Práctica 1: Servidor Web

1. Introducción

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 navegador para mostrar recursos.

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 recibe una petición de conexión TCP, se establece la conexión a través de otro puerto y se resuelve en un hilo de ejecución por separado.

Para la implementación se recomienda partir de un servidor de Eco TCP como el desarrollado en el último apartado del "Tutorial de Sockets" e ir desarrollando el código que genere las respuestas a diferentes peticiones.

2. Objetivos

Desarrollo de un servidor Web que cumpla los siguientes requisitos:

1. Acepte peticiones de manera concurrente (*multi-thread*)
2. Soporte peticiones GET y HEAD y controle la respuesta en caso de realizar otra petición no soportada.
3. Soportar peticiones mal formadas (e.g. número incorrecto de campos en la línea de petición).
4. Soportar ficheros TXT, HTML, GIF y JPEG.
5. Incluir las cabeceras *"Server"* y *"Date"* siempre que sea posible.
6. Incluir *"Content-Length"*, *"Content-Type"* y *"Last-Modified"* en las respuestas a peticiones GET y HEAD.

El funcionamiento del servidor será evaluado y corregido como se indica en el apartado 5.

3. 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 y como éstos realizan la transferencia de estas páginas.

El protocolo 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 (e.g. http://www.fic.udc.es/) sería el siguiente:

1. El cliente HTTP (navegador) 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 (e.g. imágenes).
7. Volver al paso 1 para cada uno de los nuevos recursos encontrados.

3.1. Especificación Protocolo

El protocolo HTTP sigue un sencillo modelo de peticiones y respuestas. El formato de las peticiones y respuestas se especifican en los siguientes apartados.

3.1.1. Formato Petición

Los únicos campos obligatorios son la línea de petición y la línea en blanco que indica el final de las líneas de petición y cabecera.

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

- Método

- URL
- Versión

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 introducir 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).

El campo URL hace referencia al recurso que se solicita en el servidor y la versión indica cual cumple el cliente que realiza la petición.

El servidor web desarrollado en esta práctica debe soportar los métodos GET y HEAD en la versión HTTP 1.0.

A continuación y antes de la línea en blanco se pueden incluir diferentes cabeceras, entre ellas algunas de las más comunes son:

- *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 no envíe de nuevo el recurso si no ha sido modificado con posterioridad a esa fecha.

Se incluyen estos ejemplos de manera informativa pero para el desarrollo de esta práctica no es necesario procesar el contenido de las mismas.

3.1.2. Formato Respuesta

El formato de las respuestas en HTTP está formado por cuatro elementos:

1. Línea de estado
2. Líneas de cabecera
3. Línea en blanco
4. Cuerpo de la entidad

La línea de estado contiene un código numérico que representa si la respuesta ante la petición es satisfactoria o si ha habido algún error y una frase asociada. Estos códigos y frases están especificados en el RFC 1945 pero para lo relativo al desarrollo de esta práctica se pueden resaltar los siguientes:

- "200 OK" → Petición correcta.
- "400 Bad Request" → Petición no comprendida por el servidor.
- "404 Not Found" → El recurso solicitado no existe en el servidor.

Las líneas de cabecera permiten especificar múltiples parámetros relativos a la respuesta generada, entre otros se pueden destacar:

- *Date*: fecha 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 la entidad. Los tipos más comúnmente usados y necesarios para el desarrollo de la práctica son los siguientes:
  - *text/txt*: indica que la respuesta está en texto plano.
  - *text/html*: indica que la respuesta está en formato HTML.
  - *image/gif*: indica que la imagen está en formato gif.
  - *image/jpeg*: indica que la imagen está en formato jpeg/jpg.
  - *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 construcción de las respuestas de esta práctica la línea de estado debe estar presente en todos los casos y se enviarán el mayor número de cabeceras posibles de las cinco mencionadas.

4. Iteraciones

Se propone abordar el desarrollo de las funcionalidades requeridas en el servidor de la forma indicada en los siguientes apartados.

4.1. Iteración 1

Partiendo del servidor multi-hilo desarrollado en el tutorial de introducción a *sockets* analizar el mensaje recibido para comprobar si está bien formado y determinar si es alguna de las peticiones que soportaremos. No es necesario procesar las líneas de cabecera enviadas por el cliente. El siguiente paso será enviar el contenido del recurso solicitado para un fichero HTML o TXT, pudiendo visualizar la respuesta mediante un navegador web. En este punto deberían estar correctas al menos las cuatro primeras pruebas del test (apartado 5).

4.2. Iteración 2

En esta iteración se añadirán las cabeceras necesarias para todas las respuestas generadas en la iteración anterior, tras lo que deberían cumplirse las ocho primeras pruebas del test (apartado 5).

4.3. Iteración 3

En esta última iteración se añadirá el soporte para envío de imágenes, esto es, de ficheros binarios. Una vez realizada esta iteración deberían estar correctas todas las pruebas realizadas por el *script* (apartado 5) proporcionado.

5. Evaluación y pruebas

Para la evaluación de esta práctica se requerirá haber realizado **al menos un *commit* por iteración propuesta** (i.e. 3 *commits*). La puntuación correspondiente a esta práctica es de **0.65** sobre la nota final, repartiéndose entre la implementación de cada iteración y su correcta defensa.

Para la evaluación de esta práctica se proporciona un *script* de Python que simplifica y automatiza la realización de pruebas sobre la misma. En cualquier caso se recomienda hacer uso del comando *netcat* como se explica en el tutorial de *sockets* para comprobar las peticiones y respuestas, así como de un navegador web. A modo de ejemplo, el resultado del *script* proporcionado, en caso de pasar las pruebas debería ser similar al siguiente:

```
Comprobando servidor: http://server:8080/
====================
Multihilo..................... OK
Petición no soportada......... OK
Petición incorrecta........... OK
Fichero no encontrado......... OK
Head TXT...................... OK
Get TXT....................... OK
Head HTML..................... OK
Get HTML...................... OK
Head JPG...................... OK
```

```
Get JPG..................... OK
Head GIF.................... OK
Get GIF..................... OK

Puntuación: 12/12
```

La entrega se realizará, al igual que la práctica anterior en el repositorio Git, siendo la **fecha límite las 23:59 del 22/03/2021 (CET o UTC+01:00)**. Las defensas se realizarán en la semana del 23 al 26 de marzo.

Adicionalmente, esta práctica podría realizarse y defenderse en un momento anterior a la fecha de la entrega con acuerdo previo con el profesor de prácticas.