Sistemas Distribuidos

Desde HTTP hasta CGI (y *Aplicaciones Web*)

Tabla de Contenidos

1. Introducción

Este apunte se enfocará, en realidad, en el funcionamiento de CGI (*Common Gateway Interface*), que (quizás de manera *rudimentaria*) fue el precursor de las aplicaciones web, específicamente en lo referente a ejecución de procesos del lado del servidor web y generación dinámica de páginas HTML (*HyperText Markup Language*). Dada la relación entre HTTP (*HyperText Transfer Protocol*), HTML y aplicaciones de CGI, se verán las características más importantes de HTTP y HTML relacionadas con CGI. La idea no es explicar en detalle el protocolo HTTP sino mostrar lo más importante de este protocolo en relación con las aplicaciones web y con CGI en particular. De la misma manera, no se pretende explicar completamente el lenguaje HTML sino lo específicamente relacionado con CGI, específicamente los FORMs de HTML, aunque hay otros aspectos de HTML que también están relacionados, pero se dejarán de lado en esta explicación (o a lo sumo se mencionarán sin dar muchos detalles). Dado que el objetivo más importante es llegar a una implementación completa (que ejecute efectivamente) también se darán algunos detalles de ejecución de programas vía CGI relacionados con un servidor web en particular, aunque de uso muy extendido: Apache.

Si bien CGI es un mecanismo (como se menciona antes un tanto *rudimentario*) para la creación de aplicaciones web, también es cierto que conceptualmente contiene/define todo necesario para la creación de una aplicación web. Posteriormente se han propuesto y se están utilizando otros mecanismos que son más elaborados y proveen mayor nivel de abstracción, más apropiado para estas aplicaciones, tal como los servlets en Java. La mayoría de estas abstracciones/paquetes/definiciones/etc. no son más que elaboraciones más complejas de todos los conceptos relacionados e implementados en CGI.

Resumiendo, para dar una explicación reducida aunque autocontenida de CGI, es necesario explicar detalles de HTML y HTTP. Para llegar a una aplicación que ejecute programas utilizando CGI, será necesario contar con un servidor web (como en cualquier aplicación web) en particular que sea capaz de ejecutar programas vía CGI y su correspondiente configuración. Es por esta razón que se verán los detalles relacionados en Apache, aunque efectivamente no se verán todos los detalles y, además, Apache no es el único servidor web capaz de resolver/implementar llamados a *programas CGI*.

A lo largo de todo este apunte se utilizarán servidor web y servidor HTTP como sinónimos, de la misma manera que Internet como sinónimo de World Wide Web o simplemente web, aunque es evidente que hay diferencias importantes. La idea es tomar el punto de vista de un usuario directamente a través de un navegador web, dado que es el punto de vista más *usual* en la actualidad para las aplicaciones web.

2. HTML y Navegación Web (Generación Estática y Dinámica de HTML)

El lenguaje HTML (HyperText Markup Language) es el lenguaje utilizado para la gran mayoría de los documentos que se transfieren en Internet. Básicamente es un lenguaje orientado a la *presentación* de documentos, es decir que permite la especificación de múltiples formas de mostrar información, y también incluye la posibilidad de mostrar varios “tipos” de información. Dada la utilización extensiva de Internet para acceder a los archivos que contienen HTML, normalmente se asume que un programa será el encargado de mostrar esta información: **Un navegador web**. Hay múltiples navegadores, pero evidentemente todos tienen acceso a la definición de HTML y de hecho, todos deberían interpretar y mostrar los archivos de la misma manera (aunque lamentablemente esto no siempre se cumple).

Evidentemente, no es necesario describir HTML en su totalidad para comentar CGI y de hecho la explicación de HTML sería demasiado extensa como para ser incluida en este apunte. Sí es necesario relacionar el lenguaje HTML con la transferencia desde y hacia un servidor de HTTP por parte de un cliente o navegador. Se podría dar en este punto una definición informal pero útil en cuanto a los *tipos* de documentos en HTML recuperables o accesibles en Internet: documentos **estáticos y dinámicos**. La navegación con documentos estáticos es justamente la recuperación de documentos HTML almacenados en algún servidor web. Podría afirmarse que el usuario no ingresa información más que la *propia* de elegir qué documento HTML requiere de qué servidor web. Se podría afirmar que el flujo de información se da *desde el servidor HTTP hacia el navegador/usuario*, a lo sumo el usuario no requerirá la información o no le será útil. En cierta forma, los documentos HTML

generados de manera dinámica intentan “adaptarse” al tipo o clase de información que requiere el usuario y/o el navegador web. En este caso, el usuario ingresa información más allá de un determinado documento almacenado en un servidor y de hecho, en los propios documentos HTML se determina cuánta información y de qué *tipo,* puede ingresar el usuario y esta información se transfiere *desde el navegador/usuario hacia el servidor HTTP*. Dado que CGI implica esta última forma de generación de páginas con HTML, se dejará de lado todo lo referente a generación estática de documentos HTML. Si bien se adopta esta idea respecto a CGI, es decir la de generación dinámica de documentos HTML a partir de la información ingresada por el usuario/cliente, se debe aclarar que CGI implica otras formas de generación de documentos HTML en donde no necesariamente el usuario ingresa información.

¿Cómo se representa o especifica en los documentos HTML que el usuario ingresa información para el (o que debe ser enviada al) servidor web? Una de las maneras más usuales es la inclusión en los documentos HTML de una o más secciones **FORM**. Los FORMs de HTML permiten la especificación para ingresar información del lado del cliente (que será luego enviada al servidor) así como también el *tipo* de requerimiento HTTP que se enviará al servidor. Al permitir la especificación del tipo de requerimiento, los FORMs en cierto modo no proveen *transparencia* respecto del protocolo de transporte, el propio HTTP (se está especificando en HTML el requerimiento HTTP) [11]. Desde el punto de vista sintáctico, un FORM en HTML no es nada más ni nada menos que una sección, como se aclaró previamente, *contenida* dentro de:

<FORM ...>
...
</FORM>

donde se especifica básicamente:

- METHOD = <“GET” / “POST”>. Indica qué requerimiento HTTP se enviará al servidor: GET o POST. Es un *atributo* del FORM. (Son 8. El resto son: PUT, DELETE, HEAD, TRACE, OPTIONS, CONNECT, raramente utilizados).
- ACTION = <URL>. Indica el URL (Uniform Resource Locator) del programa CGI. Es un *atributo* del FORM.
- INPUT [TYPE = <“RADIO” / “CHECKBOX” / “SUBMIT” / ...>] NAME = “<nombre>”. Indica diferentes formas/tipos de ingresar información del lado del cliente. Al menos hay un “SUBMIT” en un FORM, que es utilizado para dar por terminado el ingreso de información del lado del cliente y para iniciar el envío hacia el servidor.
- SELECT NAME = “<nombre>”... Indica la selección de una de varias alternativas (*pull down menu*).
- TEXTAREA NAME = “<nombre>”... Utilizado para ingresar texto en general.

Y se debe aclarar que existen otras especificaciones posibles además dentro de éstas enumeradas, varias de estas especificaciones tienen varias opciones y los elementos especificados dentro de un FORM se denominan controles. Estos controles son los que generalmente contienen la información que el usuario ingresa en el FORM y que luego será enviada al servidor web.

