SERVIDOR WEB MULTIPLATAFORMA CON IMPLEMENTACIÓN CGI

C.U. Loraine E. Gimson Saravia[a], C.U. Julián J. Fernández[b]

L.I.D.T.I. Universidad Nacional de Salta. Facultad de Ciencias Exactas

[a] E-Mail: saraviag@unsa.edu.ar

[b] E-Mail: julianjf@unsa.edu.ar

Asesores:
Daniel Arias. Universidad Nacional de Salta. Facultad de Ciencias Exactas
Jorge Ramírez. Universidad Nacional de Salta. Facultad de Ciencias Exactas

Julio 2001

1. RESUMEN

El objetivo que perseguimos en este trabajo fue el desarrollo de un servidor Web multiplataforma basado en el protocolo HTTP 1.0 con implementación de la interfaz CGI, utilizando como lenguaje de programación JAVA. Los métodos que implementa son GET, HEAD, POST, PUT, DELETE .También dotamos al servidor con la capacidad de entender y responder a peticiones de clientes que utilicen protocolo HTTP 0.9 (GET) y HTTP 1.1 (GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE).

Desarrollamos también un administrador del servidor que permitiera, a la persona que lo administrara, decidir dónde se encontrarían los recursos que podrían ser accedidos a través del servidor y si éstos serían de libre acceso o de acceso restringido; todo esto a través de una interfaz gráfica.

Y como complemento, decidimos construir un cliente web que permitiera subir (upload) y bajar (download) archivos en un servidor y otros métodos que navegadores comerciales no implementaban.

Palabras Claves: Servidor Web Multiplataforma, Cliente Web, HTTP, CGI

2. INTRODUCCION

Con este trabajo pretendemos explicar el desarrollo de un servidor Web multiplataforma con implementación CGI, realizado en JAVA, que proporciona el servicio de distribución de páginas www y permite el diálogo con los clientes web.

Para que el servidor fuera flexible, desarrollamos un administrador del servidor con características particulares que incluye la publicación de Webs, control de acceso y gestión de enlaces.

Finalmente para poner en práctica algunas de las funciones del servidor, decidimos construir una aplicación cliente que permitiera principalmente almacenar y eliminar archivos.

3. DISEÑO DE LA APLICACIÓN

3.1. Diseño del servidor

El diseño del servidor está basado en las RFC Requests For Comment (Solicitudes Para Comentar) y no en los servidores comerciales. Debido a ello se podría decir que implementa características que no se encuentran en muchos servidores comerciales, como ser los métodos Delete y Put.

3.1.1. Métodos que soporta el servidor

El servidor desarrollado soporta distintas versiones del protocolo HTTP, "Hyper Text Transfert Protocol, éstas son HTTP/0.9, HTTP/1.0 y parte del HTTP/1.1. De la versión 1.0 tiene implementados los métodos GET, HEAD y POST que son estándares, además implementa PUT y DELETE que no son estándares de la 1.0 según RFC 1945; y además entiende HTTP/0.9. El servidor también soporta los métodos GET, HEAD, POST, PUT, DELETE, OPTIONS y TRACE de la versión 1.1 del HTTP.

3.1.2. Cómo trabaja el servidor

- *Manejo de conexiones*

Al desarrollar el servidor pensamos en la utilización de un hilo por cada conexión. Así construimos el servidor como un proceso (padre) que se encuentra a la espera de pedidos en determinado puerto (por defecto es el puerto 80) de tal manera que una vez que recibe la petición crea un proceso hijo que se encarga de analizarla, procesarla y devolver una respuesta acorde por medio del socket correspondiente. Mientras tanto el proceso padre vuelve a escuchar en el puerto 80, dando la posibilidad de atender a varios clientes simultáneamente.

Con esto, en caso de ocurrir un problema en alguno de ellos solamente lo padecería esa conexión y en ningún momento el resto de los clientes quedarían afectados por el mismo. De esta manera tenemos varios procesos que hacen una tarea específica, esperando así lograr un funcionamiento más eficiente.

