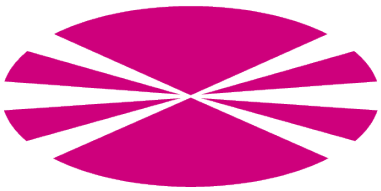


UNIVERSIDADE DA CORUÑA

LABORATORIO DE REDES

Práctica 1: Implementación de un Servidor Web

Objetivos

Esta práctica consiste en el desarrollo de un servidor web **multi-thread**, capaz de procesar múltiples solicitudes simultáneas en paralelo. El objetivo final será obtener un servidor web capaz de interactuar con un cliente para navegar a través de un sitio web. La implementación se repartirá en dos iteraciones.

Para la realización del servidor web se utilizará el protocolo HTTP versión 1.0 (definido en el RFC 1945), en donde se generan solicitudes HTTP independientes para cada componente de la página web. El servidor procesará múltiples solicitudes en paralelo, utilizando para ello múltiples hilos de ejecución.

En el hilo principal, el servidor permanece escuchando en un puerto fijo y, cuando se recibe una petición de conexión TCP, se establece la conexión a través de otro puerto y se resuelve el servicio en un hilo de ejecución separado.

Para la realización de la práctica se recomienda una aproximación en dos etapas. En una primera etapa, el servidor multi-thread únicamente mostrará por pantalla el contenido de las solicitudes HTTP recibidas, para, a continuación, (una vez que el programa se ejecuta convenientemente) incluir el código que genere la respuesta adecuada. Para comprobar el correcto funcionamiento del servidor se recomienda utilizar el comando `nc` para conectarse al puerto de escucha, e introducir directamente por línea de comandos las órdenes HTTP.

El servidor deberá lanzarse en un puerto no reservado (i.e. por encima del puerto 1024), y en el cliente se le deberá indicar el fichero que se está solicitando. A modo de ejemplo, una URL que se podría solicitar desde el navegador tendría el siguiente formato: http://localhost:1111/index.html. El servidor web localizará el fichero solicitado y será devuelto al cliente en el formato adecuado. En caso de que se produzca algún error (por ejemplo, el fichero solicitado no se encuentra), el servidor deberá responder con el código HTTP adecuado para cada situación errónea y una página HTML para el error.

La **iteración 1** (5 puntos) de esta práctica debe incluir:

- Implementación de un servidor web multi-thread.
- Implementación de los métodos GET y HEAD.
- Soporte de los formatos básicos HTML, texto plano, GIF y PNG.

Además, la **iteración 2** (5 puntos) incluirá además los siguientes **apartados**:

- Implementación de la opción “If-Modified-Since” para el método GET (1 punto), que se deberá activar a través del navegador web mediante la recarga de la página (botón “Actualizar”). De esta manera, el navegador solicitará al servidor web la página web con la opción “If-Modified-Since”, descargando de nuevo la página si ha habido cambios desde la última modificación. Para verificar el correcto funcionamiento de esta opción se recomienda utilizar la opción de Red de las Herramientas para Desarrolladores disponibles tanto para Firefox como para Chrome.

- Definición de dos ficheros de log (0,5 puntos): para accesos y para errores. En el fichero de accesos únicamente se guardará un registro de los accesos que hayan provocado una respuesta con código de estado 2xx o 3xx. Aquellas peticiones que provoquen un error 4xx (o un error 5xx, en caso de implementarse) serán almacenadas en el fichero de errores.
    - El fichero de log para accesos tendrá el siguiente formato:
        - Línea de petición del mensaje HTTP enviado por el cliente.
        - Dirección IP del cliente que realiza la solicitud.
        - Fecha y hora en la que se recibió la petición: [día/mes/año hora:minuto:segundo zona_horaria]
        - Código de estado que el servidor envía como respuesta al cliente.
        - Tamaño (en bytes) del recurso enviado al cliente.
    - El fichero de log para errores tendrá el siguiente formato:
        - Línea de petición del mensaje HTTP enviado por el cliente.
        - Dirección IP del cliente que generó el error.
        - Fecha y hora en la que se produjo el error (mismo formato que para el fichero de log de accesos).
        - Mensaje de error.
- Configuración básica del servidor web (1,5 puntos): el servidor web dispondrá de un **fichero de configuración** a través del cual se podrán indicar, al menos, los siguientes parámetros (se recomienda utilizar la clase java.util.Properties para el desarrollo de esta opción):
    - Parámetro PORT: Puerto en el que escucha el servidor.
    - Parámetro DIRECTORY_INDEX: Nombre del fichero a buscar por defecto en caso de recibir una petición a un directorio (por ejemplo, petición al directorio / por medio de http://localhost:1111/). Se buscará dicho fichero bajo el directorio o subdirectorio del que se ha recibido la petición (en el ejemplo, bajo el / del servidor).
    - Parámetro DIRECTORY: Directorio raíz del servidor web, en donde se ubicarán las páginas web.
    - Directiva ALLOW: Directiva que se aplicará cuando se reciba una petición a un directorio. Funcionará de la siguiente manera:
        - Si en el directorio existe el fichero por defecto (indicado en el DIRECTORY_INDEX), se mostrará el fichero por defecto.
        - Si en el directorio no existe el fichero por defecto y la directiva ALLOW está activada, se generará un documento