Como se ha señalado, no se incluirá una lista completa de detalles y opciones, sino lo más relevante como para mostrar/explicar las ideas y posibilidades generales. La Fig. 1 muestra un ejemplo sencillo de un FORM que solamente implica hacer un envío de un requerimiento “GET” al servidor indicando el programa CGI hola1 del directorio cgi-bin ubicado en el **directorio raíz** del sitio web.

```
<FORM METHOD="get" ACTION="/cgi-bin/hola1">
Click <input type ="submit" value="aquí"> para ver el
resultado del script/programa
</FORM>
```

Figura 1: Un FORM sin Entradas.

En la Fig. 2 se muestra la parte de la página web que corresponde a la visualización del FORM de la Fig. 1 tal como se muestra del lado del cliente. Es importante remarcar que éste es uno de los ejemplos donde el usuario de hecho no ingresa información, este FORM funciona como un ***hyperlink*** de HTML, con la diferencia fundamental de que se está haciendo referencia implícita (o explícita, de hecho en el documento HTML) a un programa a ejecutar en la computadora donde se ejecuta el

propio servidor de HTTP.

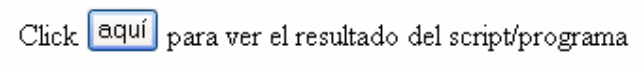

Figura 2: Visualización del FORM sin Entradas.

La Fig. 3 muestra un ejemplo sencillo de un FORM que generará un requerimiento POST al servidor indicando el programa CGI getopost del directorio cgi-bin ubicado en el directorio raíz del sitio web, donde además se le da al usuario la posibilidad de ingresar un valor.

```
<FORM METHOD="post" ACTION="/cgi-bin/getopost">
Ponga algo aquí: <INPUT NAME="algo">
Click <input type ="submit" value="aquí"> para ver el
resultado del script/programa
</FORM>
```

Figura 3: Un FORM con Una Entrada.

En la Fig. 4 se muestra la parte de la página web que corresponde a la visualización del FORM de la Fig. 3 tal como se muestra del lado del cliente.

Figura 4: Visualización del FORM con Una Entrada.

Desde la perspectiva de HTML, esto es todo lo específicamente relacionado con el ingreso y envío de información desde el cliente hacia el servidor. Desde la perspectiva del usuario/navegador, se tienen los FORMs (en realidad, su representación gráfica dentro de una página HTML, que generan una petición específica hacia el servidor con la posibilidad de incluir datos específicos del lado del cliente. Esta es una clara necesidad de toda aplicación web. Desde la perspectiva de la generación de HTML, evidentemente no necesariamente se puede esperar hasta aquí que todo HTML sea generado de manera dinámica (ni siquiera es estrictamente necesario, aunque actualmente es posible) y, de hecho, los ejemplos de FORMs que se han dado bien podrían ser partes de un documento HTML almacenado (estático) en un servidor web. Lo que se agrega hasta aquí en HTML está relacionado de manera directa con el propio protocolo HTTP, específicamente con la generación de un requerimiento GET o POST de HTTP que se incluye de manera explícita como *atributo* del FORM (valor del METHOD). También se agrega de manera explícita la indicación de un programa CGI que se ejecutará en el servidor (en realidad se requiere su ejecución desde el cliente y se *debería* llevar a cabo su ejecución en el servidor). Aunque se ha asumido hasta aquí, se debe notar que a partir de la utilización de FORMs en documentos HTML, el navegador web no solamente responde a requerimientos del usuario para recuperar información (vía *hyperlinks* con sus correspondientes URLs incluidos en los documentos HTML), sino que debe ser capaz de presentar estos formularios gráficamente, incorporar la información que el usuario ingresa y generar un requerimiento al servidor correspondiente.

La “fase final” de CGI desde la perspectiva del cliente es la visualización del documento HTML generado por el programa CGI que se requiere vía el FORM que se completó y del cual se hizo “submit”. De hecho, este resultado del CGI es el documento HTML generado dinámicamente. Se debe notar que no queda mucha alternativa más que generarlo de esa manera (dinámica) dado que debería responder (o adaptarse, como se explica antes) al ingreso de información específica del usuario, que es conocida en el servidor recién después de ser recibido el contenido del FORM. Entre el uso del INPUT “submit” por parte del cliente (posterior a la presentación de un FORM y su *correspondiente* ingreso de información) y la correspondiente visualización del resultado que llega como respuesta del servidor se tienen varios pasos que son transparentes al usuario y al propio lenguaje HTML, son manejados por el navegador y el servidor web en HTTP. Básicamente estas tareas tienen relación con la forma en que se envía el requerimiento con la información del FORM hacia el servidor web y las tareas de generación de un documento HTML en el servidor en respuesta a este requerimiento.

3. HTTP: *hacia* las Aplicaciones Web

En principio, HTTP (HyperText Transfer Protocol o Protocolo de Transferencia de HiperTexto) fue diseñado para la recuperación de documentos HTML (como lo indica el propio nombre del protocolo) almacenados en algún servidor web (HTML *estático*). A medida que avanza la utilización de Internet y las aplicaciones que se ejecutan sobre Internet, el protocolo se fue cambiando para *adaptarse* a estas aplicaciones *incorporando* características importantes/necesarias a nivel del protocolo. Como todos los protocolos relacionados con Internet su definición está hecha en al menos un RFC (Request for Comments). En el caso específico de HTTP, el RFC 2145 [6] describe la utilización de los diferentes números de versiones del protocolo: 0.9, 1.0, 1.1 y 1.2.

Todas las versiones de HTTP tienen un conjunto básico de conceptos comunes, como es de esperar, y estos conceptos son:

- Es un protocolo que sigue el modelo cliente/servidor. Más específicamente, es un protocolo de requerimiento/respuesta (request/reply), es decir que cada requerimiento del cliente hacia el servidor es respondido desde el servidor hacia el cliente.
- Es un protocolo orientado a conexión, es decir que todas las transferencias de datos se llevan a cabo en el contexto de una conexión (canal de comunicaciones bidireccional) entre el cliente y el servidor [2]. Uno de los agregados más importantes de la definición de HTTP/1.1 [2068] es la posibilidad de *persistencia* de una conexión. En la definición de HTTP/1.0, una conexión se utiliza para una sola petición/respuesta. La definición de HTTP/1.1 permite que varias peticiones/ respuestas puedan realizarse con una sola conexión, reduciendo así la sobrecarga que implica la generación de una conexión. De hecho, no se hace más que reconocer que un mismo cliente suele hacer varias peticiones a un mismo servidor en un tiempo relativamente corto de tiempo.
- Es un protocolo directamente orientado a la transferencia de la información contenida en documentos HTML, que están almacenados (o se generan) en el servidor y se presentan/utilizan en el cliente.
- Es un protocolo de nivel de aplicación [2]. Aunque en general se utilizan conexiones TCP/IP para el transporte de datos (puerto 80 TCP), HTTP solamente asume que existe un medio confiable para el transporte de los datos, no necesariamente TCP/IP.
- Es un protocolo sin estados, cada requerimiento/respuesta se lleva a cabo independientemente de los anteriores.
- Es un protocolo basado en texto [5]. Aunque se pueden transferir datos que no necesariamente son caracteres, todo lo definido por HTTP son cadenas de caracteres. Se explican los detalles a continuación.
- Toda comunicación entre el cliente y el servidor se lleva a cabo a través de *mensajes*, que son *requerimientos* (básicamente información de requerimientos) desde el cliente hacia el servidor y las *respuestas* (básicamente información de respuesta a un requerimiento previo) en sentido inverso, es decir desde el servidor hacia el cliente.