- *Procesamiento de la solicitud del Cliente*

El servidor comienza analizando la primera línea de la cabecera del pedido, donde se encuentran, el método solicitado, el recurso al que se quiere aplicar el método y el protocolo que usa el cliente[1]. Si el método solicitado o el protocolo usado no están implementados en el servidor se genera un mensaje con código de estado 501 o 505 respectivamente que indica que lo que se pide al servidor no se encuentra implementado, se envía la respuesta (compuesta de una línea de estado y varias líneas de cabecera) y se cierra la conexión; caso contrario se verifica si el usuario tiene acceso al recurso y se analiza el resto de la cabecera.

El procedimiento de verificación de acceso al recurso consta de las siguientes tareas:

- A través del alias dado, el servidor obtiene el path real y el tipo de recurso que contiene y con él arma el nombre físico del recurso. Si solo se envió un alias, sin el nombre del archivo y si el alias es del tipo "HTML", el servidor intenta buscar dentro del path real la página home.html o index.html.
- Comprueba que realmente el recurso existe, de lo contrario envía al cliente un mensaje con código de estado 404 indicando que no se pudo hallar el recurso[2] y se cierra la conexión.
- Determina si el recurso se encuentra dentro de una región y en caso de estarlo busca si en la cabecera se mandó el campo "Authorization". Si no se envió este campo genera un mensaje al cliente con código de estado 401 indicando que pertenece a un área restringida y que si se desea acceder al recurso deberá autenticarse y se cierra la conexión.
- En el caso de haber recibido "Authorization", este campo contiene el esquema de autenticación seguido de la identificación del usuario y su clave. El servidor desarrollado maneja el esquema "BASIC" de autenticación pues es el esquema estándar que todo servidor debe manejar[3]. En este

[1] Ver RFC 1945 [8].

[2] En caso del método PUT si no existe el recurso no se envía un mensaje de error sino que se crea ese recurso y se le coloca el contenido del cuerpo del pedido.

[3] Ver las RFC 1945 [8] y 2068 [9].

esquema básico la identificación y la clave del usuario se encuentran codificados en base 64, por lo tanto el servidor debe decodificarlos. Una vez decodificados debe comprobar que sea un usuario válido y que pertenezca a la región del recurso, caso contrario envía un mensaje con código de estado 403 indicando que tiene prohibido el acceso a ese recurso.

Con esta información el servidor puede entonces llevar a cabo la acción solicitada por el cliente, que dependerá del método pedido. Cumplida la misma, el servidor arma la cabecera y el cuerpo de respuesta y se los envía al cliente.

3.1.3. Comportamiento del Servidor frente a los distintos métodos

El servidor se comporta de manera diferente para los distintos métodos:

- *Método GET*

Una vez que el servidor conoce el recurso solicitado por la petición y el tipo de alias que lo referencia, si el tipo de alias:

- No es "CGI", analiza si se envió un campo de cabecera "If-Modified-Since"[4] que contiene una fecha en formato GMT, que indica que el cliente desea el recurso si su fecha de modificación es posterior a la que contiene ese campo. Si no se produjo una modificación posterior a esa fecha, se envía un mensaje con código de estado 205 indicando que no hay una versión más actualizada y no se envía el recurso, de lo contrario, se arma la cabecera apropiada y se envía el mismo.

  Sólo en este método y en el caso de que el tipo de alias no sea ejecutable, el servidor puede dejar abierta la conexión con el cliente para que este pueda realizar sus pedidos en forma mas rápida, si es que el cliente le envió en la cabecera el campo "Connection" con el valor "Keep-Alive" (manténte vivo).
- Si es "CGI", el servidor antes de realizar la verificación mencionada anteriormente debe separar del recurso el path adicional y los parámetros, si los tuviera, para obtener el nombre del CGI que el cliente quiere ejecutar.

---

[4] Ver RFC 1945 [8] y 2068 [9].

