo) mantienen el estado, lo que facilita la programación de aplicaciones web como "carritos de la compra" (*market cart)* o lectores de correo a través de la web (*webmail*). Para resolver esta carencia, se suelen emplear los campos ocultos de los formularios[6]: en ellos se almacenan las selecciones del usuario o un identificar único (*id*) que permite seguir su actividad de una página a otra.

3. La integración entre un programa **CGI** y el servidor web es muy débil. La única comunicación que se establece entre ambos es para transmitir los datos de entrada y la salida producida por el programa[7].

[5] Está es una limitación que se debe realmente al protocolo **HTTP** y no a **CGI**. El protocolo **HTTP** es un protocolo "sin estado" (*stateless*): cada vez que un cliente solicita un recurso (una página **HTML**, por ejemplo) al servidor web, es como si fuera la primera vez que lo hace. Entre las distintas peticiones no se almacena ningún tipo de información sobre el cliente en el servidor.

[6] `<INPUT TYPE="HIDDEN">`.

[7] Esto es una desventaja y a la vez una ventaja: gracias a que la integración es tan débil, un programa **CGI** bien hecho es independiente de la plataforma y se puede usar sin problemas en distintos servidores web.

4. Cada vez que se tiene que ejecutar un programa **CGI**, se crea una instancia nueva del programa en memoria.

1.8. El primer CGI

La salida o resultado que produce un programa **CGI** se tiene que dirigir a la salida estándar (`stdout`). Un programa **CGI** puede devolver cualquier tipo de documento. Cada documento que un **CGI** envía a un servidor web debe contener una cabecera (también llamado encabezado **HTTP**) al principio del mismo que indica el tipo de documento que es y así tanto el servidor como el cliente web[8] lo pueden procesar adecuadamente. El tipo del documento se expresa mediante los tipo **MIME**. Los tipos **MIME** básicos (`text`, `multipart`, `message`, `application`, `image`, `audio`, `video`) se dividen en subtipos. En el Cuadro 1.3 se muestran los tipos **MIME** más comunes y las extensiones asociadas a esos tipos.

La cabecera de la respuesta se compone de una serie de líneas con texto *American Standard Code for Information Interchange* (**ASCII**) separadas entre sí por saltos de línea. Muy importante: al final de la cabecera se tiene que dejar una línea en blanco[9], que indica donde termina la cabecera y empieza el cuerpo del mensaje de respuesta. A continuación viene el cuerpo de la respuesta, que puede estar en cualquier formato (texto **ASCII**, formato binario para una imagen, archivo de sonido, etc.).

Por ejemplo, la salida que tiene que generar un programa **CGI** para enviar una página **HTML** sencilla con la frase `¡Hola mundo!` es:

Ejemplo 1.1

```
1 Content-type: text/html
2
3 <HTML>
4 <BODY>
5 ¡Hola mundo!
```

[8] Los navegadores web usan los tipos *Multipurpose Internet Mail Extensions* (**MIME**) para saber con que programa tienen que mostrar un documento que no pueden tratar directamente (por ejemplo, un documento de Microsoft Word). Estos programas pueden ser externos al navegador o estar incluidos en él en forma de *plug-ins*.

[9] La línea en blanco se puede indicar con un salto de línea (`LF`) o con un retorno de carro y un salto de línea (`CR + LF`).

Tipo	Extensión
application/msword	doc
application/octet-stream	bin exe
application/pdf	pdf
application/x-shockwave-flash	swf
audio/midi audio/x-midi	midi mid
image/gif	gif
image/jpeg	jpeg jpe jpg
text/html	html htm
text/plain	txt
text/richtext	rtx
text/vnd.wap.wml	wml
text/xml	xml xsl
video/mpeg	mpeg mpg mpe
video/quicktime	qt mov
video/msvideo video/x-msvideo	avi

Cuadro 1.3: Tipos MIME más comunes

```
6 </BODY>
7 </HTML>
```

En la primera línea se indica el tipo **MIME** del contenido de la respuesta. El formato que se emplea es `Content-type: tipo MIME`. En este caso, como se trata de una página **HTML** empleamos `text/html`. A continuación, como la cabecera de la respuesta ha terminado, se tiene que dejar una línea en blanco. Por último, se incluye el contenido de la respuesta. El siguiente **CGI** programado en *C* genera como salida la página **HTML** anterior[10]. La línea en blanco que separa la cabecera del cuerpo de la respuesta se crea mediante la instrucción `printf("\n");` de la línea 6; esta instrucción se ha dejado en una línea sola a propósito.

Ejemplo 1.2

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5   printf("Content-type: text/html\n");
6   printf("\n");
7   printf("<HTML>\n<BODY>\n");
8   printf("¡Hola mundo!\n");
9   printf("</BODY>\n</HTML>\n");
10  return 0;
11 }
```

Por otro lado, no confundir el salto de línea `\n` con la instrucción salto de línea `<BR>` del código **HTML**. Las tres instrucciones que generan el código de la respuesta se pueden resumir en una sola sin ningún salto de línea:

Ejemplo 1.3

```
1 printf("<HTML><BODY>¡Hola mundo!</BODY></HTML>");
```

[10] Para generar la salida, se puede emplear la instrucción `printf(...)` o `fprintf(stdout, ...)`.

Los saltos de línea los incluimos para facilitar la lectura del código **HTML** si lo visualizamos directamente desde el navegador[11].

Otra posibilidad que se ofrece es redirigir (*redirect*) la respuesta a otra página[12]. En vez de generar el documento de salida, se puede simplemente indicar al cliente web donde puede encontrarlo. Para ello se emplea el formato `Location: URL`, donde `URL` puede ser una dirección de cualquier tipo (absoluta, relativa, a otro servidor, etc.). Por ejemplo, el siguiente programa **CGI** en *C* redirige la respuesta a la dirección `http://www.ua.es`.

Ejemplo 1.4

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5    printf("Content-type: text/html\n");
6    printf("Location: http://www.ua.es\n");
7    printf("\n");
8    printf("<HTML>\n<BODY>\n");
9    printf("Nueva dirección: ");
10   printf("<A HREF=\"http://www.ua.es\">http://www.ua.es</A>\n");
11   printf("</BODY>\n</HTML>\n");
12   return 0;
13 }
```

Algunos navegadores antiguos no aceptan la redirección (no reconocen la instrucción `Location`). Por ello, es conveniente, tal como se ha hecho en el código anterior, incluir la posibilidad de que los navegadores antiguos también puedan acceder a la información (¡aunque de forma manual a través de un enlace!). Si no se quiere "dar soporte" a los navegadores antiguos, el código anterior se puede reducir al siguiente:

Ejemplo 1.5

```
1  #include <stdio.h>
2
```

[11] Netscape Communicator: botón derecho del ratón y elegir **View Source**; Microsoft Internet Explorer: botón derecho del ratón y seleccionar **Ver código fuente**. También se puede acceder a través de los menús.

[12] Aunque parezca una posibilidad poco útil, se puede aprovechar esta posibilidad para mantener un registro de los enlaces que selecciona un usuario. También se puede emplear para redireccionar de forma aleatoria.

```
3 int main(int argc, char *argv[])
4 {
5   printf("Location: http://www.ua.es\n");
6   printf("\n");
7   return 0;
8 }
```

Aunque parezca repetitivo, hay que recordar siempre dejar una línea en blanco al final de la cabecera, incluso aunque no haya cuerpo. En la Figura 1.2 se muestra el mensaje de error que muestra el navegador Microsoft Internet Explorer 5.5 cuando en el programa **CGI** anterior se elimina la línea `printf("\n");` de la línea 6.

