Redes de Computadores - Laboratorio evaluado 2: Servidor HTTP simple

25 de mayo de 2023

Desde su creación a inicios de la década de los 90, el protocolo HTTP (Protocolo de transferencia de hipertexto) se ha convertido en una parte absolutamente esencial en la web y, por consiguiente, la internet como tal. Diseñado por el británico *sir* Tim Berners-Lee mientras trabajaba para la Organización Europea para la Investigación Nuclear (CERN por sus siglas en francés), HTTP permite el acceso a archivos de hipertexto (archivos que enlazan a otros archivos) desde cualquier cliente compatible, pudiendo acceder a documentos (generalmente utilizando el lenguaje HTML) que pueden enlazarse con otros, incluir archivos multimedia e incluso contener código que les dé interactividad, gracias a lenguajes como JavaScript.

Su trabajo para este laboratorio será implementar un servidor HTTP básico, que sea capaz de interactuar con navegadores modernos, pero sin necesitar de toda la complejidad que conlleva programar un servidor que pueda utilizarse en producción. Para esto, definiremos un subconjunto simplificado de las características definidas en la versión 1.1 de HTTP (HTTP/1.1, RFC 2068 y RFC 2616) que se pueden considerar para un programa de este tipo. El servidor deberá ser capaz de servir (entregar) cualquiera de los archivos contenidos dentro de alguna carpeta determinada por su administrador, las que estarán disponibles para cualquier usuario que se conecte con él por medio de un navegador web.

A pesar de que el servidor que desarrollen no necesariamente vaya a estar preparado para ocuparse en entornos serios, entender cómo funciona HTTP y los servidores que lo implementan puede ayudarles a entender cómo funciona la web en su nivel más fundamental.

1. Requerimientos

- El servidor debe poder correr en cualquier puerto definido por el usuario. El puerto por defecto queda a gusto de cada equipo, pudiendo ser el 80[1], 8080, 10080 o cualquier otro.
- El servidor debe ser capaz de responder a peticiones HTTP/1.1 bien definidas de tipo `GET`, entregando los códigos de respuesta correspondientes. Los códigos de respuesta obligatorios a implementar son los siguientes:
  - 200 `OK`: Se debe entregar cuando el archivo solicitado existe y no hubo problemas al procesar la solicitud.
  - 301 `Moved Permanently`: Se debe entregar cuando se desee forzar una redirección (obligar al navegador a consultar otra ruta en lugar de la solicitada), según indica el caso indicado más adelante.
    - Para esto, se debe agregar en la respuesta la cabecera `Header`, cuyo valor debe ser la nueva ruta como una URL completa (del tipo `http://localhost:1234/nueva/ruta.html`). Esta URL debe indicar completa la dirección del servidor.
    - Para propósitos de evaluación, el dominio puede definirse como un valor hardcodeado, usando el dominio `localhost` y el puerto que estén utilizando. Así, si están corriendo su servidor en el puerto 10080, la URL que contenga `Header` debe empezar con `http://localhost:10080/`.
  - 400 `Bad Request`: Se debe entregar cuando la URL solicitada es inválida. Para propósitos de esta tarea, basta que se consideren como inválidas las URL que no empiezan con `/`.
  - 404 `Not Found`: Se debe entregar cuando el archivo solicitado no se pudo encontrar.

[1] Los puertos entre 0 y 1023 se consideran reservados, por lo que se requieren permisos de administrador para poder levantar un servidor en dicho rango. Los puertos del 1024 al 65535 pueden ser utilizados por cualquiera.

  - 405 `Method Not Allowed`: Se debe entregar cuando el método que solicita el navegador sea distinto de `GET`.
  - 505 `HTTP Version Not Supported`: Se debe entregar cuando la versión HTTP que utiliza el navegador no sea la 1.1.

- No es necesario que el servidor implemente caché, cifrado (HTTPS), CORS, redirecciones (más allá de las indicadas) o *cookies*.