El formato de la petición y de la respuesta es común, y contiene, en general, hasta tres partes bien definidas:

1. **Una línea de petición/respuesta** (petición desde el cliente hacia el servidor y respuesta en sentido inverso). Esta línea es una cadena de caracteres con un formato determinado, dependiendo del tipo de mensaje (requerimiento o respuesta)
2. **Una sección de cabecera**, que no es más que una serie de líneas de texto indicando información extra sobre el proceso que envía la información (petición o respuesta). A la línea de cabecera debe seguirle una línea en blanco.
3. **El cuerpo**, que básicamente contiene datos importantes para ser usados en (o que han sido requeridos por) el receptor. El cuerpo de un mensaje es la única parte donde se puede incluir información que no es de texto, aunque la misma debe estar codificada como texto.

El contenido de cada una de estas tres partes depende de la propia petición/respuesta (del proceso que genera la petición/respuesta) y tanto la cabecera como el cuerpo dependen de la propia petición/ respuesta en cuanto a existencia como a contenido. La descripción de cada una de estas partes se hará en las secciones siguientes, desde la perspectiva del cliente (petición) o servidor (respuesta). Siempre debe recordarse que esta explicación será dada en el contexto de lo que es necesario para comprender CGI y, de hecho, HTTP es uno de los protocolos clásicos a usar y analizar en la bibliografía y las asignaturas de redes de computadoras [7] [3].

3.1 Peticiones HTTP

Las peticiones HTTP se llevan a cabo por parte de los clientes, que en el caso de las aplicaciones web son básicamente navegadores (*browsers*). En el contexto de la definición de HTTP, los clientes son denominados agentes de usuario (*user agents*) o directamente agentes. Desde la perspectiva del cliente, siempre las peticiones se llevan a cabo para obtener información (documentos HTML en la gran mayoría de los casos) del servidor al que le envían la petición. Se describe a continuación lo más importante del contenido de las tres partes mencionadas anteriormente: línea de petición, cabecera y cuerpo.

Línea de Petición

La línea de petición es una línea de texto con formato definido para realizar requerimiento de diferentes tipos de información, tal como lo muestra la Fig. 5. Los espacios en blanco son relevantes y se utilizan como separadores (recuérdese que HTTP es un protocolo basado en texto):

```
<GET/POST/HEAD/...>  <URI>  <HTTP/<1.0/1.1/...>>\r\n
```

Fig. 5: Formato de la Línea de Requerimiento HTTP.

donde las < / > indican opciones (y se debe especificar una de manera explícita) excepto en el caso de HTTP/ y los … indican que hay más posibilidades de las que se incluyeron de manera explícita en esta explicación.

Las primeras opciones de la línea de petición, es decir GET/PUT/HEAD/... son los literales que indican directamente el tipo de requerimiento. En este punto ya se puede completar la visión de la *falta de transparencia* que tienen los FORMs en HTML respecto del protocolo de transferencia de la información, que es HTTP. Un FORM en HTML genera *directamente* una petición GET o POST según el valor que se le da al atributo METHOD del FORM [11]. El URI (Uniform Resource Identifier) indica directamente qué se requiere del servidor siguiendo la nomenclatura de un URI. Siguiendo con la relación entre un requerimiento y lo que se tiene en un FORM, este URI está claramente relacionado con el atributo ACTION del FORM, indicando una vez más la falta de transparencia de los FORMs respecto de HTTP. Por último, se tiene la indicación de la versión del protocolo HTTP utilizado: 1.0 ó 1.1. etc.

Es interesante en este punto identificar con mayor nivel de detalle las características de los requerimientos GET y POST, dado que estos requerimientos son los que se referencian de manera directa desde un FORM de HTML [11]. Tanto el requerimiento GET como el requerimiento POST son relativamente intuitivos en cuanto a su definición, aunque **varía la sintaxis o la forma en que se incluyen los datos** dentro del requerimiento relacionados con un FORM de HTML:

- **Para un requerimiento GET**, todo lo necesario se especifica básicamente en la propia línea del requerimiento (que indica GET, por supuesto). En el URI se incluye la información referente al programa CGI especificado en el FORM, para el atributo ACTION y también toda la información que el usuario ingresa para el FORM (relacionados con los INPUTs). En general, desde la perspectiva de HTTP, el requerimiento GET está definido para la recuperación del contenido de un documento referenciado por el URI de la línea de petición. En el caso específico de utilizar programas CGI, el "documento referenciado por el URI" es el documento que se genera en la ejecución del programa dado explícitamente en el URI.
- **Para un requerimiento POST**, todo lo que se incluye es la referencia al programa CGI especificado en el atributo ACTION del FORM. A diferencia del requerimiento GET, no se incluye en esta línea la información introducida por el usuario en el FORM. En general, desde la perspectiva de HTTP, el requerimiento POST está definido para enviar información a un proceso corriendo en el servidor. En el caso específico de utilizar programas CGI, el "proceso corriendo en el servidor" es en realidad el proceso que se genera por la ejecución del programa dado explícitamente en el URI.

Es interesante notar que a pesar de las diferencias sintácticas, los requerimientos HTTP GET y HTTP POST generados a partir de un FORM de HTML no tienen diferencias significativas y, de hecho, comparten las características que son generales de CGI desde el cliente hacia el servidor web:

- Incluyen en el URI, la referencia explícita a un programa, que se ejecutará en el servidor, como parte de las tareas asociadas al procesamiento del requerimiento del cliente.
- Incluyen información relevante para la ejecución de un programa en el servidor (el que está referenciado por el URI).

De hecho, lo que es *esencialmente* diferente se da en el servidor, así que a priori no tiene mucha justificación la posibilidad de generar *lo mismo* de dos formas diferentes en un documento HTML, derivados de los posibles valores del atributo METHOD de un FORM.

Sección de Cabecera de la Petición

En general, la sección de cabecera de la petición es utilizada para dar información sobre el proceso origen del mensaje y sobre el mensaje en sí mismo. En el caso de la petición, la cabecera proporciona información tal como el tipo de datos que se pueden recibir como respuesta, si la conexión cliente/servidor debe ser mantenida después de la respuesta al requerimiento (solamente a partir de HTTP/1.1), etc. Esta última característica es muy importante para la navegación en general, pero podría afirmarse que es más importante a partir de las aplicaciones web (y aplicaciones basadas en CGI en particular), donde un mismo cliente efectúa varios requerimientos al mismo servidor en un período relativamente corto.

Aunque no está directamente relacionado con CGI sino con HTTP en general, es importante remarcar que una de estas informaciones del cliente, indica el tipo de datos que se pueden recibir como respuesta, es lo que determina que HTTP puede transportar datos binarios en general. En realidad, HTTP sigue la nomenclatura MIME (Multipurpose Internet Mail Exchange) para diferentes tipos de datos, con la clasificación MIME de tipo/subtipo [5] [8]. La Fig. 6 muestra un ejemplo de lo que se puede incluir en la cabecera al respecto, indicando en este caso específico que el cliente puede recibir cualquier tipo de información.

```
Accept: *.*
```

Fig. 6: Ejemplo de Información de la Cabecera de un Requerimiento HTTP.

Cuerpo de la Petición

Como se puede estimar a esta altura de la explicación, el cuerpo de la petición depende del tipo de petición y de los datos necesarios para procesar la petición por parte del servidor. En el caso específico de la utilización de CGI a través de FORMs en HTML, en el cuerpo de la petición se incluye toda la información que el usuario ingresa en el FORM.