HTML de manera dinámica que mostrará el listado de todos los archivos/directorios que lo componen, con un enlace que permitirá abrir cada fichero y acceder a cada uno de los subdirectorios, aplicando, en este último caso, la directiva ALLOW recursivamente.

        - Si en el directorio no existe el fichero por defecto y la directiva ALLOW no está activada, se mostrará el error 403 (Access Forbidden).

- Javadoc (0,5 puntos): se deberán documentar (en inglés) las clases java presentes en el repositorio y que hayan sido implementadas por el alumno (sin incluir las proporcionadas en el fichero p1-files.zip). En concreto se pide lo siguiente:
    - A nivel de clase, se deberá incluir un comentario explicando el cometido de la misma, y se deberá añadir la etiqueta @author.
    - A nivel de método, se deberá explicar el objetivo del mismo, y se deberá emplear una etiqueta @param por cada parámetro, la etiqueta @return (explicando el valor devuelto por el método) y una etiqueta @throws por cada excepción devuelta (indicando en qué situaciones se obtendrá la excepción).
    - El alumno deberá poder generar (a la hora de la defensa) la documentación, bien usando el comando javadoc, bien empleando las herramientas disponibles en el IDE utilizado (Netbeans).
- Páginas dinámicas (1,5 puntos): el servidor web deberá poder generar páginas dinámicas en función de los parámetros que reciba en la petición HTTP. Llamaremos página dinámica a aquella cuyo contenido es generado dinámicamente en el servidor para cada petición. Para el propósito de este apartado, crearemos una clase Java por cada una de estas páginas. Por convención, emplearemos **.do** como extensión para identificar este tipo de peticiones.
    - Con el enunciado de la práctica se proporciona un fichero **p1-files.zip** con un ejemplo formado por los siguientes recursos:
        - saludo.html: formulario ejemplo que generará una petición GET con diferentes parámetros.
        - ServerUtils.java: una clase utilidad carga dinámicamente la clase que atenderá a la petición HTTP.
        - MiniServlet.java: una interfaz que deberán implementar todas las clases Java encargadas de generar los contenidos dinámicos de la respuesta HTTP.
        - MiServlet.java: una clase de ejemplo que implementa la interfaz.

- Los ficheros java proporcionados han de ser copiados en el directorio **java-labs/src/es/udc/redes/webserver**, en donde debería figurar ya el archivo WebServer.java. El resto de ficheros (udc.gif, fic.png, saludo.html e index.html) deberán copiarse en el directorio raíz de los recursos proporcionados por el servidor.
- Se podrán implementar tantos ficheros .java como se consideren necesarios. Todos ellos tendrán que estar en el paquete es.udc.redes.webserver, o en un subpaquete del mismo.
- El objetivo es entender el ejemplo que se proporciona, desarrollar al menos una clase que implemente la interfaz (MiniServlet.java) y modificar el formulario proporcionado (o crear uno nuevo) para que se generen nuevas peticiones a páginas dinámicas (que serán atendidas por la clase implementada). Además, se deberá hacer una gestión correcta de las excepciones.
- El servidor ha de ser capaz de identificar qué clase ha de atender qué petición. Así, por ejemplo, para atender a la petición http://localhost:1111/TuServlet.do, se buscará la clase TuServlet.class, puesto que el recurso a solicitar representa el nombre de la clase.

La práctica deberá ser realizada utilizando las funcionalidades propias de los sockets en Java, sin utilizar ninguna clase que implemente total o parcialmente las características del protocolo HTTP (como, por ejemplo, la clase HttpURLConnection).