El servidor carga las variables de entorno que necesita el CGI, ejecuta el recurso CGI que fue solicitado y le envía las variables de entorno y los parámetros, si los hubiera. Podemos observar este procedimiento en la Figura 1.

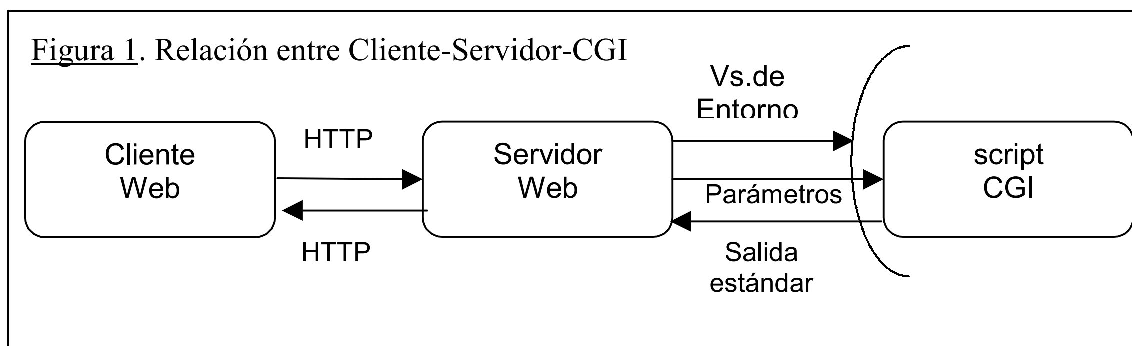


Figura 1. Relación entre Cliente-Servidor-CGI

Luego, si el nombre del CGI comienza con "nph-" el servidor no analiza los datos que le devuelve el CGI y simplemente transmite todo al cliente que lanzó la petición. Caso contrario arma la cabecera correspondiente al resultado de la ejecución del CGI (el servidor verifica que el CGI haya enviado un valor para "Content-Type", si esto no es así el servidor considera que es del tipo "text/plain").

- *Método HEAD*

Ante este método, el servidor funciona en forma análoga que con GET pero no manda el recurso sino *sólo envía su información de cabecera.* Tampoco brinda la posibilidad de dejar la conexión abierta ni analiza el campo "If-Modified-Since" de la cabecera de la petición.

- *Método POST*

Con el método POST el comportamiento del servidor es similar a cuando recibe un pedido GET para un recurso que hace referencia a un script CGI. El servidor, una vez que verificó el acceso al recurso manda a ejecutarlo y trata de obtener el cuerpo del pedido enviado por el cliente para pasárselo al CGI. La diferencia con GET es que *permite pasarle más información a la aplicación* en ejecución.

Como la única forma que tiene el servidor para saber la cantidad de información que va a recibir a través del cuerpo de la petición es que el cliente le indique ese valor, el servidor analiza en la cabecera del pedido si el campo "Content-Length" no es nulo para poder ejecutar el script CGI en

forma correcta. Si este campo es nulo envía un mensaje de error con código de estado 400 indicando que la petición estuvo mal formulada[5] y no manda a ejecutar el CGI.

- *Método PUT*

Al igual que con POST, el servidor recibe información en el cuerpo de la petición del cliente. Aquí, el servidor verifica el acceso al recurso, si éste existe el cuerpo del pedido del cliente se considera como el contenido actualizado del mismo, caso contrario se crea el recurso que va a tener como contenido el cuerpo del pedido[6].

El servidor desarrollado no permite que mediante este método se coloquen recursos en el directorio por defecto o en los destinados a los archivos ejecutables, ya que la única manera para subir al servidor recursos que puedan ejecutarse, es a través del administrador de archivos que se encuentra en el servidor y al que en un principio solo el administrador del servidor tiene acceso.

Si el servidor realizó con éxito la incorporación del nuevo recurso o bien la actualización, envía un mensaje con código de estado 201 (Creado), sino devuelve un mensaje de error con el código de estado correspondiente.