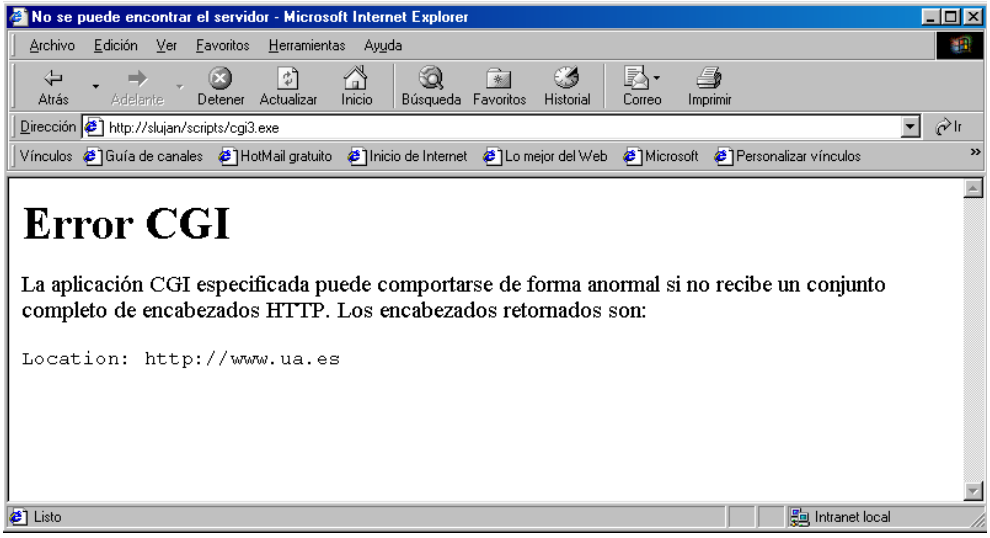

Figura 1.2: Mensaje de error porque el encabezado no es correcto

Existe una última directiva que permite a un programa **CGI** comunicar un código y mensaje de error. Para ello, se emplea la instrucción `Status: nnn xxxxx`, donde `nnn` es un código de estado de tres dígitos y `xxxxx` es un mensaje de error. En el Cuadro 1.4 mostramos algunos de los códigos más usuales.

Código	Resultado	Descripción
200	OK	Ningún problema
202	Accepted	La petición se está procesando, pero ha sido aceptada
204	No Response	El servidor no desea enviar ninguna respuesta
301	Moved	El documento se ha trasladado a un nuevo sitio
302	Found	El documento no está donde se esperaba, pero se ha encontrado en algún otro sitio en el servidor
400	Bad Request	La sintaxis de la petición HTTP no es correcta
401	Unauthorized	El documento requiere unos permisos que no posee el usuario
403	Forbidden	El servidor deniega el acceso al documento
404	Not Found	El servidor no puede encontrar el documento
500	Server Error	El servidor ha generado un error
502	Service Overloaded	El servidor está muy ocupado y no puede servir la petición

Cuadro 1.4: Códigos de estado HTTP más usuales

1.9. Cómo comunicarse directamente con el cliente

Cuando el programa **CGI** envía su salida al servidor web, éste le añade las instrucciones necesarias para formar un mensaje **HTTP** correcto. En algunos casos, se puede querer evitar esta sobrecarga y "hablar" directamente con el cliente web. En este caso, el programa **CGI** es el responsable de crear un mensaje **HTTP** correcto.

Para que el servidor web sepa distinguir unos programas **CGI** de otros, cuando se desee hablar directamente con el cliente, el nombre del programa **CGI** debe comenzar por `nph-`[13]. Por ejemplo, las siguientes instrucciones representan un mensaje **HTTP** correcto:

Ejemplo 1.6

```
1 HTTP/1.0 200 OK
2 Server: IIS/4.0
3 Content-type: text/html
4
5 <HTML><BODY>
6 Esto es un mensaje HTTP correcto
7 </BODY></HTML>
```

1.10. Cómo envía el servidor información a un CGI

Un programa **CGI** puede recibir información desde un servidor web de cuatro formas distintas:

- A través de la línea de comandos (*command line*).
- A través de la **URL** (`QUERY_STRING`).
- A través de la entrada estándar (`stdin`).
- A través de información de ruta (`PATH_INFO`).

Un programa **CGI** tiene que saber como va a recibir la información, ya que en cada caso tiene que actuar de distinta forma. Los dos métodos más populares son a través de la **URL** (también llamado método `GET`) y a través de la entrada estándar (método `POST`).

[13] *No Parse Header*: no se debe analizar la cabecera.

1.10.1. A través de la línea de comandos

La línea de comandos se emplea únicamente en el caso de una búsqueda `ISINDEX`. En estas consultas, el programa **CGI** recibe una lista de términos separados por espacios en blanco. Esta lista se recibe de dos formas:

- Por la línea de comandos: cada término es un argumento de la línea de comandos. Además, los términos se encuentran descodificados.
- Por la `QUERY_STRING`: el servidor crea una variable de entorno[14] llamada `QUERY_STRING` y le asigna una cadena que contiene los términos de la búsqueda. La cadena no se encuentra descodificada.

Desde el cliente web, se puede enviar una petición de consulta `ISINDEX` al servidor web de dos formas: mediante el uso de la etiqueta `ISINDEX` o directamente en la **URL**.

La sintaxis de la etiqueta `ISINDEX` es:

Ejemplo 1.7

```
1 <ISINDEX PROMPT="texto">
```

donde `texto` es el texto que acompaña al cuadro de texto que el usuario puede emplear para introducir términos de búsqueda. Esta etiqueta tiene que emplearse en la cabecera del documento **HTML** (`<HEAD> ... </HEAD>`). En la Figura 1.4 podemos ver como el siguiente código **HTML** que contiene esta etiqueta se muestra en un navegador:

Ejemplo 1.8

```
1  <HTML>
2  <HEAD>
3  <ISINDEX PROMPT="Ciudad de residencia ">
4  </HEAD>
5  <BODY>
6  Introduzca el nombre de la ciudad donde reside. El sistema
7  le informará sobre el estado del tiempo.<BR>
8  Posibles ciudades:
9  Alicante, Valencia, Castellón
10 </BODY>
11 </HTML>
```

[14] Más adelante se explican las variables de entorno CGI.

Como se puede observar, en ninguna parte se indica el programa **CGI** que se tiene que ejecutar cuando el cliente realice una consulta (que se realizará cuando el usuario pulse la tecla **Enter** (↩) y el foco esté situado en el cuadro de texto que representa la etiqueta `ISINDEX`). La página **HTML** se va a llamar así misma, así que para que haya procesamiento de algún modo, el código anterior lo tiene que haber generado previamente un programa **CGI**.

El siguiente código en *C* es un programa **CGI** que muestra la primera vez que se ejecuta el documento **HTML** de la Figura 1.4. En la Figura 1.3 podemos ver la salida que produce este programa cuando se ejecuta directamente desde una ventana de MS-DOS.

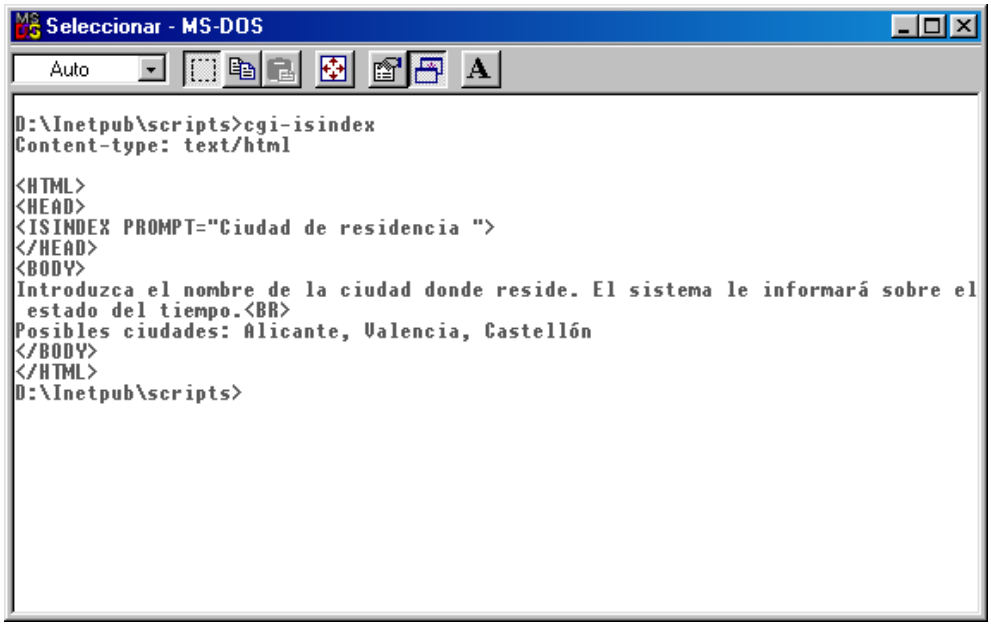

Figura 1.3: Ejecución desde una ventana de MS-DOS

Cuando se realiza una consulta, se vuelve a ejecutar el programa **CGI** y éste detecta que se le pasa alguna información a través de la línea de comandos: en el código del programa se puede observar como se consulta la variable `argc` en la línea 9 para saber si se han recibido parámetros a través de la línea de comandos. Por ejemplo, en la Figura 1.5 podemos ver la página que se genera cuando se introduce en el campo de entrada la cadena `Alicante`.