Introducción a HTTP

El protocolo HTTP (HyperText Transfer Protocol) está especificado en el RFC 1945 (versión 1.0) y en el RFC 2616 (versión 1.1). Este protocolo define cómo los clientes (navegadores) solicitan páginas web a los servidores web, y cómo éstos realizan la transferencia de estas páginas.

HTTP se basa en el protocolo TCP (que ofrece un servicio orientado a conexión y fiable), lo que garantiza que cada mensaje HTTP emitido por el cliente o el servidor es recibido en el otro extremo sin sufrir modificaciones. Además, HTTP es un protocolo sin estado, esto es, el servidor HTTP no almacena ningún tipo de información sobre sus clientes. Cada petición recibida por el servidor se trata independientemente de las peticiones anteriormente recibidas de ese u otros clientes.

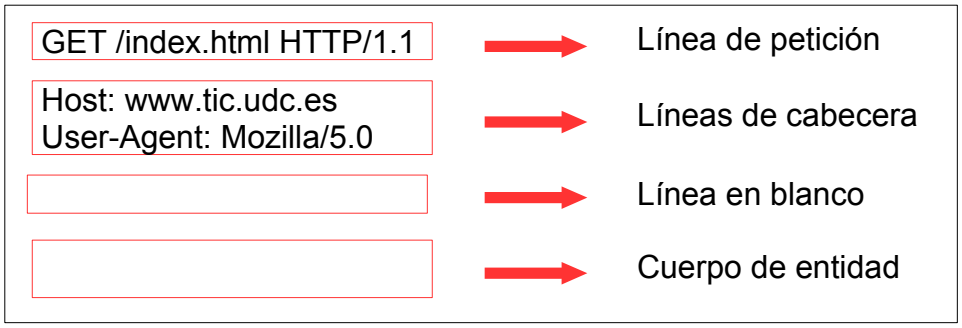
El protocolo HTTP 1.0 utiliza conexiones no persistentes, ya que es necesario establecer una conexión TCP independiente para cada uno de los componentes de una página web. De manera esquemática, el procedimiento para la petición de una URL (p.e. http://www.tic.udc.es/index.html) sería el siguiente:

1. El cliente HTTP inicia la conexión TCP con el servidor www.tic.udc.es en el puerto 80.
2. El cliente HTTP envía al servidor el mensaje de petición solicitando el recurso /index.html
3. El servidor HTTP recibe la petición, lee el recurso, lo encapsula en el mensaje HTTP de respuesta y lo envía.
4. El servidor finaliza la conexión TCP.
5. El cliente HTTP recibe la respuesta y finaliza la conexión TCP.
6. El cliente extrae el archivo del mensaje de respuesta, examina el archivo HTML y encuentra referencias a otros recursos HTML (p.e. imágenes)
7. Volver al paso 1, para cada uno de los nuevos recursos HTML.

El principal inconveniente de esta aproximación se centra en el retardo introducido, ya que es necesario esperar dos veces el RTT (Round Trip Time), uno para el establecimiento de la conexión y otro para la petición y respuesta del recurso solicitado. Además, el hecho de utilizar una conexión para cada recurso solicitado implica una mayor utilización de recursos (buffers, variables, timeouts…), tanto en el cliente como en el servidor.

Estos problemas se resuelven mediante el protocolo HTTP 1.1, que utiliza conexiones persistentes, en donde el servidor deja abiertas las conexiones TCP establecidas en espera de nuevas peticiones. Tras un período de inactividad estas conexiones se cierran por parte del servidor.

El protocolo HTTP sigue un sencillo modelo de peticiones y respuestas. El formato de una petición se puede observar en la siguiente figura. Los únicos campos obligatorios son la línea de petición y la línea en blanco.



La línea de petición especifica el tipo de petición que se está realizando, y está formada por tres campos:

- Método
- URL: recurso al que se hace referencia.
- Versión: versión del protocolo HTTP utilizada por el navegador.

Los métodos básicos definidos en el protocolo HTTP son los siguientes:

- GET: solicitud de un recurso por parte del cliente.
- HEAD: el cliente solicita al servidor metainformación de un recurso. El mensaje HTTP devuelto por el servidor será el mismo que en el caso del método GET, pero incluyendo solo la información de cabeceras (el cuerpo del mensaje se devolverá vacío).
- POST: permite incluir datos en el cuerpo de entidad.
- PUT: permite a un cliente cargar un archivo en la ruta especificada (solo en HTTP 1.1).
- DELETE: permite a un cliente borrar un archivo de un servidor (solo en HTTP 1.1).