3.2 Respuestas HTTP

Como se explica antes, las respuestas HTTP siguen el esquema general de los mensajes HTTP: línea de respuesta, sección de cabecera de la respuesta (terminando con una línea en blanco) y cuerpo de la respuesta. A continuación se detallan los aspectos más importantes de cada una de estas partes, relacionadas con la ejecución de programas CGI.

Línea de Respuesta

Normalmente la línea de respuesta (también denominada *línea de estado*) contiene la información que se muestra en la Fig. 7.

```
<http/<1.0/1.1/...>>  <Código>  <Descripción>\r\n
```

Fig. 7: Formato de la Línea de Respuesta HTTP.

donde cada una de las partes tiene el significado intuitivo dado por los nombres de la Fig. 7. Más específicamente, el código de respuesta/estado están bien definidos como para describir la gran mayoría de las situaciones que pueden presentarse para la respuesta de un servidor HTTP.

Sección de Cabecera de la Respuesta

La cabecera de la respuesta contiene información sobre la propia respuesta en general (como por ejemplo el tiempo estimado de generación/existencia de la respuesta [4]) y sobre el contenido o tipo de dato de la respuesta. Por ejemplo, la Fig. 8 muestra la información de la cabecera que indica que se estima que la respuesta se ha generado/existe desde hace 3 segundos y que el contenido es un documento Postscript.

```
Age: 3
Content-Type: application/postscript
```

Fig. 8: Ejemplo de Información de la Cabecera de una Respuesta HTTP.

A partir de este ejemplo se puede completar la idea de que aunque HTTP está basado en texto, no necesariamente siempre se transfiere información en modo texto. Desde la perspectiva de las aplicaciones web, tanto el requerimiento como la respuesta son expresados en texto, y la respuesta en particular suele ser una página web, es decir text/html en términos de Content-Type según la especificación dada por MIME.

Cuerpo de la Respuesta

Como en el caso de la petición HTTP, el cuerpo de la respuesta depende del tipo de respuesta y de los datos que se generan en el servidor como resultado de procesar la petición del cliente. En el caso específico de la utilización de programas CGI, el cuerpo de la respuesta es el documento web generado *dinámicamente* por el programa CGI que fue referenciado en el requerimiento.

4. Servidores Web, HTTP y CGI

A partir de lo explicado en las secciones anteriores respecto de la generación de HTML y del protocolo de transporte, en realidad no falta mucho para la explicación *completa* del procesamiento relacionado con CGI. Lo que aún no está definido se podría resumir en:

1. La idea general del procesamiento con CGI.
2. La forma en que los datos de un FORM se envían al servidor web y al programa CGI.
3. La forma en que el servidor web interactúa con el programa CGI y la relación de esta interacción con el requerimiento del cliente.

Si bien en la mayoría de los casos se presentan las aplicaciones web y de CGI en particular como para acceder a bases de datos en el servidor, conceptualmente es más general y, de hecho, un programa CGI podría llevar a cabo cualquier tarea (como cualquier programa) para la generación de un documento HTML. Esto incluye el acceso a una o más bases de datos y también el acceso a otra/s computadora/s diferentes de la que está ejecutando el servidor web.

4.1 Procesamiento de los Servidores Web con CGI

Los servidores web que *proveen acceso* a programas CGI o *simplemente respeta* o *cumple* la norma o el estándar CGI están preparados para el procesamiento estándar para la recuperación de documentos HTML estáticos y para identificar y procesar requerimientos a un programa externo al propio servidor web. En cierta forma, el servidor web puede seguir considerándose un proveedor de páginas web, sólo que a partir de CGI será capaz de identificar qué páginas web están directamente almacenadas en el sistema de archivos al que tiene acceso y qué requerimientos serán resueltos por un programa externo. Desde la perspectiva de estos requerimientos (los que hacen referencia a un programa externo) el servidor web no es nada más ni nada menos que un *gateway*, una *pasarela* o un *puente* hacia el programa que efectivamente resuelve el requerimiento. En términos de lo que ya se ha mostrado en este apunte, el programa externo al servidor web es el que se indica en el atributo ACTION de un FORM. El documento HTML generado de manera dinámica es lo que produce este programa externo al servidor web, cuya tarea (del servidor web), es hacerlo llegar al cliente que produjo el requerimiento.

La Fig. 9 muestra un servidor web que cumple la norma CGI, con las dos formas de procesar un

requerimiento. Nótese que en todos los casos la interacción de un proceso cliente (navegador web en la mayoría de los casos) siempre se da con el servidor web, el programa externo no se relaciona con el cliente ni para recibir el requerimiento ni para enviar la respuesta.

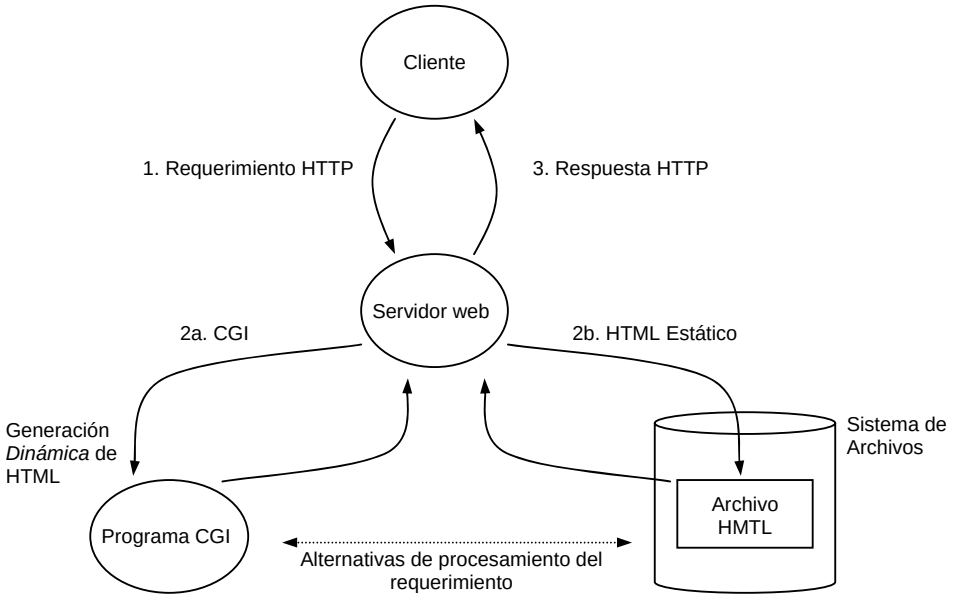

Figura 9: Procesamiento de un Requerimiento en un Servidor Web que *Implementa* CGI.

Es necesario aclarar que la Fig. 9 solamente incluye lo referido a CGI, no todas las alternativas actuales de procesamiento de un servidor web, que podría incluir otras características no contempladas en este apunte dedicado a CGI. Desde la perspectiva de un servidor web, algunas URL son provistas por un programa CGI. Siempre que un cliente hace referencia a una de esas URL se genera el programa CGI de forma tal que se le hace accesible la información que el cliente ha enviado, se recibe la salida del mismo programa y en general se le agrega la información necesaria (línea de respuesta HTTP y cabecera) y se envía al cliente como respuesta al requerimiento [12]. En algunos casos específicos, la salida del programa CGI *indica* al servidor que debe hacer algún tipo de *postprocesamiento* de la información que envía el propio programa CGI [1].