- *Método DELETE*

Ante una petición de Delete, el servidor, después de verificar acceso al recurso, analiza si el alias está asociado al directorio por defecto o a los destinados a contener ejecutables. En ambos casos no permite eliminarlo y devuelve un mensaje con código de estado 403 (“Prohibido”). Si el servidor realizó con éxito la eliminación del recurso envía un mensaje con código de estado 200.

- *Método OPTIONS*

Ante la presencia del comando Options, el servidor devuelve todos los métodos que puede realizar el cliente, no necesitando realizar la verificación de acceso anteriormente mencionada.

El servidor actúa de manera diferente según sea:

a) que el nombre del recurso sea asterisco (“*”) o bien
b) que realmente sea el nombre de un recurso.

---

[5] Ver RFC 1945 [8] y 2068 [9].

[6] Ver RFC 2068 [9].

Si la URI (Universal Resource Identifier) solicitada es el asterisco, el servidor devuelve todos los métodos que él soporta[7], por lo que colocará en la etiqueta de cabecera de respuesta "Allow" los siguientes métodos: TRACE, OPTIONS, GET, HEAD, POST, PUT y DELETE.

En cambio, si la URI pedida es el nombre de un recurso existente, el servidor colocará junto al la etiqueta "Allaw" los métodos TRACE y OPTIONS y adicionará los métodos:

a) GET y HEAD; si el recurso:

- no pertenece a una región;
- pertenece a una región y además el cliente envió su clave e identificación correspondientes

b) PUT y DELETE; si el tipo de alias que lo referencia no es "Default" ni "CGI" y el cliente envió la clave e identificación de acceso a la región del recurso.

- *Método TRACE*

Cuando el servidor recibe una solicitud con Trace, no hace verificación sobre el recurso, simplemente arma la cabecera y envía como cuerpo el pedido tal cual lo recibió[8].

Podemos ver en la Tabla 1 algunas de las diferencias existentes entre los distintos métodos contemplados por el servidor.

Tabla 1 Diferencias entre los métodos

Métodos	Verificación de acceso	Petición con cuerpo	Respuesta con cuerpo	Respuesta solo cabecera	Versión del Protocolo
GET	Si	No	Si	Si	0.9; 1.0;1.1
HEAD	Si	No	No	Si	1.0
POST	Si	Si	Si	Si	1.0;1.1
PUT	Si	Si	No	Si	1.0;1.1
DELETE	Si	No	No	Si	1.0;1.1
OPTIONS	No	No	No	Si	1.1
TRACE	No	No	Si	No	1.1

[7] Ver RFC 2068[9].

[8] Ver RFC 2068[9].