Ejemplo 1.9

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  int i;

  if(argc == 1)
  {
    printf("Content-type: text/html\n\n");
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<ISINDEX PROMPT=\"Ciudad de residencia \">\n");
    printf("</HEAD>\n");
    printf("<BODY>\n");
    printf("Introduzca el nombre de la ciudad donde reside. ");
    printf("El sistema le informará sobre el estado del tiempo.");
    printf("<BR>\n");
    printf("Posibles ciudades: Alicante, Valencia, Castellón\n");
    printf("</BODY>\n</HTML>");
  }
  else
  {
    printf("Content-type: text/html\n\n");
    printf("<HTML>\n<BODY>\n");
    if(!strcmp(argv[1], "Alicante"))
    {
      printf("<CENTER><IMG SRC=\"nubes.gif\"></CENTER>\n");
      printf("Cielo nublado. ");
      printf("Posibilidad de precipitación al anochecer.");
    }
    else if(!strcmp(argv[1], "Valencia"))
    {
      printf("<CENTER><IMG SRC=\"sol.gif\"></CENTER>\n");
      printf("Cielo despejado. ");
      printf("Vientos de aire caliente procedentes de levante.");
    }
    else if(!strcmp(argv[1], "Castellón"))
    {
      printf("<CENTER><IMG SRC=\"lluvias.gif\"></CENTER>\n");
```

```
42        printf("Lluvias durante todo el día. ");
43        printf("Riesgo alto de granizo y nieve.");
44      }
45      else
46        printf("El nombre de ciudad <I>%s</I> no es correcto", argv[1]);
47      printf("</BODY>\n</HTML>\n");
48    }
49
50    return 0;
51  }
```

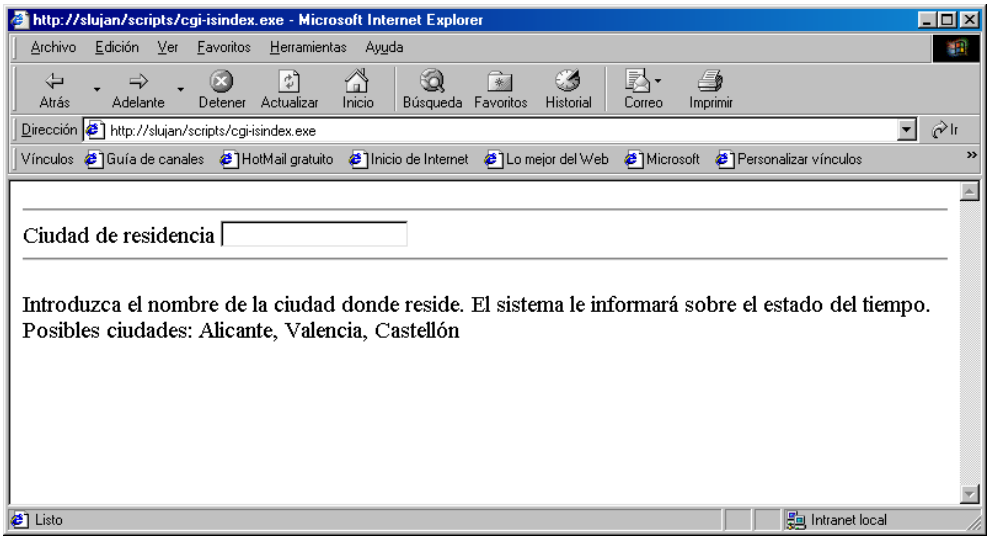

Figura 1.4: Página con cuadro de texto ISINDEX para realizar una búsqueda

Otra forma de realizar consultas `ISINDEX` es directamente a través de la **URL** (de forma manual). Cuando se llama a un programa **CGI** (por ejemplo, en un enlace) se pueden añadir términos de búsqueda a continuación del nombre del **CGI**: separado por el signo interrogación (`?`), se escriben los términos de búsqueda. Si hay más de uno, se tienen que separar por un signo más (`+`)[15].

[15] Más adelante veremos que el signo más se emplea para codificar los espacios en blanco en la **URL**.

Figura 1.5: Página de respuesta a una búsqueda ISINDEX

Muy importante: si los términos de búsqueda contienen un signo igual (=), entonces no se realizará una consulta `ISINDEX` y la información no se pasará por la línea de comandos. Esto no ocurre si el signo igual se escribe en el cuadro de texto de una etiqueta `ISINDEX`, ya que el navegador se encarga de codificarlo[16].

1.10.2. Cómo tratar los formularios

Los dos siguientes métodos (a través de la **URL** y a través de la entrada estándar) permiten que un programa **CGI** reciba los datos introducidos por el usuario en los controles de un formulario. Pero antes de estudiar esos dos métodos, hay que saber que el navegador codifica automáticamente la entrada del usuario cuando la envía al servidor web. Los datos introducidos en un formulario se envían al programa **CGI** con el siguiente formato:

Ejemplo 1.10

```
1 control1=valor1&control2=valor2&...&controln=valorn
```

[16] El código del signo igual es `%3D`.

donde `control1, control2, ..., controln` son los distintos nombres de los controles que forman el formulario y `valor1, valor2, ..., valorn` son los distintos valores que ha introducido o seleccionado el usuario y las distintas parejas `control=valor` se separan mediante *ampersand* (`&`). Por ejemplo, a partir de un formulario con tres controles se puede obtener una entrada como la siguiente:

Ejemplo 1.11

```
1 nombre=Jose&universidad=UA&carrera=Derecho
```

Si el usuario no ha especificado un determinado valor en algún control, aparecerá de todas formas la correspondiente cadena `control=`, sin ningún valor asociado.

Otro aspecto importante es que en los valores introducidos por el usuario, los espacios en blanco se sustituyen por el signo `+` y si aparecen caracteres especiales, como por ejemplo "`&`", "`%`", "`$`" o "`ñ`", se codifican usando el símbolo "`%`" seguido de dos dígitos que expresan, en hexadecimal, su código **ASCII** [17]. Esta codificación se conoce como "codificación URL" (*URL encoding* o *escaping*). Por ejemplo, la cadena "`&%$ñ`" se codificaría como "`%26%25%24%F1`". Se emplea esta codificación de los datos de entrada para evitar una interpretación accidental de caracteres especiales por parte del sistema operativo, lo que podría originar un agujero de seguridad.

Por tanto, el programa **CGI** debe realizar la descodificación de la entrada "antes de poder hacer nada":

1. Tiene que separar las distintas parejas `control=valor`. Para ello, hay que dividir los datos recibidos cada vez que se encuentre un *ampersand* (`&`). No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un *ampersand*, se envía codificado (`%26`).

2. Una vez que se tienen las distintas parejas, se separan en nombre de control y valor de control usando para ello el signo igual (`=`). No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un signo igual, se envía codificado (`%3D`).

[17] Se codifican los caracteres con un código **ASCII** menor de 33 (`21` hexadecimal) o mayor que 127 (`7F` hexadecimal). El espacio en blanco podría codificarse como `%20`, pero como el espacio en blanco es tan común, se ahorra espacio y es "más elegante emplear" el signo más (`+`).

3. Los distintos valores se descodifican. Se substituyen los signos más por espacios en blanco y se buscan cadenas de la forma `%##`, donde `##` son códigos hexadecimales. No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un signo porcentaje, se envía codificado (`%25`).

En el Cuadro 1.5 se han resumido los caracteres especiales que se emplean en la codificación **URL**.

Nombre	Carácter	Propósito
Ampersand	`&`	Separa pares `control=valor`
Equal	`=`	Separa el nombre del control del valor del control
Percent	`%`	Marca el inicio de un carácter codificado
Plus	`+`	Substituye espacios en blanco

Cuadro 1.5: Caracteres especiales en la codificación URL

1.10.3. A través de la URL

Este método se emplea cuando se usa un formulario[18] con el método de envío `GET` o directamente a través de la **URL**. El programa **CGI** recibe la información codificada a través de la `QUERY_STRING`. El navegador se encarga de codificar la información que introduce el usuario en el formulario. Por tanto, si usamos el método directo (directamente escrito en una **URL**), tenemos que codificar manualmente los datos.

Cuando se usa este método directamente a través de la **URL**, los datos que se quieren enviar se añaden al final de la **URL**, separados del nombre del programa **CGI** mediante un signo de interrogación (`?`). Por ejemplo, si queremos que al pulsar sobre un enlace se llame al programa `cgi.exe` y se le pase la palabra `subtotal`, pondremos:

Ejemplo 1.12