Dentro las líneas de cabecera, un navegador típico puede incluir múltiples opciones, siendo las más relevantes:

- Host: especifica el host en el que reside el recurso solicitado.
- User-Agent: especifica el tipo de cliente que está haciendo la petición.
- If-Modified-Since: utilizada junto con el método GET, especifica una fecha para que el servidor, si el recurso no ha sido modificado con posterioridad a esa fecha, no lo envíe de nuevo.

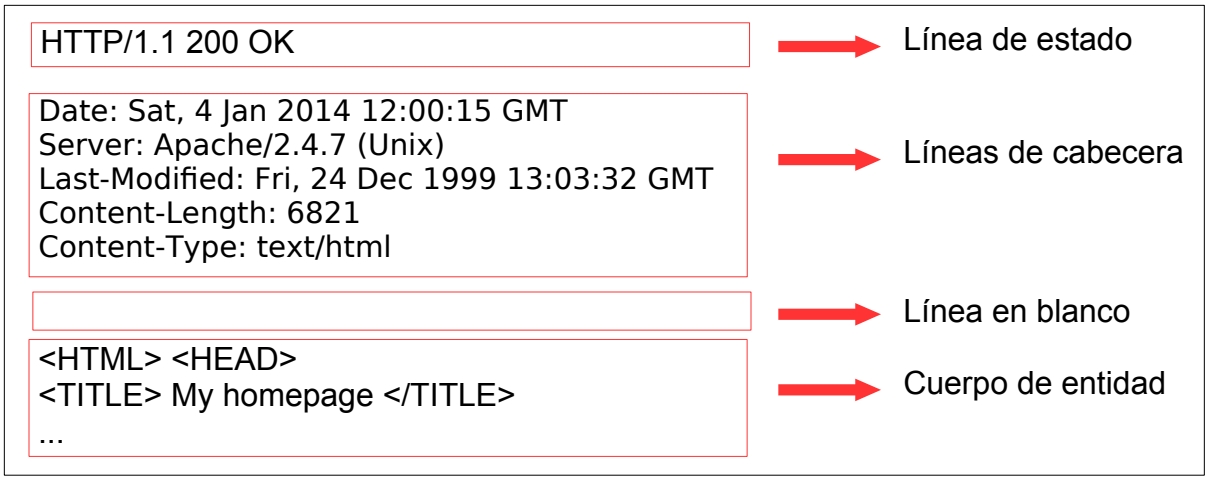
La opción "*If-Modified-Since*" es utilizada en las implementaciones de las cachés en el cliente. El formato de la petición sería el siguiente:

```
GET /images/udc.gif HTTP/1.1
User-Agent: Mozilla/5.0
If-Modified-Since: Fri, 10 Jan 2014 13:03:32 GMT
```

Mientras que en el servidor, si el recurso no ha sido modificado, la respuesta sería la siguiente:

```
HTTP/1.1 304 Not Modified
Date: Wed, 5 Feb 2014 20:30:43 GMT
Server: Apache/2.4.7 (Unix)
```

El formato de las respuestas HTTP se puede observar en la siguiente figura.



La línea de estado incluye tres campos:

- Versión: versión utilizada por el servidor web.
- Código de estado: código numérico que representa si la respuesta ante la petición es satisfactoria o si ha habido algún error.
- Frase: asociada a cada código numérico existe una frase que informa sobre la naturaleza del código. En el RFC 1945 se pueden consultar todos los códigos y frases del protocolo, destacando los siguientes:
    - 200 OK
    - 400 Bad Request: petición no comprendida por el servidor.
    - 403 Forbidden: petición comprendida por el servidor, pero que rechaza satisfacer.
    - 404 Not Found: el recurso solicitado no existe en el servidor.

Respecto a las líneas de cabecera, existen múltiples parámetros que el servidor puede especificar, entre los que destacan:

- Date: fecha y hora en la que se creó y envió la respuesta HTTP.
- Server: especifica el tipo de servidor web que ha atendido la petición.
- Content-Length: indica el número de bytes del recurso enviado.
- Content-Type: indica el tipo de recurso incluido en el cuerpo de entidad. Este campo es necesario, ya que la extensión del archivo no especifica (formalmente) el tipo de recurso asociado. Los tipos más comúnmente utilizados son:
    - text/html: indica que la respuesta está en formato HTML.

