Informe

Proyecto 2

Micro Servidor Web

Redes de Computadoras - 2016

Matías Marzullo (80902)
Ricardo Ferro Moreno (85611)

Introducción

La propuesta del Proyecto 2 es implementar utilizando el lenguaje *C* un *Micro Servidor Web*, que sea capaz de responder peticiones de tipo *GET* manejando los archivos solicitados, los cuales podrán ser de las siguientes extensiones: *HTML*, *HTM*, *JPG*, *GIF*, *PNG* y *PHP*. Por el caso de los archivos *PHP*, el servidor será capaz de utilizar la herramienta *PHP-CGI* como intérprete de salida.

Compilación

La compilación del código fuente se realiza de forma normal mediante el compilador de GNU: GCC. Un ejemplo de compilación sería:

gcc servidorHTTP.c -o servidorHTTP

con el cual se creará el código ejecutable.

Modo de uso

La forma de invocación del programa deberá ser:

./servidorHTTP [IP][:puerto] [-h]

donde cada uno de los argumentos son opcionales.
En caso de no pasar argumentos, el servidor tomará por defecto la dirección de localhost (usualmente 127.0.0.1) y el servidor web estándar (80). Lo propio sucede si únicamente se le pasa o bien una dirección, o bien un puerto, el campo faltante tomará su valor por defecto recién mencionado.
Si el usuario usara la opción *-h*, el programa mostrará su modo de uso y dará por finalizada su ejecución, sin importar ni analizar el resto de los argumentos.

Algoritmo Principal

Para entender a alto nivel cuál es el funcionamiento que sigue el programa servidor, a continuación se presenta un algoritmo con la secuencia de pasos que se ejecutan:

```
Inicio
        Si ingresaron un -h, mostrar ayuda y cerrar.
        Sino, analizar si los parámetros brindados son correctos.
        Si los parámetros son correctos, crear un socket y ligarlo a la dirección.
        Comenzar a escuchar desde el socket
        Repetir infinitamente
                Si se recibe una conexión:
                        Crear un nuevo socket, asociarlo al proceso hijo
                        Hacer que el proceso hijo se encargue de atender el pedido
Fin
```

Algoritmo Secundario

Por otro lado, para entrar en detalle de lo que significa “Atender el pedido” en el algoritmo principal, podemos observar un segundo algoritmo que es el que se encarga de la parte de resolver la consulta (Request) enviada por el cliente. Un algoritmo a alto nivel podría ser el siguiente:

```
Inicio
    Recibir mensaje (hasta que el cliente envíe dos “ENTER” seguidos)
    Desglosar el mensaje recibido en Tipo, Ruta y Protocolo.
    Si la ruta es /, redefinir por index.html/htm/php si existiera alguno.
    Si la ruta tiene extensión php:
        Separar los archivos de la consulta (en caso de ésta exista).
    Si el Tipo, la Ruta o el Protocolo son vacíos:
        Enviar mensaje de Error 400
    sino, Si el mensaje es GET:
        Si el archivo existe:
            Si el archivo se puede abrir:
                Si el archivo es HTML/HTM:
                    Mandar headers y archivo
                sino, Si el archivo es JPG/PNG/GIF:
                    Mandar headers y archivo
                sino, Si el archivo es PHP:
                    Procesa pedido PHP
                sino, (el archivo existe pero la extensión no es válida)
                    Mandar Error 403
            sino, (el archivo no se puede abrir)
                Mandar Error 403
        sino, (el archivo no existe)
            Mandar Error 404
    sino, (el mensaje no es GET)
        Mandar Error 501
Fin
```