```
1 <A HREF="cgi.exe?subtotal">Ver subtotal</A>
```

[18] En la mayoría de los navegadores, el método de envío por defecto es `GET`: si en un formulario no se indica el método con el atributo `METHOD`, se asume el método `GET`.

También existe el método `HEAD`, similar al método `GET`, excepto que con el método `HEAD` sólo las cabeceras **HTTP** (y no el cuerpo del mensaje) se envían desde el servidor web hacia el navegador.

1.10.4. A través de la entrada estándar

Este método se emplea cuando se usa un formulario con el método de envío `POST`. El programa **CGI** recibe la información codificada a través de la entrada estándar (`stdin`) (el navegador se encarga de codificar la información que introduce el usuario en el formulario).

El servidor web no tiene la obligación de enviar una marca de final de fichero (`EOF`) al final de los datos. Para saber cuántos datos hay que leer de la entrada, se tiene que consultar la variable de entorno `CONTENT_LENGTH`, que proporciona el número de bytes que se pueden leer. El servidor web también informa sobre el tipo de datos que va a recibir el programa **CGI** mediante la variable de entorno `CONTENT_TYPE`. La codificación estándar para los datos de un formulario es `application/x-www-form-urlencoded`.

Cuando se emplea este método, la variable de entorno `QUERY_STRING` está vacía, a no ser que después del nombre del programa **CGI** aparezca un signo de interrogación (`?`) y algo más. Por ejemplo, en el siguiente formulario, se envía la entrada del usuario mediante `POST`, pero también se pasa información a través de la **URL**:

Ejemplo 1.13

```
1 <FORM ACTION="cgi.exe?id=es" METHOD="POST">
2 Nombre: <INPUT TYPE="TEXT" NAME="nombre">
3 </FORM>
```

La ventaja principal del método `POST` sobre el método `GET` es que el primero no tiene ninguna limitación sobre el número de bytes que se pueden enviar, mientras que el segundo, como los datos se envían en la **URL** y la información se almacena en la variable de entorno `QUERY_STRING`, puede verse limitado por el tamaño máximo que pueda tener una **URL** (1024 bytes normalmente) o por el tamaño máximo de una variable de entorno en el sistema operativo.

Un programa **CGI** puede saber si se le han enviado los datos mediante `GET` o `POST` consultando la variable de entorno `REQUEST_METHOD`.

1.10.5. A través de información de ruta

También existe otra forma de enviar datos al programa **CGI** desde el cliente a través de la **URL**, incluyendo información extra en la vía de acceso al programa **CGI**. Esta información adicional no se codifica de ninguna manera. En este caso, el programa **CGI** recibe la información extra en la variable de entorno `PATH_INFO`.

Esta forma de enviar información se emplea normalmente para transmitir la localización de ficheros al programa **CGI**, aunque se puede emplear para otros usos. Por ejemplo, imaginemos que tenemos un **CGI** llamado `cgi-orden.exe` que es capaz de ordenar las líneas de un fichero y mostrar el resultado en una página **HTML**. Si queremos que procese el fichero `lista.txt` que se encuentra en el directorio ficheros que pertenece al directorio principal del sitio web, se tiene que realizar la llamada al programa **CGI** de esta forma:

Ejemplo 1.14

```
1 cgi-orden.exe/ficheros/lista.txt.
```

1.11. Variables de entorno CGI

Además de las variables de entorno que hemos visto (`CONTENT_LENGTH`, `CONTENT_TYPE`, `PATH_INFO`, `QUERY_STRING` y `REQUEST_METHOD`), el servidor web asigna valor a otras variables cuando ejecuta el programa **CGI**. A continuación se muestran las variables más importantes agrupadas en tres grupos: específicas del servidor, específicas del cliente y específicas de la petición. Algunas variables puede ser que no estén disponibles en algunos servidores web. Por otro lado, además de a todas estas variables de entorno específicas de **CGI**, también podemos acceder a las pertenecientes al sistema operativo, como `PATH`, `TEMP`, etc.

Cada programa **CGI** recibe sus propias variables de entorno con sus propios valores. Se pueden ejecutar concurrentemente varios programas sin problemas, ya que cada uno recibirá su propia copia de las variables de entorno.

1.11.1. Específicas del servidor

Estas variables comunican al programa **CGI** características sobre el servidor web en que se está ejecutando. Normalmente, se sabe en que servidor se

está ejecutando un programa **CGI**, así que estás variables se suelen usar poco.

- `GATEWAY_INTERFACE`. El nombre y la versión de la especificación **CGI** utilizada por el servidor. El formato es CGI/versión. Ejemplo: `CGI/1.1`.
- `SERVER_NAME`. El nombre del servidor, el alias *Domain Name System* (**DNS**) o la dirección **IP** tal como aparecería en las direcciones **URL** que hacen referencia a sí mismas. Ejemplo: `www.ua.es`.
- `SERVER_PORT`. El número de puerto en el que el servidor ha recibido la petición **HTTP**. Ejemplo: `80`[19].
- `SERVER_PROTOCOL`. El nombre y la versión del protocolo empleado por el servidor para procesar las peticiones. El formato es protocolo/versión. Ejemplo: `HTTP/1.1`.
- `SERVER_SOFTWARE`. El nombre y la versión del software del servidor que responde a la petición y que ejecuta el **CGI**. El formato es nombre/versión. Ejemplo: `Microsoft-IIS/4.0`.

1.11.2. Específicas del cliente

Mediante estas variables, el servidor web informa al programa **CGI** sobre el cliente web (navegador). El servidor web obtiene la información a partir de las cabeceras que envía un cliente web en una petición (por ello, todas las variables comienzan por **HTTP**, ya que el contenido de estas variables se recibe con cada petición **HTTP**[20]). No todos los clientes web proporcionan toda la información posible.

- `HTTP_ACCEPT`. Enumera los tipos de respuesta que acepta el cliente. El formato es tipo/subtipo, tipo/subtipo, ... Ejemplo: `image/gif, image/-x-xbitmap, image/jpeg, image/pjpeg, image/png, */*`.
- `HTTP_ACCEPT_ENCODING`. Identifica los tipos de esquemas de codificación que acepta el cliente. Ejemplo: `gzip, deflate`.

[19] El puerto 80 es el puerto por defecto para comunicaciones **HTTP**, pero puede ser cambiado.

[20] En la cabecera **HTTP**, el signo de subrayado _ de los nombres de las variables de entorno específicas del cliente aparece realmente como un guión -. Además, todos los caracteres se han pasado a mayúsculas en el nombre de la variable de entorno.

- `HTTP_ACCEPT_LANGUAGE`. Enumera los códigos *International Organization for Standards* (**ISO**) de los lenguajes que el cliente entiende y espera recibir. Ejemplo: `es-ES,en,pdf`.

- `HTTP_REFERER`. Identifica la URL del documento que contiene el enlace que apunta al documento actual. Ejemplo: `http://www.ua.es/-index.html`.

- `HTTP_USER_AGENT`. Identifica el software del cliente web. Ejemplo: para Netscape Communicator 4.7 se obtiene `Mozilla/4.7 [en] (Win98; I)` y para Microsoft Internet Explorer 5.5 la cadena `Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)`.

1.11.3. Específicas de la petición

- `AUTH_TYPE`. El método de autenticación que el servidor utiliza para validar a los usuarios cuando intentan tener acceso a un programa **CGI** protegido. Normalmente la autenticación se realiza mediante un nombre de usuario y una contraseña. Ejemplo: `BASIC`.

- `AUTH_USER`. Nombre de usuario autenticado.

- `CONTENT_LENGTH`. Número de bytes enviados a la entrada estándar (`stdin`) de un programa **CGI** debido a una petición `POST`. Esta variable está vacía cuando el método empleado es `GET`.

- `CONTENT_TYPE`. El tipo **MIME** de los datos enviados por el cliente web mediante el método `POST`. Esta variable está vacía cuando el método empleado es `GET`. Ejemplo: `application/x-www-form-urlencoded`.

- `PATH_INFO`. Información adicional de ruta para el programa **CGI** pasada como parte de la **URL**, a continuación del nombre del programa. Ejemplo: `/myhome`.

- `PATH_TRANSLATED`. La versión traducida de `PATH_INFO`. La ruta virtual se convierte en ruta física. Ejemplo: `D:\Inetpub\wwwroot\myhome`.

- `QUERY_STRING`. Información de consulta almacenada en la cadena que sigue al signo de interrogación (`?`) en la **URL**.

- `REMOTE_ADDR`. La dirección **IP** del cliente web que hace la petición. Ejemplo: `156.78.65.9`.

- `REMOTE_HOST`. El nombre de host del cliente que realiza la petición. Si el servidor no posee esta información, debe fijar el valor de `REMOTE_ADDR` y dejar esta variable en blanco.

- `REMOTE_USER`. Nombre del usuario remoto, si el usuario se ha autenticado correctamente.

- `REQUEST_METHOD`. El método que se utiliza para hacer la petición. Los más usuales son `HEAD`, `GET` y `POST`.

- `SCRIPT_NAME`. La ruta virtual al programa **CGI** que se está ejecutando. Esta variable es útil en los programas **CGI** que se llaman a sí mismos[21]. Ejemplo: `/scripts/cgivar.exe`.

En la Figura 1.6 se muestra el valor de algunas de las variables de entorno **CGI** en un servidor Microsoft Personal Web Server 4.0 ejecutándose en Microsoft Windows 98 y cuando recibe una petición de un cliente Microsoft Internet Explorer 5.5.

1.11.4. Cómo acceder a las variables desde C

Para acceder a las variables de entorno desde *C* se puede emplear la función `getenv()` que se encuentra en la librería `stdlib.h`. El prototipo de la función es:

Ejemplo 1.15