- El servidor no necesita procesar las cabeceras que envíe el navegador, pero sí debe utilizar al menos las siguientes cabeceras al generar sus respuestas:

  - `Content-Type`: Indica el tipo de archivo a entregar (conocido como tipo MIME). Deben soportar por lo menos los valores indicados en el siguiente ítem.
  - `Content-Length`: Indica el tamaño del archivo en bytes.
  - `Location` (sólo para código 301): Indica la nueva ruta que el navegador debe visitar. Debe ser una URL completa.

- El programa debe servir los archivos guardados dentro de algún directorio definido por el usuario.

  - Por ejemplo, si al servidor se le indica que sirva el directorio `/home/jorge/http` y un navegador intenta acceder a la ruta `/prueba/hola.html`, el programa debe abrir el archivo `/home/jorge/http/prueba/hola.html` y entregar su contenido, si es que existe.
  - Si la última parte de la URL (separando por `/`) corresponde a un directorio real en la carpeta a servir y no a un archivo, el servidor debe asumir que el usuario desea consultar el archivo `index.html` que debiera estar contenido en ese directorio. Además, si dicha ruta no termina en `/`, debe gatillar una redirección, para así evitar problemas con URL relativas. Algunos ejemplos de consultas para este caso son las siguientes:
    - `GET /`: El servidor asume que el usuario consulta por el archivo `/index.html` y lo retorna (si existe).
    - `GET /index.html`: El servidor entrega directamente el archivo `/index.html`.
    - `GET /pagina/`: El servidor asume que el usuario consulta por el archivo `/pagina/index.html` y lo retorna.
    - `GET /pagina`: Después de confirmar que `pagina` es un directorio, el servidor retorna un error 301 redirigiendo al usuario a `/pagina/`, lo cual nos lleva al caso anterior.
    - `GET /pagina/imagen.png`: El servidor entrega directamente el archivo `/pagina/imagen.png`.
  - Los archivos mínimos a soportar son documentos HTML, imágenes PNG y JPEG, *scripts* JavaScript y archivos de estilo CSS. Los valores de `Content-Type` para estos formatos son los siguientes:
    - HTML (`.html`, `.htm`): `text/html`
    - CSS (`.css`): `text/css`
    - JavaScript (`.js`): `application/javascript`
    - PNG (`.png`): `image/png`
    - JPEG (`.jpg`, `.jpeg`): `image/jpeg`
    - Para cualquier otro tipo de archivo se debe utilizar `application/octet-stream` (archivo binario genérico) o `text/plain` (archivo de texto plano genérico). Para propósitos de evaluación, no importa cuál de los dos utilice el servidor.

- Se puede utilizar cualquier lenguaje de programación para implementar el servidor, mientras se tenga acceso a una API de *sockets* TCP, ya sea provista por el sistema operativo o una biblioteca de terceros.

- **No se pueden utilizar bibliotecas ni programas que ya implementen el protocolo HTTP.**

Cualquier requerimiento que no haya quedado bien definido o esté incompleto podrá ser consultado para su clarificación o corrección, las que realizarán sobre este mismo documento.

2. Pauta de evaluación

Para evaluar este laboratorio se utilizará una escala de 6 puntos, con nota mínima 1.0. Este trabajo puede realizarse de forma individual o de a pares.

Los puntos a evaluar son los siguientes:

- **Funcionamiento base** (2 puntos): El servidor puede responder a peticiones HTTP del tipo `GET` y generar respuestas entendibles por el navegador.
- **Entrega de archivos** (2 puntos): El servidor puede entregar íntegramente los archivos (desde cualquier subdirectorio) que le solicite el navegador, utilizando su `Content-Type` correspondiente e indicando su tamaño, lo que incluye la entrega de archivos `index.html` implícitos, como se indica en la sección anterior. Además, el servidor podrá entregar todos los tipos de archivo indicados en la sección Requerimientos.
- **Códigos de error** (1 punto): En caso de que alguna respuesta no tenga código 200 (ya que no fue posible enviar el archivo o hubo algún error en la aplicación), el servidor deberá responder con el código de error correspondiente (dentro de los indicados anteriormente) y un pequeño archivo HTML que le indique al usuario qué sucedió, indicando al menos el código HTTP y su nombre. Los archivos de error pueden estar almacenados en una carpeta aparte dentro del proyecto o usarse *strings* hardcodeados. Los errores extremadamente fatales (como errores de *socket* o fallas del sistema operativo) pueden no generar una respuesta, debido a su naturaleza.
- **Documentación** (0.5 puntos): El código entregado incluye instrucciones para compilar y ejecutar el servidor, por medio de un archivo `README`.
- **Demostración** (0.5 puntos): El funcionamiento del servidor es demostrado en video, pudiéndose ver la visita de alguna página entregada por el servidor.

De forma adicional, los siguientes ítems complementarán el puntaje obtenido en los puntos obligatorios, hasta completar 6 puntos:

- **Limpieza de código** (0.5 puntos): El código del programa tiene una sintaxis y estructura clara, consistente y limpia, indentando consistentemente el código, separando secciones en párrafos y estructurando el código en funciones o clases con propósitos bien definidos.
  - **Pista**: Para el formato del código, pueden utilizar herramientas como `clang-format` (C/C++), `autopep8` (Python) o `rustfmt` (Rust), dependiendo del lenguaje que quieran utilizar. Esta clase de utilidades pueden ser integradas en editores de código como Visual Studio Code, (Neo)vim o Emacs.
- **Compilación automatizada** (0.5 puntos): El proyecto utiliza un *toolchain* de compilación, que se encarga de ejecutar los pasos de compilación de forma automática, sin tener que llamar manualmente al compilador para generar los ejecutables. Este ítem sólo puede aplicarse a proyectos que usen lenguajes compilados.
  - **Pista**: Dependiendo del lenguaje, pueden utilizar herramientas como CMake, `autotools` o Cargo.
- **Cifrado con HTTPS** (1 punto): El servidor puede responder a peticiones cifradas (HTTPS, HTTP seguro) sobre el puerto 443 u otro, utilizando TLS por encima de los mensajes HTTP.
  - **Pista**: Considerando lo complejo que es implementar SSL/TLS, pueden utilizar bibliotecas como OpenSSL para implementar este ítem.
- **Implementar código 418** `I'm a teapot` (0.1 puntos): Un verdadero clásico en desarrollo web, la RFC 2324 especifica el protocolo HTCPCP (Protocolo de control de cafeteras de hipertexto), concebido como broma del Día de los Inocentes en los Estados Unidos (1 de abril). Este protocolo define cómo debiera funcionar una cafetera conectada a la red, que puede preparar café mediante peticiones HTTP. Para conseguir esta décima, el servidor debe responder con el código de error 418 `I'm a teapot` a cualquier petición con el verbo `BREW` (preparar [café]) a cualquier ruta. Así, le indicamos al cliente que nuestro servidor web no puede servir café, porque es una tetera.

3. Entrega

El proyecto deberá entregarse a más tardar el 22 de junio de 2023 a las 23.59 h a través de Canvas. La no entrega sin su debida justificación a tiempo conlleva nota NCR. Se aceptarán preguntas sobre el proyecto y correcciones sobre este enunciado durante las sesiones de laboratorio y por Teams o correo hasta el día 20 de junio.

La entrega de este laboratorio debe incluir:

- Un comprimido (en cualquier formato) conteniendo el código fuente del proyecto o un enlace a algún repositorio público (GitHub, GitLab, BitBucket, etc.) que lo contenga. Este comprimido o repositorio debe contener un `README` indicando las instrucciones para compilar el código (de ser necesario) y el procedimiento para levantar el servidor.
- Un video de demostración (idealmente no mayor a 5 minutos) que muestre el funcionamiento del servidor, incluyendo la salida que reporte y el acceso a algún contenido hospedado por el servidor desde un navegador web. El video puede subirse como privado o no listado si así lo desean.