Procesar pedido PHP

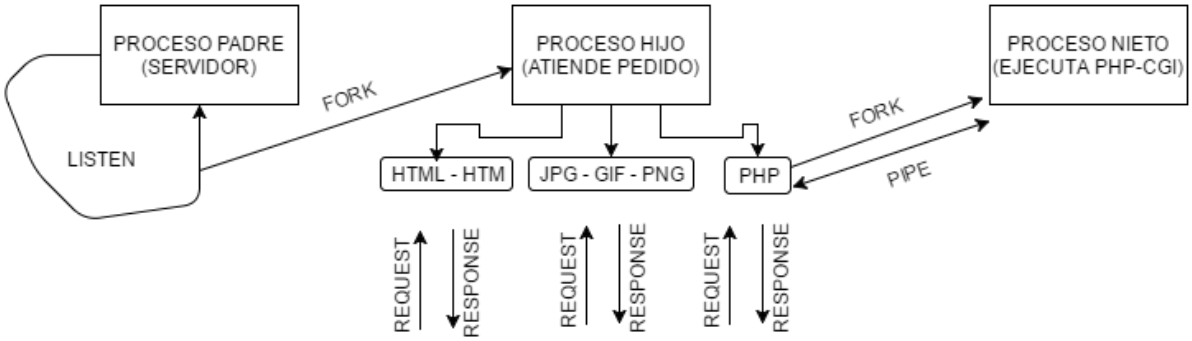
Del algoritmo anterior, conviene dejar en claro cómo se procesa un pedido PHP: A la hora de procesar un pedido PHP, se tiene la certeza de que el archivo PHP que se es solicitado existe (dado que ha llegado hasta ese punto del programa), y además para ese entonces ya se han desglosado los parámetros del Request enviado por el cliente.

Lo que realiza el método encargado de procesar el pedido PHP es dividirse en dos procesos (padre e hijo), estableciendo un canal de comunicación mediante el método ***pipe***. El proceso hijo se encargará de configurar las variables de entorno y ejecutar la herramienta ***php-cgi*** de acuerdo a la solicitud del usuario, y

una vez ejecutada la herramienta, la salida de éste proceso se enviará mediante un buffer (pipe) al proceso padre. El proceso padre se encargará de esperar por la finalización del hijo y una vez sucedido esto, cargará en un buffer propio cuál fue el mensaje resultante de su hijo y enviará un header de OK seguido del contenido del mensaje al cliente solicitante.

Representación gráfica

Con los algoritmos mencionados anteriormente, para complementar la explicación de qué sucede en el servidor durante su ciclo de vida, a continuación se refleja gráficamente cuáles serían los procesos durante el transcurso de programa.



Un proceso padre que se encarga en todo momento de escuchar por nuevos pedidos. Cuando un nuevo pedido sucede (accept), se hace un fork y el proceso hijo se encarga de atender ese pedido, de acuerdo a su extensión. Si su extensión fuera PHP, se realiza un nuevo fork para que el nuevo proceso hijo (en este caso, “nieto”) se encargue de hacer la ejecución mediante PHP-CGI y comunique su respuesta a su padre (“hijo”) a través del pipe. Una vez realizado esto, el proceso encargado de atender el pedido enviará la respuesta.

Prueba del Servidor

El servidor HTTP fue compilado y ejecutado exitosamente en la máquina virtual provista por la cátedra (Fedora 15), compilado con el compilador de GNU GCC 4.6.3, y usando la versión PHP 5.3.6 para el módulo PHP-CGI.
Por otro lado, también fue probado en Deepin (distribución basada en Debian), sin embargo en dicho sistema operativo el manejo de señales a los procesos no funciona como es esperado. El resto de funcionalidades funcionó correctamente.
Junto con los archivos fuente, se entregan algunos ejemplos de páginas html, htm, php y algunas imágenes. Recordamos que las mismas deben estar en el mismo directorio donde es puesto en marcha el servidor.

Terminación mediante SIGUSR1

Para probar la terminación del proceso mediante SIGUSR1 se puede ejecutar en la terminal la siguiente secuencia de comandos:

1. **ps -A | grep servidorHTTP** para mostrar que el proceso está corriendo y la información (pid) asociada al mismo.
2. **pkill -SIGUSR1 servidorHTTP** para enviarle la señal SIGUSR1 al proceso propiamente dicho.
3. **ps -A | grep servidorHTTP** nuevamente, para ver si el proceso finalizó su ejecución.

Registrando mensajes en el sistema

Como el proceso correrá en *background*, para guardar registro de la información y los errores que pudieran producirse se hace uso de los *System Logs*. La información del servidor y principalmente cuando se produzca un error pueden verse en el archivo del sistema operativo ubicado en ***/var/log/messages***. Por otro lado, cuando se produce un error, además de registrar el mensaje de error en el archivo recién mencionado, también se envía el error por *stderr* mediante el método ***perror*** de C.