3.2. Diseño del cliente

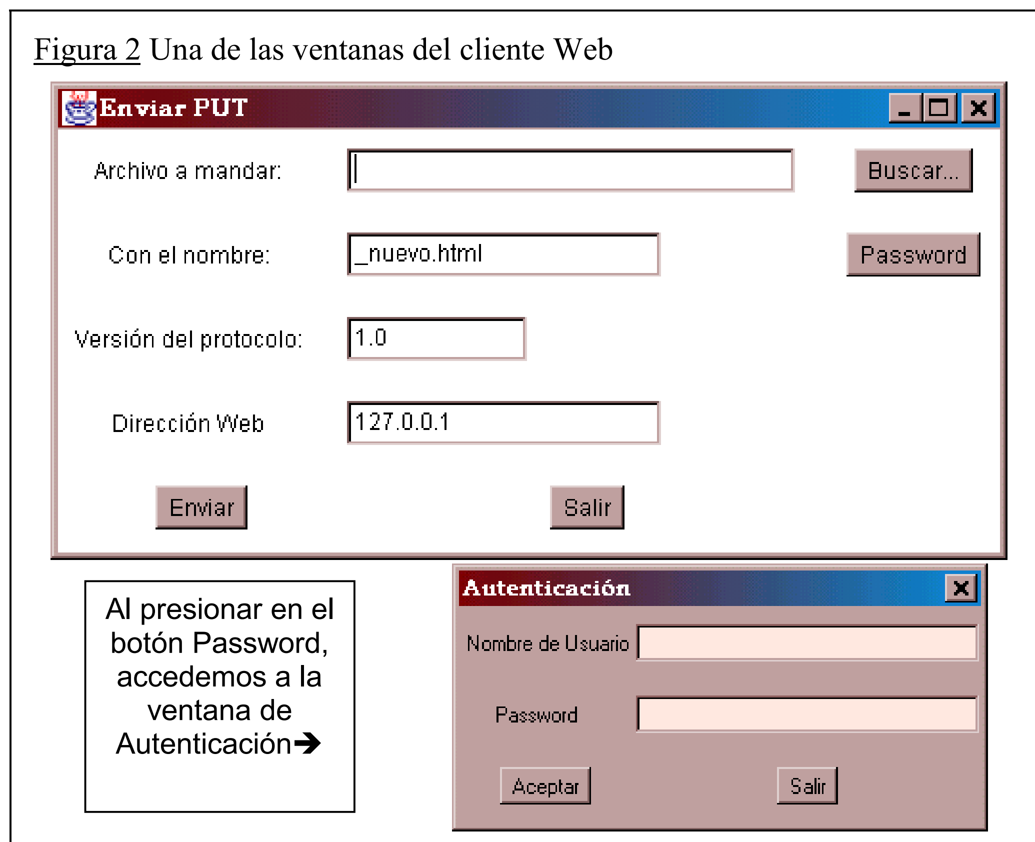
Para realizar la aplicación cliente nos basamos en las RFC, especialmente la RFC 1945 y la RFC 2068 que describen el comportamiento estándar de un servidor/cliente HTTP/1.0 y HTTP/1.1.

El cliente puede generar peticiones HTTP/0.9, HTTP/1.0 y HTTP/1.1. Los métodos que puede solicitar al servidor son: GET, HEAD, PUT, DELETE, OPTIONS y TRACE.

3.2.1. Como trabaja el cliente

El cliente busca conectarse a través del puerto 80 con el servidor especificado por el usuario. La primer línea del encabezado contiene el método solicitado, el recurso al que se le aplicará el método y el protocolo usado[9], obteniendo la información a través de una interfaz gráfica. Ver Figura 2

Figura 2 Una de las ventanas del cliente Web



[9]Ver RFC 1945 [8] y 2068 [9].

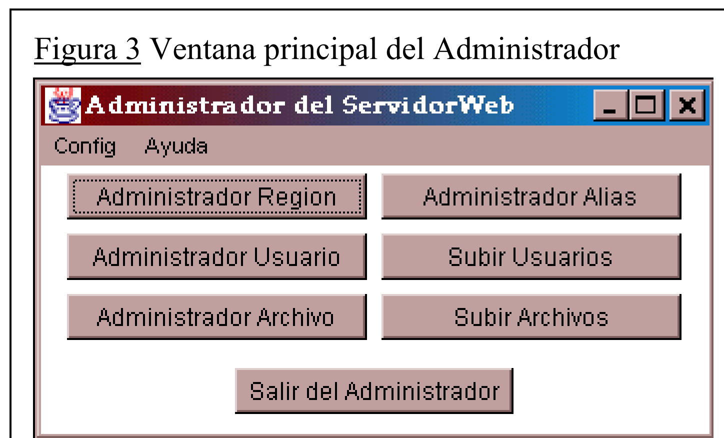
La principal característica de la aplicación cliente es que desde cualquier máquina en la que éste se ejecute se podrán almacenar o borrar recursos en el servidor (métodos put y delete respectivamente), siempre y cuando se satisfagan las restricciones de seguridad y el servidor tenga implementado el método solicitado. En caso del método PUT, el cliente obtiene el contenido del archivo a subir (upload) y lo envía como cuerpo de la petición.