```
1 char *getenv(const char *name);
```

Por ejemplo, el siguiente código muestra el valor de las variables específicas del servidor mostradas en la Figura 1.6:

[21] Por ejemplo, en los programas **CGI** que generan un formulario y también lo procesan cuando se envía. Mediante la variable `REQUEST_METHOD` se puede distinguir el primer caso (`GET`) del segundo (`POST`).

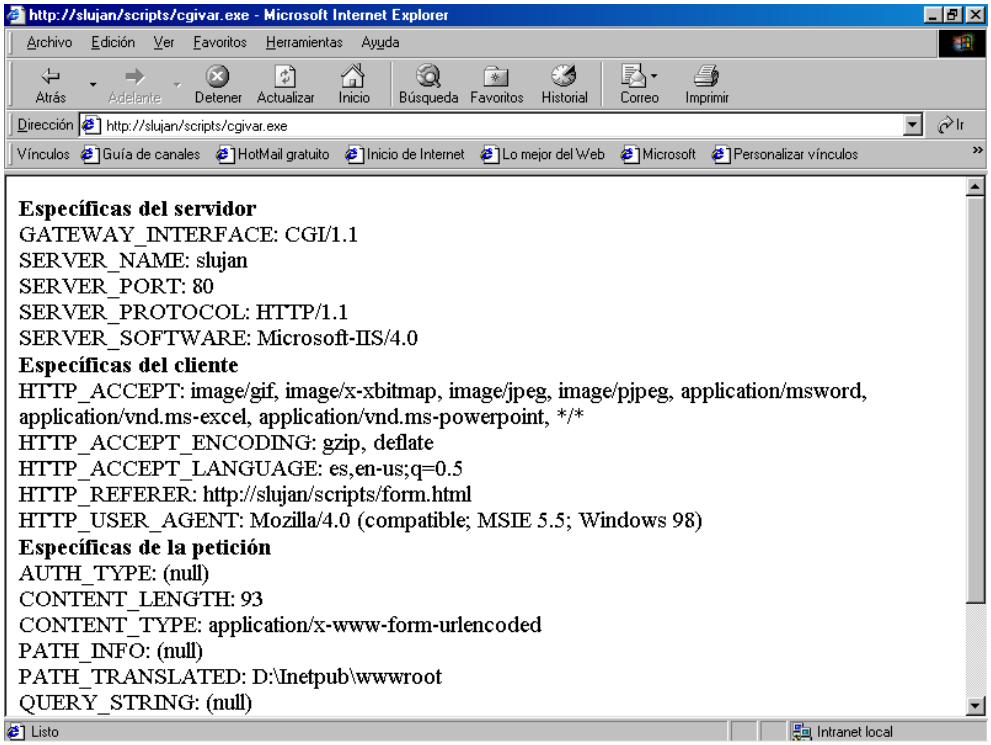

Figura 1.6: Ejemplo de variables de entorno

Ejemplo 1.16

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  char *variable;

  fprintf(stdout, "Content-type: text/html\n\n");
  fprintf(stdout, "<HTML>\n<BODY>\n");

  /* SERVIDOR */
  printf("<B>Específicas del servidor</B><BR>");

  variable = getenv("GATEWAY_INTERFACE");
  fprintf(stdout, "GATEWAY_INTERFACE: %s<BR>", variable);

  variable = getenv("SERVER_NAME");
  fprintf(stdout, "SERVER_NAME: %s<BR>", variable);

  variable = getenv("SERVER_PORT");
  fprintf(stdout, "SERVER_PORT: %s<BR>", variable);

  variable = getenv("SERVER_PROTOCOL");
  fprintf(stdout, "SERVER_PROTOCOL: %s<BR>", variable);

  variable = getenv("SERVER_SOFTWARE");
  fprintf(stdout, "SERVER_SOFTWARE: %s<BR>", variable);

  fprintf(stdout, "</BODY>\n</HTML>\n");
  return 0;
}
```

1.12. Un ejemplo más complejo

El siguiente ejemplo muestra un programa **CGI** más complejo. Se compone de dos ficheros: `cgi-select.c` que contiene el código del programa y `cgi.data` que contiene la información que emplea el programa para construir la página.

El programa genera dos páginas web. En la primera (Figura 1.7), se mues-

tra una lista desplegable que contiene una serie de valores leídos del fichero `cgi.data`. Una vez que se elige un valor de la lista, se vuelve a ejecutar el programa **CGI**, pero en la segunda página (Figura 1.8) se muestran dos listas: la que se mostraba antes y otra cuyos valores dependen del valor elegido en la primera lista.

cgi-select.c

Este fichero contiene el código en *C* del programa **CGI**.