Decisiones sobre el diseño

A continuación se enumeran algunas decisiones sobre el diseño del servidor:

- El método HTTP válido será únicamente "*GET*" en mayúsculas, ya que según se especifica en la RFC 1945, los métodos son sensibles a las mayúsculas.
- Los únicos métodos que acepta el servidor son solamente los GET. Cualquier otro método será producto de una respuesta 501 (no implementado).
- Sobre las rutas de archivos, por simplicidad decidimos que el programa sea sensible a las mayúsculas (Apache por defecto lo es, mientras que IIS no).
- Por simplicidad, para probar desde algunos navegadores, el servidor no se limitará a las peticiones únicamente HTTP/1.0. Es decir, aceptará por ejemplo peticiones HTTP/1.1, pero las responderá como si hubiera solicitado mediante el protocolo HTTP/1.0. Esto se debe a que el analizador sintáctico desarrollado en el servidor solamente se limita a analizar si la primera línea del mensaje enviado posee el método GET seguido de una ruta válida, pero no establece restricciones sobre el protocolo.
- El servidor se implementa para versión de IPv4, pero no resuelve direcciones por dominio (***gethostbyname***) y tampoco para IPv6.

- Cuando el archivo existe pero no corresponde a una extensión válida (html, htm, jpg, gif, php) decidimos mostrar un error 403. Fácilmente se podría modificar esto para mostrar el archivo solicitado, pero por cuestiones de seguridad decidimos no hacer esta acción.
- Cuando el archivo solicitado no existe, se responde un error 404 Not Found (el archivo solicitado no se encontró).
- Cuando el archivo no se puede abrir, se responde con un error 403 Forbidden (la acción solicitada está prohibida).
- Si la primera línea de solicitud del HTTP es incorrecta, entonces no se especificó la ruta, tipo de mensaje o protocolo, se enviará un error 404, no lo encontró.
- Los mensajes de error 4xx ó 5xx son enviados junto con un contenido HTML para que se vea de forma más amigable si la consulta viene desde un navegador Web.
- Los únicos headers que se envían en una respuesta son el código de estado (Status) y el tipo de contenido (Content-type). Se omiten otros tipos de headers que se podrían implementar, como por ejemplo: Content-Length, Server, Date.
- Se asume que el programa PHP-CGI existe en el path en las variables de entorno del sistema operativo. Bajo esa suposición, la ejecución de PHP-CGI se hace mediante ***execlp*** y como argumento recibe "php-cgi" (en vez de recibir "*/usr/bin/php-cgi*", el cual sería el path absoluto para sistemas de tipo Unix).
- Las únicas variables de entorno brindadas al PHP-CGI son SCRIPT_FILENAME y QUERY_STRING. Las mismas son brindadas bajo el método ***putenv*** de C.
- Para correr el programa en *background*, usamos el método ***daemon*** de C.
- Para evitar que el proceso hijo se quede en modo zombie al finalizar su ejecución, se redefinió la señal SIGCHLD con SIG_IGN (cualquier señal del hijo al padre será ignorada y luego el proceso será removido del sistema).
- Las señales que fueron redefinidas y que serán ignoradas para no abortar un proceso serán SIGUSR2, SIGABRT, SIGHUP, SIGINT, SIGQUIT y SIGTERM. El único modo de terminación de un proceso es mediante el SIGKILL propio del sistema o la señal SIGUSR1.

Métodos Utilizados

A continuación se describen métodos auxiliares implementados para modular la solución:

- **error (Mensaje)**: Muestra el mensaje de error y termina el programa con EXIT_FAILURE.
- **ayuda ()**: Muestra el mensaje de ayuda al usuario y termina el programa con EXIT_SUCCESS.
- **log_error (Mensaje)**: Registra el mensaje de error dado en el system log.