Servidor Web Apache

En este punto es apropiado explicar con un poco más de detalle qué es un servidor de HTTP, como para incluir los detalles de, por ejemplo, cómo se identifican las URL que serán provistas por programas CGI. Si bien es un servidor de HTTP en particular, Apache HTTP Server [9] [10] comparte con la mayoría de los servidores, el tipo de procesamiento que se requiere para un sitio web así como para los requerimientos incluyendo CGI. Por otro lado, la instalación y configuración de un sitio web con Apache en la mayoría de las distribuciones de Linux así como en otros sistemas operativos es muy sencilla. En esta explicación se dan los detalles de la distribución Fedora (Core 5 en particular), pero son similares o iguales al menos a las demás distribuciones de Linux.

Normalmente, en la instalación de Linux se pueden seleccionar diversos paquetes, bibliotecas o simplemente software de aplicación a utilizar en la computadora en la cual se está instalando el sistema operativo. Una de las alternativas es, justamente, el servidor web Apache. Si no se selecciona Apache para ser instalado junto con Linux se puede incluir luego en un “upgrade” (actualización) o, directamente instalando desde un RPM (**R**ed Hat **P**ackage **M**anager) binario en Fedora o, incluso desde los fuentes obtenidos directamente del sitio web de Apache.

Una vez instalado, el servidor web de Apache debe ser configurado, asumiendo una estructura de directorios que conformarán el sitio web que será manejado por este servidor. Normalmente, para la configuración se debe recurrir al archivo

```
/etc/httpd/conf/conf.d
```

Que contiene todo lo relacionado al funcionamiento del httpd, el proceso que en ejecución dará respuesta a los requerimientos HTTP. Este archivo de configuración contiene información de múltiples alternativas de funcionamiento/parámetros del servidor web Apache. En particular, interesan dos opciones/informaciones/directivas de configuración, relacionadas con el sitio web en general y con el funcionamiento de CGI en particular.

Quizás la directiva más importante de la configuración de Apache es la que se incluye en la sección

```
### Section 2: 'Main' server configuration
```

y que se denomina DocumentRoot. Esta directiva indica directamente a partir de dónde, en el sistema de archivos local al servidor de HTTP se tienen las páginas web (HTML estático, en principio) a proporcionar a los usuarios. La configuración que se tiene a partir de la instalación del servidor web de Apache es

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "/var/www/html"
```

Es decir que se espera que en el directorio /var/www/html haya un archivo index.html que es el *inicio* del sitio web. Si, por ejemplo, la computadora es conocida bajo el nombre DNS (Domain Name Service)

```
servidorweb.unp.edu.ar
```

al poner en la *Dirección* de un navegador web el URL (Uniform Resource Locator)

http://servidorweb.unp.edu.ar

se obtendrá en el navegador la visualización del archivo index.html del directorio /var/www/html, es decir que se visualizará

```
/var/www/html/index.html
```

en la computadora del navegador (donde se ingresó el URL *correspondiente*), si es que hay un archivo index.html en el directorio mencionado. A partir de este punto, todo lo relacionado con navegación estática del sitio dependerá, por supuesto del contenido de index.html y de lo que se incluya a partir de /var/www/html en el sistema de archivos.

La directiva que está directamente relacionada con CGI es ScriptAlias y también está incluida en la sección 2 y directamente indica el directorio donde están ubicados los programas CGI:

```
#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

La sintaxis de la directiva ScriptAlias es bastante intuitiva: todo lo que se referencie dentro de /cgi-bin/ del DocumentRoot será tratado como un programa CGI y el programa a ejecutar será buscado en el directorio /var/www/cgi-bin/ del sistema de archivos local. Es así que, si se ingresa en la *Dirección* de un navegador web el URL

```
http://servidorweb.unp.edu.ar/cgi-bin/uncgi
```

el servidor web buscará el archivo uncgi dentro del directorio del sistema de archivos local

/var/www/cgi-bin/ y seguirá los pasos que se muestran en la Fig. 9, en particular el 2a.

Respecto a los permisos de Archivo debe tenerse en cuenta que apache corre como el usuario **www-data**. Debido a ello los directorios donde el CGI lea y/o escriba debe tener permisos otorgados para ese usuario.

4.2 Datos del FORM: desde el Usuario hasta el Programa CGI

Como se explica antes, los datos de un FORM de HTML se presentan al usuario normalmente a través de un navegador web con un formato gráfico, como todo el documento HTML en el que está incluido el FORM. El usuario no hace más que ingresar la información que se requiere en el FORM, los controles INPUT, SELECT, TEXTAREA, etc. La explicación se podría separar en lo que corresponde al envío desde el navegador hasta el servidor HTTP y desde el servidor HTTP al programa CGI.

Desde el Navegador hasta el Servidor Web

Los valores ingresados en los controles por parte del usuario son "asociados" a cada uno de los NAME de esos controles y enviados al servidor como un conjunto de pares nombre=valor separados por &, denominado cadena de interrogación o *texto de consulta* (*query string*) dentro de un requerimiento HTTP GET o HTTP POST dependiendo del valor del atributo METHOD del FORM. La Fig. 10 muestra la cadena de interrogación que se genera a partir de un FORM con tres controles con nombres: catedra, universidad y sede respectivamente, donde el usuario ha ingresado para cada uno de ellos los valores Sistemas Distribuidos, UNP y Pto. Madryn-Trelew respectivamente.

```
catedra=Sistemas+Distribuidos&universidad=UNP&sede=Pto.+Madryn-Trelew
```

Figura 10: Cadena de Interrogación Generada a Partir de Tres Controles.

Tal como se puede identificar en la Fig. 10, algunos caracteres como el espacio se codifican de manera *especial* (con el signo + en el ejemplo). Claramente, otros caracteres como + y & deben ser codificados también de manera especial, y normalmente se recurre al hexadecimal que los representa, precediéndolo con %, lo cual hace que el propio % se codifique de manera especial con su código hexadecimal que lo representa.

Como se explica antes, la cadena de interrogación se incluye en el requerimiento GET o POST que genera el navegador a partir de que el usuario indica que ha ingresado todos los datos necesarios. La forma en que se incluye esta cadena de interrogación depende del requerimiento. En el caso de un requerimiento HTTP GET (que se genera a partir de un FORM con atributo METHOD="get"), la cadena de interrogación se incluye *como parte del* URI del requerimiento, de hecho se concatena al URL que se especifica en el atributo ACTION del FORM, separándolo del URL con el símbolo ?. Si se tiene el FORM de la Fig. 11

```
<FORM METHOD="get" ACTION="/cgi-bin/datos">
Ingrese Cátedra <INPUT NAME="catedra"> <br> <br>
Ingrese Universidad: <INPUT NAME="universidad"> <br> <br>
Ingrese Sede/s: <INPUT NAME="sede"> <br> <br>
Click <input type ="submit" value="aquí"> para enviar
los datos al servidor
</FORM>
```

Figura 11: Un FORM con Tres Controles.

se mostrará en un navegador de manera gráfica como en la Fig. 12, y si se ingresan los valores Sistemas Distribuidos, UNP y Pto. Madryn-Trelew en cada uno de los controles, se tendrá la línea de requerimiento HTTP de la Fig. 13 donde se asume que HTTP/1.0 es el protocolo que utiliza el cliente. En la Fig. 13 se puede completar la idea de por qué el espacio se codifica de manera especial en la cadena de interrogación: no debe generar errores al procesar HTTP. En espacio debe ser reemplazado por alguna codificación dado que en HTTP se utiliza el espacio para separar el requerimiento del URL y el URL del protocolo, tal como lo muestra la Fig. 5. Puesto de otra forma: el protocolo HTTP utiliza el espacio para *delimitar* cada una de las partes que compone la línea de texto

del requerimiento y, por lo tanto, no se pueden incluir en esta línea más espacios de los que establece el propio protocolo HTTP.