Ejemplo 1.17

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  /* Construye las opciones de la primera lista */
6  void
7  optionsA(FILE *f)
8  {
9    char centinela = 1;
10   char linea[100];
11
12   while(centinela)
13   {
14     /* Fin si se llega al final del fichero */
15     if(fgets(linea, 100, f) == NULL)
16       centinela = 0;
17     else
18     {
19       if(linea[0] == '*' && linea[strlen(linea) - 2] == '*')
20       {
21         /* Elimina el salto de linea */
22         linea[strlen(linea) - 1] = '\0';
23         printf("<OPTION VALUE=\"%s\">", linea);
24         /* Elimina el ultimo asterisco */
25         linea[strlen(linea) - 1] = '\0';
26         /* +1: elimina el primer asterisco */
27         printf("%s</OPTION>\n", linea + 1);
28       }
29     }
30   }
```

```
31 }
32
33 /* Construye las opciones de la segunda lista */
34 void
35 optionsB(FILE *f, char *s)
36 {
37   char centinela = 1;
38   char linea[100];
39
40   while(centinela)
41   {
42     /* Fin si se llega al final del fichero */
43     if(fgets(linea, 100, f) == NULL)
44       centinela = 0;
45     else
46     {
47       if(!strncmp(s, linea, strlen(s)))
48       {
49         /* Lee todos los productos mientras que no
50            se llegue al final del fichero a una linea
51            con un salto de linea (fin de producto) */
52         while(centinela)
53         {
54           /* Fin si se llega al final del fichero */
55           if(fgets(linea, 100, f) == NULL)
56             centinela = 0;
57           else if(linea[0] == '\n')
58             centinela = 0;
59           else
60           {
61             /* Elimina el salto de linea */
62             linea[strlen(linea) - 1] = '\0';
63             printf("<OPTION>%s</OPTION>\n", linea);
64           }
65         }
66       }
67     }
68   }
69 }
70
71 int
72 main(void)
```

```
73 {
74   char *var;
75   char entrada[100];
76   int i, lon;
77   FILE *f;
78
79   f = fopen("cgi.data", "r");
80   if(f == NULL)
81   {
82     printf("Content-type: text/html\n\n");
83     printf("<HTML>\n");
84     printf("<HEAD>\n");
85     printf("<TITLE>CGI con listas desplegables - Error</TITLE>\n");
86     printf("</HEAD>\n");
87     printf("<BODY>\n");
88     printf("Error: no encuentro el fichero cgi.data\n");
89     printf("</BODY>\n</HTML>");
90
91     return 0;
92   }
93
94   var = getenv("REQUEST_METHOD");
95   if(!strcmp(var, "GET"))
96   {
97     printf("Content-type: text/html\n\n");
98     printf("<HTML>\n");
99     printf("<HEAD>\n");
100    printf("<TITLE>CGI con listas desplegables - Página 1</TITLE>\n");
101    printf("</HEAD>\n");
102    printf("<BODY>\n");
103    /* El formulario llama al propio CGI */
104    var = getenv("SCRIPT_NAME");
105    printf("<FORM ACTION=\"%s\" METHOD=\"POST\">\n", var);
106    printf("Seleccione sistema operativo:<BR>\n");
107    printf("<SELECT NAME=\"sistema\" ONCHANGE=\"submit();\">\n");
108    printf("<OPTION SELECTED></OPTION>\n");
109    optionsA(f);
110    printf("</SELECT>\n");
111    printf("</FORM>\n");
112    printf("</BODY>\n</HTML>");
113  }
114  else
```

```
115   {
116     /* Lee los datos recibidos por la entrada estándar */
117     var = getenv("CONTENT_LENGTH");
118     lon = atoi(var);
119     for(i = 0; i < 100 & i < lon; i++)
120       entrada[i] = fgetc(stdin);
121     entrada[i] = '\0';
122     printf("Content-type: text/html\n\n");
123     printf("<HTML>\n");
124     printf("<HEAD>\n");
125     printf("<TITLE>CGI con listas desplegables - Página 2</TITLE>\n");
126     printf("</HEAD>\n");
127     printf("<BODY>\n");
128     /* El formulario llama al propio CGI */
129     var = getenv("SCRIPT_NAME");
130     printf("<FORM ACTION=\"%s\" METHOD=\"POST\">\n", var);
131     printf("Seleccione sistema operativo:<BR>\n");
132     printf("<SELECT NAME=\"sistema\" ONCHANGE=\"submit();\">\n");
133     printf("<OPTION SELECTED></OPTION>\n");
134     optionsA(f);
135     printf("</SELECT>\n");
136     printf("<BR><BR>\n");
137     /* +8: elimina sistema= */
138     printf("Seleccione producto para %s:<BR>\n", entrada + 8);
139     printf("<SELECT>\n");
140     /* El puntero se posiciona en el principio del fichero */
141     fseek(f, 0, SEEK_SET);
142     /* +8: elimina sistema= */
143     optionsB(f, entrada + 8);
144     printf("</SELECT>\n");
145     printf("</FORM>\n");
146     printf("</BODY>\n</HTML>");
147   }
148
149   fclose(f);
150   return 0;
151 }
```

cgi.data

Este fichero contiene la información que se quiere mostrar en las listas. Cada valor tiene que escribirse en una línea independiente; los valores principales que se quieran mostrar en la primera lista tienen que aparecen encerrados entre asteriscos (*), a continuación se escriben los valores (segunda lista) correspondientes al valor principal. Muy importante: los valores principales no pueden contener espacios en blanco ni caracteres especiales. A continuación se muestra el fichero empleado en la Figura 1.7 y 1.8.

Ejemplo 1.18