  - text/plain: indica que la respuesta está en texto plano.
  - image/gif: indica que se trata de una imagen en formato gif.
  - image/png: indica que se trata de una imagen en formato png.
  - application/octet-stream: utilizado cuando no se identifica el formato del archivo.
- Last-Modified: indica la fecha y hora en que el recurso fue creado o modificado por última vez.

Para la realización de esta práctica la línea de estado deberá estar presente en todas las respuestas generadas y las cuatro primeras líneas de cabecera (Date, Server, Content-Length y Content-Type) se enviarán siempre que sea posible.

La cabecera Last-Modified será necesaria si se implementa el apartado de If-Modified-Since.

¿Cómo probar la iteración 1 de la práctica?

1. Arrancar un nc al puerto del servidor y dejarlo conectado (para probar que es multithread).
2. Abrir un navegador y cargar una página HTML que incluya texto plano, imágenes gif y png.
3. Si se muestra la página => Multithread OK, GET OK, formatos básicos OK
4. Volver al nc y enviar una petición HEAD (p.e. HEAD /index.html HTTP/1.0).
5. Comprobar que devuelve la línea de estado con las líneas de cabecera => HEAD OK.
6. Comprobar que se envían correctamente las cabeceras Date, Server, Content-Type, Content-Length y Last-Modified (si se ha implementado el apartado de If-Modified-Since) => Parámetros cabecera HTTP OK.
7. Arrancar otro nc al puerto del servidor y enviar una petición incorrecta (p.e. GETO /index.html HTTP/1.0): Comprobar que devuelve un error 400 Bad Request + una página de error (con las líneas de cabecera correctas) => Bad Request OK.
8. Enviar una petición a un recurso que no exista (p.e. GET /indeeex.html HTTP/1.0): Comprobar que devuelve un error 404 Not Found + una página de error (con las líneas de cabecera correctas) => Not Found OK.
9. Se proporciona una aplicación Java y un conjunto de ficheros para automatizar la validación de todos los apartados anteriores. Por tanto, comprobar de nuevo todos los apartados empleando el fichero httptester.jar (*java -jar httptester.jar <host> <puerto> [<0-9>]*). NOTA: los ficheros fic.png, index.html, LICENSE.txt y udc.gif (contenidos dentro del archivo p1-files.zip) han de situarse en el directorio raíz bajo el que cuelguen los ficheros proporcionados por el servidor web.

ENTREGA

Esta práctica se realizará desde el 17 de febrero al día 20 de marzo de 2020.

Para la evaluación de la misma se emplearán los ficheros subidos por el alumno a su repositorio de prácticas en gitlab:

https://git.fic.udc.es/<user-login>/java-labs.

Para indicar cuál es la versión de la práctica que debe ser evaluada, tras realizar los últimos cambios para la entrega se deberá etiquetar el commit y posteriormente publicar el tag ejecutando los siguientes comandos:

1. `git tag p1`
2. `git push origin p1`

Aunque no es obligatorio, se recomienda crear un tag “it1” para etiquetar el commit en el que se haya implementado por completo el contenido de la primera iteración.

La fecha límite para subir la práctica al repositorio será el día 20 de marzo de 2020 a las 20:00. Una vez entregada, el alumno deberá mostrar al profesor su correcto funcionamiento para poder considerar la práctica como apta. Esta corrección tendrá lugar en la clase de prácticas de la semana del 23 de marzo de 2020.

Aquellos ficheros que sean subidos al repositorio después de la fecha límite NO serán tenidos en cuenta en la evaluación. La defensa de la práctica se realizará en el grupo de prácticas habitual. En caso de hacerlo en un grupo posterior se podrá considerar que la práctica ha sido entregada fuera de plazo.

Para la presentación de esta práctica, se mostrará su funcionamiento en el laboratorio de prácticas y el alumno deberá ser capaz de explicar cualquier componente de la misma.

En caso de no subir al repositorio la práctica en plazo o en caso de que no exista en el repositorio el tag p1, se considerará que la práctica NO ha sido presentada.

Durante la defensa, se comprobará parte de la funcionalidad de la práctica empleando el fichero httptester.jar, por lo que se aconseja haberlo ejecutado previamente antes de la defensa. Cada una de las iteraciones supondrán hasta 0,5 puntos en la nota final de la asignatura.