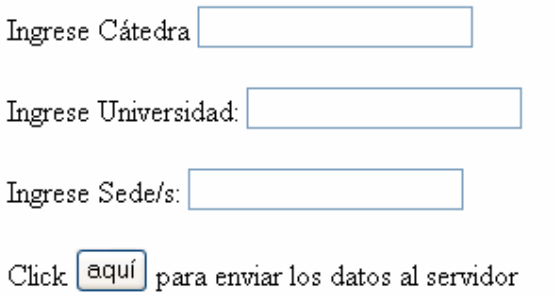

Figura 12: Visualización de un FORM con Tres Controles.

```
GET /cgi-bin/datos?catedra=Sistemas+Distribuidos&universidad=UNP&sede=Pto.+Madryn-Trelew HTTP/1.0\r\n
```

Figura 13: Línea de Requerimiento HTTP GET Generado a partir de Tres Controles.

En el caso de un requerimiento HTTP POST (que se genera a partir de un FORM con atributo METHOD="post"), la cadena de interrogación se incluye en el cuerpo del requerimiento, tal como se mencionó antes. La codificación de la cadena de interrogación no varía dependiendo del tipo de requerimiento, se codifica como se explicó para HTTP GET. Si se tiene el mismo FORM de la Fig. 11 pero con METHOD="post", la línea de requerimiento HTTP será

```
POST /cgi-bin/datos HTTP/1.0\r\n
```

y en el cuerpo del requerimiento se tendrá

```
catedra=Sistemas+Distribuidos&universidad=UNP&sede=Pto.+Madryn-Trelew
```

Nótese que ya no es necesario el ?, dado que no hay necesidad de *separar* el URL de la cadena de interrogación, están *naturalmente* separados.

Desde el Servidor Web hasta el Programa CGI

A partir de la Fig. 9 queda explícito que el servidor web debería **generar un proceso** con el programa CGI al que se hace referencia en el requerimiento HTTP. Lo que no se ha especificado aún es la forma en que el programa CGI recibe los datos que eventualmente incluye el FORM. Esto también depende del tipo de requerimiento: HTTP GET o HTTP POST. En el caso de un HTTP GET, toda la información necesaria para el programa CGI se tiene en *variables de entorno*. Más específicamente, el servidor web crea y asigna un conjunto de variables de entorno antes de generar el proceso con el programa CGI. Una de estas variables de entorno es

```
REQUEST_METHOD
```

y la asignación de esta variable en el caso de un HTTP GET es, justamente,

```
GET
```

Pero claramente el programa CGI debería recibir la cadena de interrogación de alguna manera y para esto el servidor define la variable de entorno

```
QUERY_STRING
```

y le asigna la cadena de interrogación que corresponde al HTTP GET, es decir la cadena entre el ? y el blanco de la línea de requerimiento de HTTP GET (de la Fig. 13, por ejemplo). A partir de este

punto, el programa CGI no necesita más que obtener los valores de esas variables de entorno y utilizar tales valores. En el lenguaje C, por ejemplo, no se necesitaría más que la llamada al sistema getenv(). Dado que el manejo/codificación de la cadena de interrogación es un poco complejo, usualmente se utilizan bibliotecas de manejo de estas cadenas de caracteres para obtener los pares (nombre, valor) del FORM. Estas bibliotecas incluyen funciones para, por ejemplo, reemplazar los + por los espacios que le corresponden.

En el caso de un requerimiento HTTP POST se sigue utilizando la variable de entorno `REQUEST_METHOD`, como era de esperar, pero la cadena de interrogación no se asigna a otra variable de entorno. El servidor web en este caso no solamente crea el proceso con el programa CGI sino que le proporciona la cadena de interrogación como la entrada estándar del proceso creado. Es decir que el proceso generado a partir del programa CGI no debe hacer nada más ni nada menos que procesar su entrada estándar. Nuevamente, la cadena de interrogación está codificada tal como llega del navegador del usuario y suelen utilizarse las bibliotecas mencionadas antes para manejarlas, para obtener los pares (nombre, valor) del FORM. Otra de las variables de entorno que el servidor web crea y asigna en el caso de un HTTP POST es

```
CONTENT_LENGTH
```

y el valor de esta variable de entorno es la longitud de la cadena de interrogación. A diferencia de otros procesos, los generados a partir de requerimientos HTTP POST conocen a priori la longitud de la entrada, dado que es la de la cadena de interrogación.

4.3 HTML desde el Programa CGI hasta el Usuario

Lo último que quedaría por explicar para completar el procesamiento de la Fig. 9 es bastante intuitivo: la forma en que la información generada por el programa CGI llega al servidor web y de allí al usuario/cliente. El programa CGI procesa y escribe en su salida estándar y esta información será la que el servidor web utilizará para responder al cliente/usuario.

Quizás lo más confuso de la salida de un programa CGI es que se debe producir información que correspondería a dos niveles de abstracción: HTTP y HTML o de aplicación. Dado que el servidor web no puede conocer a priori qué tipo de respuesta o que tipo de información se generará desde el programa CGI, es el propio programa CGI el que debe indicarlo, a nivel de HTTP. Tomando como referencia la generación de HTML, el programa CGI debe indicar con una línea que corresponde a la cabecera de la respuesta HTTP que generará texto HTML, produciendo una línea en su salida estándar con el contenido

```
Content-Type: text/html
```

y normalmente con una línea en blanco que le sigue, para indicar el fin de lo que sería la cabecera de HTTP. Dado que a partir de allí comenzaría el cuerpo de la respuesta, el programa CGI debería generar, ahora sí, el documento HTML que verá el usuario. Si, por el contrario, el programa CGI genera un documento postscript, lo primero que debería producir en su salida estándar es una línea con el contenido

```
Content-Type: application/postscript
```

seguida de una línea en blanco y a partir de allí comenzaría el contenido del documento postscript. Evidentemente, **en todos los casos el servidor web deberá anteponer a nivel de HTTP el resto de la información a lo que genera el programa CGI**, básicamente la línea de respuesta HTTP y la información de cabecera de HTTP que sea necesaria. Quizás esta *mezcla* de tipos de información o niveles de abstracción sean lo más representativo de una de las características por las cuales se menciona a CGI como un mecanismo un tanto rudimentario para la creación de aplicaciones web.

5. Un Ejemplo *Completo*

Como para mostrar de manera más completa la idea de CGI se presenta un sitio con algunas páginas web que hacen referencia a programas CGI vía FORMs. Se utilizará el servidor web de Apache y los programas CGI serán hechos en lenguaje C. La estructura de directorios del sitio web en el sistema de archivos local del servidor web será la estándar (la definida en/por la instalación en Fedora Core 5) del servidor de HTTP de Apache, que se muestra en la Tabla 1. En la raíz del sitio web, como el archivo index.html hace contiene una referencia directa a un programa CGI (la única sección FORM del documento) y referencias a otros documentos estáticos HMTL. El contenido de index.html se muestra en la Fig. 14 y también se incluye da en el Anexo 1.