```
1  *Windows95*
2  Actualización USB
3  Parche efecto 2000
4
5  *Windows98*
6  Parche problemas de apagado
7  Parche agujero de seguridad
8
9  *Windows2000*
10 Actualización Office 2000
11 Parche seguridad IIS
12 Internet Explorer 6.0
```

1.13. Seguridad

El estándar **CGI** no es inseguro por sí mismo: simplemente define un interfaz para que un servidor web se comunique con aplicaciones externas. Pero como un **CGI** es un programa ejecutable, al usarlo en nuestra web estamos permitiendo que "extraños" ejecuten un programa en nuestro servidor, lo cual no es lo más seguro del mundo. Por tanto, existen una serie de precauciones que hay que tener en cuenta a la hora de programar un **CGI**.

1.13.1. Permisos de ejecución

Lo primero que hay que saber es que para que se ejecute un **CGI**, éste tiene que residir en un directorio especial, de forma que el servidor web sepa que

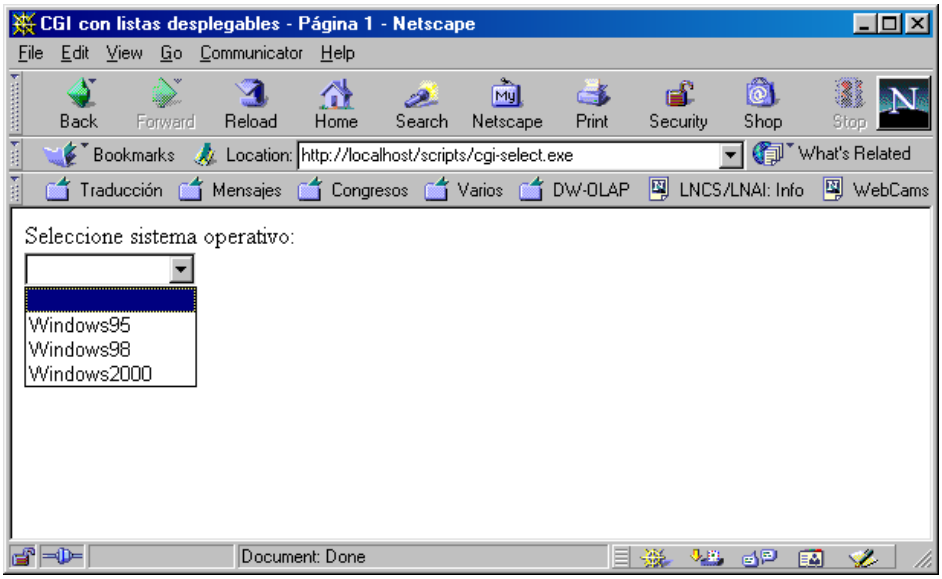

Figura 1.7: cgi-select: página 1

tiene que ejecutar el programa en vez de mostrarlo[22]. Además, de este modo, un usuario particular no puede colocar un programa **CGI** en su directorio particular sin que el administrador del sistema lo sepa y lo permita.

Normalmente, en los servidores web que se ejecutan en sistemas Unix, el directorio se llama `/cgi-bin` o `/cgibin`. En los dos servidores web de MICROSOFT, Microsoft Personal Web Server y Microsoft Internet Information Server, el directorio suele llamarse `/Scripts`. En la Figura 1.9 podemos ver los permisos que posee por defecto el directorio `D:\Inetpub\scripts` en el servidor Microsoft Personal Web Server 4.0. Como se puede apreciar, están activos los permisos **Ejecución** y **Archivos de comandos**. Para que se ejecute un **CGI**, sólo hace falta tener activado el permiso **Ejecución**.

[22] Si no fuera así, los usuarios podrían acceder y descargarse el **CGI**, lo que plantea un grave problema de seguridad.

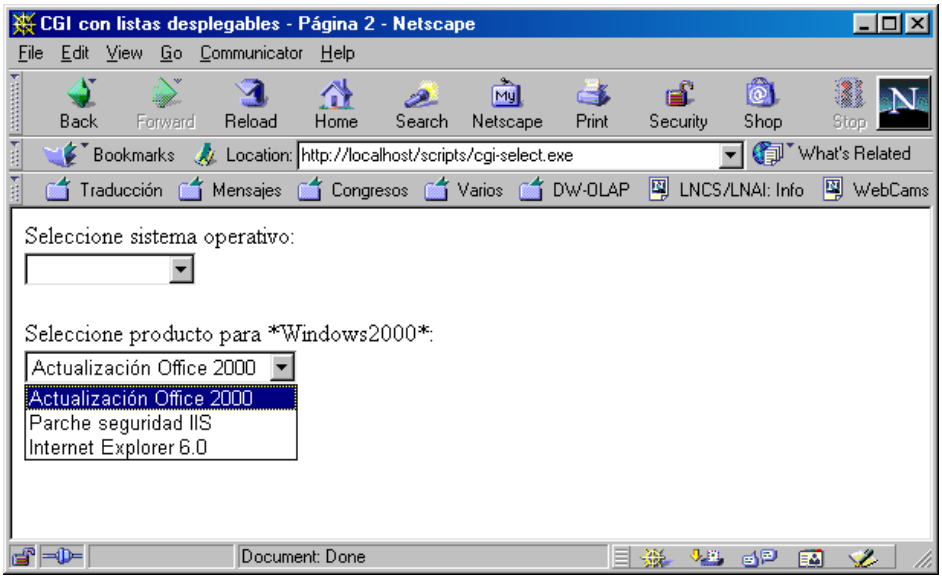

Figura 1.8: cgi-select: página 2

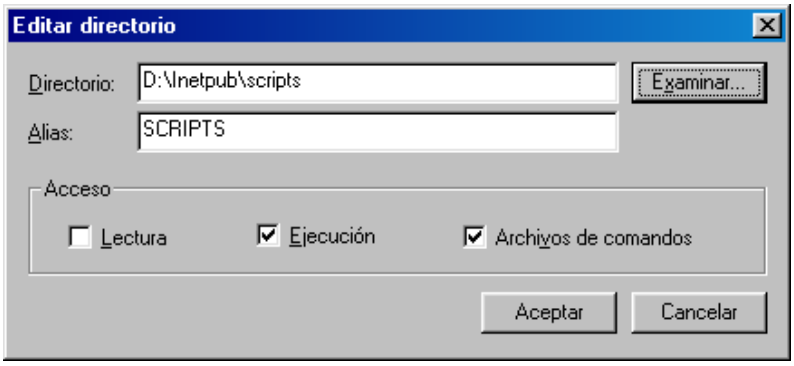

Figura 1.9: Permisos de ejecución en Microsoft Personal Web Server

1.13.2. Examina el código

Si se usa un **CGI** programado por otra persona, es conveniente revisar el código para comprobar qué hace y cómo lo hace realmente. No hay que fiarse de los programas **CGI** ya compilados: pueden esconder un "caballo de Troya"[23] o una puerta trasera (*backdoor*) de entrada a nuestro sistema. Por ejemplo, un programa **CGI** puede realizar una función dada "benigna", pero además, sin que nadie se entere, puede enviar a su creador el fichero `/etc/passwd` cada vez que se ejecute.

Este problema también se puede dar cuando usamos librerías de código de origen desconocido. Aunque sea una librería conocida, hay que descargarla de un sitio de confianza, para evitar que haya sido manipulada previamente.

1.13.3. Versiones estables

Siempre que sea posible, hay que emplear las últimas versiones estables de los programas empleados. Nada de versiones "beta", ya que suelen tener muchos problemas de vulnerabilidad, que además son publicados en Internet y conoce todo el mundo rápidamente.

1.13.4. Las presunciones son peligrosas

A la hora de tratar los datos de entrada del usuario, las presunciones son muy peligrosas:

- Suponer que los datos que se reciben provienen de nuestro formulario es un error. Cualquiera puede apuntar un formulario cualquiera a nuestro programa **CGI** o generar una petición **HTTP**[24] que parezca el resultado de un formulario, pero que contenga datos peligrosos.
- Es peligroso asumir que los datos que recibe el **CGI** se pueden almacenar correctamente. Cualquier limitación que se imponga en un formulario[25], se puede saltar fácilmente con un formulario distinto o con una

[23] Un caballo de Troya es una aplicación "maligna" que se camufla como un programa que realiza una función "benigna", pero que realmente realiza una serie de tareas ocultas sin que el usuario se de cuenta. Al contrario que los virus, los caballos de Troya no se replican ni infectan otros ficheros, pero pueden ser tan destructivos como ellos.

[24] Se puede crear un programa que haga cosas que no puede hacer un navegador, como enviar cientos de megabytes a un **CGI**.

[25] Por ejemplo, `<INPUT TYPE="TEXT" MAXLENGTH="10">`.

petición **HTTP** directa. El exceso de datos produce desbordamientos de buffers (*buffer overrun*), que pueden bloquear el sistema o permitir el acceso en modo superusuario (administrador) al sistema. Si se reciben los datos mediante `POST`, es conveniente verificar su longitud a través de `CONTENT_LENGTH`.

- También es un error suponer que los caracteres especiales en los datos han sido codificados por el navegador mediante las secuencias `%xx`.

En definitiva, un programa **CGI** tiene que estar preparado para esperar datos de entrada que contienen basura, están vacíos, son aleatorios o superan el tamaño máximo esperado. Evidentemente, tanta prevención tiene un inconveniente: el código del programa aumenta considerablemente y el mantenimiento futuro es más complicado.

1.13.5. Programa defensivamente

A la hora de tratar los datos recibidos, hay que elegir un criterio. Por ejemplo, si un campo de un formulario tiene que contener una dirección de correo electrónico, el programa **CGI** tiene que rechazar aquellos datos que no se ajusten a una dirección de correo electrónico.

Si un cuadro de texto tiene un límite de longitud (`MAXLENGTH`), los datos recibidos pueden superar dicho valor. El programa **CGI** debe verificar la longitud de los datos.

Si en un campo se espera una única línea de texto, el programa **CGI** tiene que rechazar aquellos datos que contengan un salto de línea (más de una línea).

Si un formulario incluye listas desplegables, cuadros de verificación o botones de radio, el programa **CGI** tiene que rechazar cualquier dato que no coincida con los presentados al usuario en el formulario.

1.13.6. Limpia los datos antes de usarlos

Los datos que introduce un usuario hay que limpiarlos o validarlos antes de emplearlos. Por ejemplo, si en un cuadro de texto se espera el nombre de un fichero, antes de realizar cualquier operación que suponga trabajar con un fichero hay que verificar que se trata de un nombre de fichero válido. Por ejemplo, verificar que[26]:

[26] Las verificaciones dependen del sistema de archivos de cada sistema operativo.

- No comienza por un punto.
- No contiene separadores de ruta (/ o \).
- No contiene dos puntos (:), subrayado (_) o cualquier otro carácter especial.
- Tiene una longitud máxima.

Para verificar que unos datos son válidos, se pueden tomar dos soluciones: verificar que contiene caracteres válidos o verificar que no contiene caracteres no válidos. Es preferible emplear la primera alternativa, ya que si se emplea la segunda es probable que se olvide comprobar algún carácter no válido.

El siguiente código en *C* permite validar el contenido de una cadena. Tiene como argumentos dos punteros a cadenas: la primera cadena contiene los datos de entrada que se quieren validar y la segunda los caracteres aceptados. La función devuelve un puntero a la primera cadena con los caracteres no aceptados eliminados.

Ejemplo 1.19