3.3. Diseño del Administrador de archivos del servidor

El administrador de archivos del servidor proporciona la interfaz gráfica (Ver Figura 3) para el administrador del servidor. Este administrador permite:

- *definir distintos alias.* Con lo que se puede decidir:
    - los directorios a los que se podrán acceder a través el servidor
    - el directorio por defecto
    - el lugar de ubicación de los ejecutables, permitiendo tener mas de un directorio dedicado a ello.
- *crear regiones y usuarios.* Con lo cual se permite decidir que archivos serán de acceso exclusivo para los usuarios de la región a la que pertenece el archivo.

Figura 3 Ventana principal del Administrador



4. CONCLUSIONES Y PERSPECTIVAS

El desarrollo de nuestro servidor web, usando el lenguaje Java, nos permitió: en primer lugar, que el mismo código del servidor funcionase en Windows 95/98 y Linux sin ninguna modificación (debido a la multiplataforma de Java), y en segundo lugar, que el servidor sea fácil de actualizar y de ampliar, sin realizar prácticamente ninguna modificación en el código ya generado (debido a la orientación de objetos que tiene Java).

De manera similar, gracias a que utilizamos la especificación CGI para las aplicaciones ubicadas del lado del servidor web, queda a elección del programador el lenguaje y sistema operativo a utilizar para realizar los scripts cgi.

De todo esto concluimos que el servidor Web desarrollado puede ser instalado bajo el sistema operativo que se desee.

Asimismo pudimos observar con respecto a la aplicación realizada lo siguiente:

- Que el servidor desarrollado soporta conexiones persistente para GET, pero lo que observamos es que con Internet Explorer, aunque éste le envíe un “Keep-Alive”, al momento de realizar otro pedido establece una nueva conexión. En el caso de Netscape vimos que a veces manda la siguiente petición dentro de la conexión que ya está establecida y a veces crea una nueva conexión. A pesar que las RFC establecen que cuando el cliente no quiere mantener la conexión se debe enviar “Connection: Close”, con nuestra aplicación nunca lo pudimos observar.
- Que el cliente proporciona a través del método PUT o DELETE del http, algo similar a lo que ofrece el protocolo ftp (protocolo de transferencia de archivos).
- Que el administrador de archivos del servidor desarrollado hace más flexible el servidor debido a la facilidad para gestionar los enlaces, controlar el acceso, etc.; todo ello a través de una interfaz gráfica.

En un futuro se podría adaptar la aplicación cliente y el administrador de archivos para que puedan ser accedidos a través de Internet, tomando las respectivas consideraciones de seguridad para realizar la gestión de archivos en el servidor; como así también transformar los archivos que maneja el administrador en tablas de una base de datos.

BIBLIOGAFIA

- [1] Deitel & Deitel, 1998, *Como programar en Java,* Prentice-Hall Hispanoamericana.
- [2] Jamsa K. & Ken Cope, 1996, *Programación en Internet,* Mac Graw – Hill.
- [3] Marketos J., 1997, *The JAVA Developer's Toolkit,* Wiley Computer Publishing.
- [4] Naughton P., *Manual de Java,* Osborne-McGraw-Hill.
- [5] Riefflet Jean-Marie, *Comunicaciones en UNIX,* MacGraw-Hill.
- [6] Tittel, 1997, *La Biblia de la Programación en CGI,* Anaya Multimedia.

Páginas Web visitadas:

Multipurpose Internet Mail Extensions (MIME)

- [7] RFC 882: http://www.faqs.org/rfcs/rfc822.html.

Protocolo HTTP

- [8] RFC 1945: http://www.faqs.org/rfcs/rfc1945.html.
- [9] RFC 2068: http://www.faqs.org/rfcs/rfc2068.html
- [10] RFC 2069: http://www.faqs.org/rfcs/rfc2069.html

URL

- [13] RFC 1738: http://www.faqs.org/rfcs/rfc1738.html.

Java

- [14] Sun Microsystems: http://java.sun.com/docs/books/tutorial/security1.1/overview/index.html

HTML

- [15] RFC 1866: http://www.faqs.org/rfcs/rfc1866.html.