Directorio/archivo	Contenido
/etc/http/conf/httpd.conf	Configuración de httpd
/var/www/html/	"Raíz" del sitio web del servidor httpd
/var/www/cgi-bin/	Directorio para contener los programas CGI

Tabla 1: Archivos/Directorios/Configuración del Servidor Web de Apache.

```
<HTML>
<BODY>

<h1 style="TEXT-ALIGN: center"> Ejemplos de Uso de CGI </h1>
<p>
Este es el CGI original, solamente para ver si funciona: Buéh, ahora veremos
si se ve algo o un error
<FORM METHOD="post" ACTION="/cgi-bin/hola1">
Click <input type ="submit" value="aquí"> para ver el resultado del
script/programa
</FORM>
</p>

<p>
<a href="getopost/getopost.html"><font color= blue>Esta es otra página web
</font></a>, que hace referencia al CGI getopost, que muestra las variables
de entorno "que corresponden".
</p>

<p>
<a href="formdata/formdata.html"><font color= blue>Esta es otra página web
con un formulario</font></a> (ahora con entradas), que hace referencia al
CGI getopost, que es igual al anterior: muestra las variables de entorno
"que corresponden".
</p>

<p>
<a href="rightread/rightread.html"><font color= blue>Esta es otra página
web con un formulario</font></a> (ahora con entradas), que hace referencia
al CGI rightread, que es similar al anterior: muestra las variables
de entorno "que corresponden" y además muestra la entrada correcta
(vía QUERY_STRING o stdin) del CGI.
</p>

</BODY>
</HTML>
```

Figura 14: Contenido del Archivo index.html.

La Fig. 15 muestra la forma en que un navegador visualiza en pantalla el documento estático HTML de la Fig. 14.

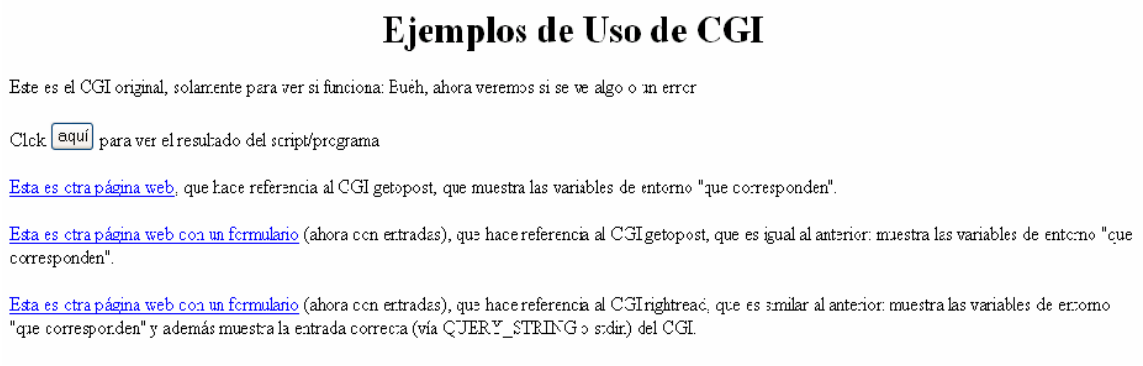

Figura 15: Visualización de index.html en un Navegador.

El primer programa CGI, al que se hace referencia en el FORM del documento index.html de la Fig. 14, se muestra en la Fig. 16. Se debe notar lo comentado antes, la primera salida del programa no corresponde al contenido de un documento HTML sino a una parte de la cabecera HTTP de la respuesta para el cliente (navegador web). Claramente, el resto es de la salida del programa son las líneas de texto correspondiente a un documento HTML que será visualizado en el cliente como se muestra en la Fig. 17.

```
main(int argc, char *argv[])
{
// Si esta primera linea se comenta se producira un error...
printf("Content-type: text/html%c%c", 10, 10);

printf("<HTML><BODY>\n");
printf("<font color = blue>\n");
printf("<h1 style=\"TEXT-ALIGN: center\">Hey, ahora si</h1>\n");
printf("</font>\n");
printf("Esto no es un error, es la salida de un programa sin entradas (siempre
lo mismo...)\n");
printf("</HTML></BODY>\n");
}
```

Figura 16: Programa CGI que no Procesa Entradas.

Se deben recordar o hacer notar dos aspectos relacionados con el programa de la Fig. 14: generación y ubicación del ejecutable y generación de la respuesta para el cliente desde el servidor web. A partir del código fuente dado en la Fig. 16 necesariamente debe generarse el binario/ejecutable, y debe existir una copia de este ejecutable dentro del directorio `/var/www/cgi-bin` como lo indica la Tabla 1. Es más, el nombre de este ejecutable debe ser **hola1**, que es el nombre con el que se referencia en el FORM del documento index.html dado en la Fig. 14. Además, está claro que el programa de la Fig. 14 no tiene todo lo necesario para la respuesta HTTP que debe llegar al cliente, es decir que al menos el servidor web debería generar la línea de respuesta y el mínimo contenido necesario de la cabecera de la respuesta HTTP. Es interesante que este contenido mínimo puede ser generado de manera independiente de lo que genere el programa CGI, dado que éste genera el final de la cabecera donde indica el tipo de información que contiene el cuerpo y el cuerpo mismo.

Figura 17: Visualización de la Salida HTML del Programa CGI.

Tal como de alguna manera lo indica el documento index.html de la Fig. 14, el *sitio web* de este ejemplo debería tener los archivos/directorios que se muestran en la Tabla 2 (se incluyen los de la Tabla 1 para que esté el ejemplo completo).

Directorio/archivo	Contenido
`/etc/http/conf/httpd.conf`	Configuración de httpd
`/var/www/html/`	"Raíz" del sitio web del servidor httpd
`/var/www/cgi-bin/`	Directorio para contener los programas CGI
`/var/www/html/index.html`	Documento inicial del sitio web, Anexo 1
`/var/www/html/getopost/getopost.html`	Página web, Anexo 2
`/var/www/html/formdata/formdata.html`	Página web, Anexo 3
`/var/www/html/rightread/rightread.html`	Página web, Anexo 4
`/var/www/cgi-bin/hola1`	Programa CGI referenciado en index.html, Anexo 5
`/var/www/cgi-bin/getopost`	Programa CGI referenciado en getopost.html, Anexo 6
`/var/www/cgi-bin/rightread`	Programa CGI referenciado en rightread.html, Anexo 7

Tabla 2: Archivos/Directorios/Configuración del Servidor Web de Apache.

En todos los casos, debe recordarse que se dan los fuentes de los programas CGI, pero se deben generar los ejecutables y poner copias de éstos en el directorio `/var/www/cgi-bin` con los nombres que corresponden, es decir con los nombres con los que se hace referencia a ellos en los FORMs de los documentos HTML dados.

6. Bibliografía

[1] CGI - Common Gateway Interface, CGI Script Output, http://hoohoo.ncsa.uiuc.edu/cgi/out.html

[2] T. Berners-Lee, R. Fielding, and H. F. Nielsen, Hypertext Transfer Protocol, HTTP/1.0., RFC 1945, HTTP Working Group, May 1996.

[3] D. E. Comer, The Internet Book: Everything You Need to Know About Computer Networking and How the Internet Works, 4th Edition, Prentice Hall, ISBN: 0132335530, 2006.

[4] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol, HTTP/1.1, RFC 2616, June 1999.

[5] M. L. Liu, Distributed Computing: Principles and Applications, Addison-Wesley, 2004, ISBN: 0-201-79644-9.

[6] J. C. Mogul, R. Fielding, J. Gettys, H. F. Nielsen, Use and Interpretation of HTTP Version Numbers, RFC 2145, May 1997.

[7] A. S. Tanenbaum, Computer Networks, 4th Edition, Prentice Hall, ISBN: 0-13-066102-3, 2003.

[8] A. S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, 2nd Ed., Prentice Hall, ISBN 0132392275, 2006.