```
1  char *stripchars(char *cadena, const char *acepta)
2  {
3    char flags[256], *chr, *pos;
4    int n;
5
6    /* Tabla de flags que indica si un caracter es aceptado */
7    for(n = 0; n < 256; n++)
8      flags[n] = 0;
9    for(chr = acepta; *chr != '\0'; chr++)
10     flags[*chr] = 1;
11   /* Sobre la propia cadena, copia unicamente los caracteres
12      validos */
13   for(chr = cadena, pos = cadena; *chr != '\0'; chr++)
14   {
15     *pos = *chr;
16     pos += flags[*chr];
17   }
18   *pos = '\0';
19
20   return cadena;
21 }
```

1.13.7. Limpia los datos antes de pasarlos a otro programa

Es conveniente evitar el tener que pasar datos a otros programas. Si no hay más remedio porque el programa **CGI** simplemente actúa como pasarela y pasa los datos a otro programa, es conveniente que el **CGI** verifique que los datos no contienen ningún carácter especial que pueda producir un error: no se sabe como va a responder el programa externo ante datos inadecuados.

Cuando se empleen programas externos hay que indicar de forma explícita la ruta al programa, y no confiar en que se encuentran en el `PATH`. Si no se lleva cuidado, puede ser que se ejecute el programa que no es (incluso un programa que haya colocado una persona maliciosa en nuestro servidor).

1.13.8. Cuidado con HTML

Otra posible fuente de problemas es recibir código **HTML** cuando se espera texto plano. Suponed que tenemos un libro de firmas (*guestbook*); si el usuario introduce en alguno de los campos (por ejemplo, el nombre) códigos **HTML**, cuando se visualice su entrada en el libro de firmas, no se mostrará como el programador espera, si no que se aplicarán los formatos que el usuario haya introducido. De este modo tan sencillo se pueden insertar enlaces o imágenes de cualquier naturaleza en una web de otra persona.

Mucho peor es que inserte algún comando que permita realizar alguna operación. Por ejemplo, si el servidor web sabe procesar *Server Side Include* (**SSI**), un usuario puede incluir una instrucción como `<!-- #include file="/etc/passwd" -->` para visualizar el fichero de contraseñas o `<!-- #exec cmd="rm -rf /" -->` para borrar todo el sistema de archivos.

Existen dos soluciones para evitar este problema:

1. Impedir que el usuario pueda introducir los caracteres < y >. Si el usuario los introduce, se le muestra un mensaje de error para que vuelva a introducir su entrada o automáticamente se eliminan.

2. Traducir los dos caracteres a sus respectivos códigos de escape: `&lt;` para < y `&gt;` para >.

1.13.9. Nivel de privilegio

Si el sistema operativo lo permite, es recomendable ejecutar los programas **CGI** como un usuario no privilegiado, preferiblemente como un usuario espe-

cífico al que se le pueda asignar privilegios concretos (normalmente, para ello el servidor web se tiene que ejecutar con ese usuario).

1.13.10. Nivel de prioridad

Hay que evitar que un **CGI** nunca termine: varios programas **CGI** en un bucle infinito pueden colapsar y bloquear un servidor web. Para evitar estas situaciones, es conveniente asignar a los programas **CGI** una prioridad menor que el resto de procesos: de este modo, aunque nunca termine de ejecutarse un programa, no bloqueará el sistema.

1.13.11. Usa un ordenador para los CGIs

La mejor solución para evitar la mayoría de los problemas que se han comentado es elegir un ordenador para que sea servidor de **CGI**. Este ordenador no contendrá información importante y además no poseerá permisos de acceso a los otros ordenadores de la red. De este modo, aunque se produzca un agujero de seguridad, el atacante no podrá obtener mucha información. Además, es preferible que este ordenador se encuentre fuera del cortafuegos (*firewall*).

1.13.12. Consulta listas de correo y grupos de noticias

En los distintos medios de comunicación de Internet se puede encontrar información sobre agujeros de seguridad, nuevas versiones de software, etc. Toda esa información nos puede ayudar a tener un sitio web seguro.

1.13.13. Nunca olvides el código fuente

Nunca hay que dejar el código fuente de un programa **CGI** en el mismo directorio donde reside el ejecutable. Si se dispone del código fuente es más fácil localizar posibles agujeros de seguridad. Desgraciadamente, en aquellos casos en los que se emplea algún lenguaje interpretado (*Perl*, *shell* de Unix) no es posible evitar este problema.

Además, si sabes que tu programa **CGI** posee alguna vulnerabilidad, no lo indiques con un comentario en el código fuente. Si cae en manos ajenas, el atacante sólo necesita seguir las instrucciones que has dejado.

1.14. WinCGI

En 1994, Bob Denny creó el primer servidor web específico para Microsoft Windows 3.1. En aquella época, el lenguaje de programación más empleado en ese sistema operativo era Microsoft Visual Basic. Para facilitar la programación de **CGI** mediante Microsoft Visual Basic[27], Bob Denny creo un interfaz de programación similar a **CGI**, al que bautizó como WinCGI[28].

Poco después, el servidor de Bob Denny fue comprado por O'Reilly & Associates y se ha comercializado desde entonces con el nombre de O'Reilly WebSite Professional[29]. Este servidor web se encuentra disponible para todos los sistemas operativos de Microsoft.

Mientras que en el estándar **CGI** el servidor web pasa al programa **CGI** la información a través de variables de entorno, en WinCGI la información se pasa mediante los típicos ficheros `.ini` de Microsoft Windows. El intercambio de información en ambos sentidos (del servidor al programa **CGI** y viceversa) se realiza a través de ficheros (*file spooling*), lo que supone una merma en la velocidad de procesamiento frente a **CGI**.

Cuando el servidor web ejecuta un programa WinCGI, le pasa un único parámetro que indica la localización del fichero `.ini`. La especificación WinCGI define que en un fichero `.ini` existen ocho secciones y cada sección se compone de parejas `clave = valor`:

- `[CGI]`. Contiene las variables **CGI** usuales, como `Request Protocol`, `Request Method` o `Query String`.
- `[Accept]`. Indica los tipos **MIME** que el cliente comunica que acepta en el encabezado **HTTP**.
- `[System]`. Contiene variables que son específicas del estándar WinCGI. Las más importantes son `Content File` que indica el fichero que contiene los datos de la petición enviada por el cliente y `Output File` que indica el fichero en el que el programa tiene que almacenar su salida.

[27] Este lenguaje presentaba en sus primeras versiones varios inconvenientes que impedían su uso a la hora de programar **CGI**, como la dificultad que planteaba a la hora de acceder a las variables de entorno.

[28] WinCGI no se encuentra estandarizado como **CGI**, sólo existe una especificación informal.

[29] Desde el 20 de agosto de 2001, O'Reilly & Associates ha cedido los derechos de su servidor web a Deerfield.com.

- `[Extra Headers]`. Encabezados adicionales que se han encontrado en la petición del cliente.
- `[Form Literal]`. Si la petición se ha enviado mediante `POST`, el servidor descodifica los datos recibidos y los coloca en esta sección en forma de parejas `campo=valor`.
- `[Form External]`. Si alguno de los valores recibidos supera los 254 caracteres o contiene caracteres de control, se almacena en un fichero temporal y en esta sección se indica la localización del fichero.
- `[Form File]`. Si el formulario contiene controles para enviar ficheros[30], en esta sección se indica la localización de los ficheros recibidos.
- `[Form Huge]`. Si los datos recibidos superan los 65 535 bytes, el servidor web no los descodifica, pero en esta sección indica la localización de cada valor en el fichero indicado por `Content File`.

[30] `<INPUT TYPE="FILE">`.