- **log_info (Mensaje)**: Registra el mensaje informativo en el system log.
- **signalHandler (sig):** Método que se encarga de manejar las señales recibidas.
- **verificarIP (servidor):** Dada una IP, verifica si la IP corresponde a una IPv4 válida.
- **verificarPuerto(puerto):** Dado un puerto, verifica si el puerto es válido (está entre 1 y 65535).
- **parseMsg (buffer, Mensaje, ruta, protocolo):** Analiza el mensaje brindado en el buffer y retorna cuál es el tipo de mensaje, la ruta del archivo y el protocolo utilizado. No se encarga de hacer chequeos en cuanto a la correctitud del mensaje.
- **archivoExiste (archivo):** Dada la ruta de un archivo, retorna verdadero si el archivo existe y falso en caso contrario.
- **archivoAbrible (archivo):** Dada la ruta de un archivo, retorna verdadero si el archivo se puede abrir y falso en caso contrario.
- **archivoSize (archivo):** Dada la ruta de un archivo, retorna el tamaño del contenido del archivo.
- **esGet (Mensaje):** Dada una cadena de texto, analiza si la misma es un "GET"
- **esBarra (Mensaje):** Dada una cadena de texto, analiza si la misma es una barra "/".
- **mandarHeader (socket, resp):** Dada una cadena de texto y un socket, envia el texto a traves de ese socket.
- **mandarHeaders (socket, tipoResp, tipoCont):** Método para encapsular y mandar 2 mensajes a través de un socket. Hace dos llamadas al método "mandarHeader", con tipoResp y con tipoCont. Normalmente se usa para mandar un tipo de respuesta (status) y el tipo de contenido a ser enviado (content type).
- **mandarRechazo (socket, tipoResp, titulo, mensaje):** Dado un socket, un tipo de respuesta, un título y un mensaje, el método se encarga de mandar una respuesta del estado de tipoResp. Y además se encarga de crear una página de contenido HTML con el título y el mensaje otorgado. El método normalmente se usa para mandar respuestas de tipo 4XX o 5XX y mostrar de forma amigable una página, para los que estén haciendo solicitudes mediante un navegador web.
- **mandarArchivo (socket, archivo):** Dado un socket y la ruta de un archivo, manda el archivo en modo binario a través del socket.
- **esHTML (archivo):** Dada una cadena de texto referente a la ubicación de un archivo, revisa si el archivo tiene la extensión HTML o HTM.
- **esJPG (archivo):** Dada una cadena de texto referente a la ubicación de un archivo, revisa si el archivo tiene la extensión JPG.
- **esGIF (archivo):** Dada una cadena de texto referente a la ubicación de un archivo, revisa si el archivo tiene la extension GIF.
- **esPNG (archivo):** Dada una cadena de texto referente a la ubicación de un archivo, revisa si el archivo tiene la extensión PNG.

- **esPHP (archivo):** Dada una cadena de texto referente a la ubicación de un archivo, revisa si el archivo tiene la extensión PHP.
- **getExtension (archivo):** ada una cadena de texto, obtiene su extensión (si la tuviese).
- **appchr (str, caracter):** Dada una cadena de texto y un caracter, agrega dicho caracter al final de la cadena de texto (append).
- **recibirMensaje (socket):** Dado un socket, se hace una lectura a través de dicho socket hasta que se lean dos enter seguidos (CR-LF o LF-CR). Luego, se retorna un string con el contenido de lo leído.
- **minusculas (str):** Dada una cadena de caracteres, devuelve la misma cadena pero convertida a minúsculas.
- **verificarPHP (archivo, argumentos):** Método para desglosar una ruta seguida de argumentos. Se utiliza para los archivos PHP que reciben parámetros, para poder separar la ruta del archivo de sus argumentos. El método retorna en su primer dato únicamente la ruta del archivo, y en su segundo argumento los parámetros dados, si los hubiera.
- **procesarPHP (socket, archivo, parámetros):** Método para atender el pedido a un archivo PHP. Dada la ruta de un archivo y sus respectivos parámetros, el método se encarga de llamar al CGI de PHP y de responder a través del socket según lo resultante de PHP-CGI.
- **atenderPedido (socket):** Método principal. Dado un socket, se encarga de atenderlo. El método se encarga de toda la inteligencia del servidor: - Obtener el mensaje, analizarlo y devolver el Response HTTP según corresponda.