[9] The Apache Software Foundation, http://www.apache.org/

[10] The Apache Software Foundation, Apache HTTP Server Project, http://httpd.apache.org/

[11] W3C Recommendation, Forms in HTML Documents, http://www.w3.org/TR/html4/interact/forms.html

[12] Wikipedia, the free encyclopedia, Common Gateway Interface, http://en.wikipedia.org/wiki/Common_Gateway_Interface

Anexo 1: index.html

```
<HTML>
<BODY>

<h1 style="TEXT-ALIGN: center"> Ejemplos de Uso de CGI </h1>
<p>
Este es el CGI original, solamente para ver si funciona: Buéh, ahora veremos
si se ve algo o un error
<FORM METHOD="post" ACTION="/cgi-bin/hola1">
Click <input type ="submit" value="aquí"> para ver el resultado del
script/programa
</FORM>
</p>

<p>
<a href="getopost/getopost.html"><font color= blue>Esta es otra página web
</font></a>, que hace referencia al CGI getopost, que muestra las variables
de entorno "que corresponden".
</p>

<p>
<a href="formdata/formdata.html"><font color= blue>Esta es otra página web
con un formulario</font></a> (ahora con entradas), que hace referencia al
CGI getopost, que es igual al anterior: muestra las variables de entorno
"que corresponden".
</p>

<p>
<a href="rightread/rightread.html"><font color= blue>Esta es otra página
web con un formulario</font></a> (ahora con entradas), que hace referencia
al CGI rightread, que es similar al anterior: muestra las variables
de entorno "que corresponden" y además muestra la entrada correcta
(vía QUERY_STRING o stdin) del CGI.
</p>

</BODY>
</HTML>
```

Anexo 2: getopost.html

```
<HTML>
<BODY>

<h1 style="TEXT-ALIGN: center">El Segundo CGI</h1>

<p>
Ejemplo que muestra la entrada de un programa "referenciado/invocado"
con GET. Dado que el formulario no tiene más que el propio "get"
(con el correspondiente "submit"), el programa no debería recibir
nada como entrada, es decir que la entrada debería ser nula.
</p>

<FORM METHOD="get" ACTION="/cgi-bin/getopost">
Click <input type ="submit" value="aquí"> para ver el resultado
del script/programa
</FORM>

</BODY>
</HTML>
```

Anexo 3: formdata.html

```
<HTML>
<BODY>

<h1 style="TEXT-ALIGN: center">El "Tercer" CGI (el segundo Form, en
realidad)</h1>

<p>
Ejemplo que muestra la entrada de un programa "referenciado/invocado"
con GET. Ahora el formulario tiene más que el propio "get" (con el
correspondiente "submit") y el programa debería recibir algo como
entrada, es decir que la entrada no debería ser nula.
</p>

<!
if METHOD="post" then the cgi will not receive any parameters via
the QUERY_STRING environment variable
--
>

<FORM METHOD="get" ACTION="/cgi-bin/getopost">
Ponga algo aquí: <INPUT NAME="algo">
Click <input type ="submit" value="aquí"> para ver el resultado
del script/programa
</FORM>

</BODY>
</HTML>
```

Anexo 4: rightread.html

```
<HTML>
<BODY>

<h1 style="TEXT-ALIGN: center">El Cuarto CGI: referencia a rightread</h1>

<p>
Ejemplo que muestra la entrada de un programa "referenciado/invocado"
con GET o POST. Los dos primeros formularios de esta página web se
utilizan para identificar diferencias entre diferentes valores de
METHOD, dado que uno es con METHOD="get" y el otro con METHOD="post".
El programa rightread es capaz de mostrar el contenido de la cadena
de interrogación  correctamente, es decir mostrando la variable de
entorno o la entrada (stdin).
</p>

<FORM METHOD="get" ACTION="/cgi-bin/rightread">
Ponga algo aquí: <INPUT NAME="algo">
Click <input type ="submit" value="aquí"> para ver el resultado
del script/programa con METHOD="get"
</FORM>

<FORM METHOD="post" ACTION="/cgi-bin/rightread">
Ponga algo aquí: <INPUT NAME="algo">
Click <input type ="submit" value="aquí"> para ver el resultado
del script/programa con METHOD="post"
</FORM>

<FORM METHOD="post" ACTION="/cgi-bin/rightread">
Este es solamente otro FORM con método POST <br> <br>
Ingrese Cátedra <INPUT NAME="catedra"> <br> <br>
Ingrese Universidad: <INPUT NAME="universidad"> <br> <br>
Ingrese Sede/s: <INPUT NAME="sede"> <br> <br>
Click <input type ="submit" value="aquí"> para enviar
los datos al servidor
</FORM>

</BODY>
</HTML>
```

Anexo 5: hola1.c

```
main(int argc, char *argv[])
{
// Si esta primera linea se comenta se producira un error...
printf("Content-type: text/html%c%c", 10, 10);

printf("<HTML><BODY>\n");
printf("<font color = blue>\n");
printf("<h1 style=\"TEXT-ALIGN: center\">Hey, ahora si</h1>\n");
printf("</font>\n");
printf("Esto no es un error, es la salida de un programa sin entradas (siempre
lo mismo...)\n");
printf("</HTML></BODY>\n");
}
```

Anexo 6: getopost.c

```
#include <stdio.h>     /* printf() */
#include <stdlib.h>    /* getenv() */

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

  // Si esta primera linea se comenta se producira un error...
  printf("Content-type: text/html%c%c", 10, 10);

  /* Variables de ambiente relacionadas con un Form-get o Form-post + Query_String */
  reqmet   = getenv("REQUEST_METHOD");
  querystr = getenv("QUERY_STRING");

  printf("<HTML><BODY>\n");
  printf("<font color = blue>\n");
  printf("<h1 style=\"TEXT-ALIGN: center\">¿Get o Post? - ¿Parámetros por Variable de
Entorno?</h1>\n");
  printf("</font>\n");
  printf("REQUEST_METHOD = %s<br>", reqmet, 10);
  printf("QUERY_STRING = %s\n", querystr);
  printf("</HTML></BODY>\n");
}
```

Anexo 7: rightread.c

```
#include <stdio.h>     /* printf() */
#include <stdlib.h>    /* getenv(), malloc() */
#include <string.h>    /* strcmp() */

main(int argc, char *argv[])
{
  char *reqmet, *querystr;
  int numofchars = -1;
  int retfromfread = -1;

  // Si esta primera linea se comenta se producira un error...
  printf("Content-type: text/html%c%c", 10, 10);

  /* Variables de ambiente relacionadas con un Form-get o Form-post + Query_String */
  reqmet   = getenv("REQUEST_METHOD");

  if (strcmp(reqmet, "GET") == 0)
    querystr = getenv("QUERY_STRING");
  else
  {
    numofchars = atoi(getenv("CONTENT_LENGTH"));
    querystr = (char *) malloc(numofchars);
    retfromfread = fread(querystr, 1, numofchars, stdin);
    querystr[numofchars] = '\0';
  }


  printf("<HTML><BODY>\n");
  printf("<font color = blue>\n");
  printf("<h1 style=\"TEXT-ALIGN: center\">Get o Post: Mostrar el QUERY_STRING</h1>\n");
  printf("</font>\n");
  printf("REQUEST_METHOD = %s<br>", reqmet);
  printf("CONTENT_LENGTH = %d<br>", numofchars);
  printf("retfromfread = %d<br>", retfromfread);
  printf("QUERY_STRING (variable de ambiente o stdin) = %s\n", querystr);
  printf("</HTML></BODY>\n");
}
```