Práctica 4: Servidor HTTP

<table>
<tr><td colspan="2">UNIDAD DE APRENDIZAJE : Aplicaciones para comunicaciones en red
UNIDAD TEMÁTICA III: Arquitectura Cliente-Servidor</td></tr>
<tr><td>No. Y Título de la práctica:

Práctica no. 5 Servidor HTTP</td><td rowspan="2">Tiempo de realización: 4.5 horas</td></tr>
<tr><td>Objetivo de la práctica: El estudiante implementará un servidor FTP basándose en sockets de flujo, hilos y principios de funcionamiento del protocolo HTTP</td></tr>
<tr><td colspan="2">Situación problemática: El IETF (Internet Engineering Task Force) es un grupo de trabajo encargado de validar y aprobar todos los protocolos que se usan en Internet para garantizar la interoperabilidad entre aplicaciones en red independientemente de arquitecturas, sistemas operativos, lenguajes de programación, etc. Para lograr este objetivo desarrollan especificaciones llamadas RFC (Request For Comments). Una ventaja de contar con la especificación de un protocolo es el poder implementarla o modificar algunas de sus características basándonos en una implementación. En este caso nos gustaría implementar una pequeña versión de un Servidor HTTP la cual se ajuste a la especificación del RFC 2616
¿Qué tipo de socket conviene usar para implementar un servidor HTTP? ¿por qué?</td></tr>
<tr><td colspan="2">Competencia específica: Desarrolla un servidor HTTP, con base en el modelo cliente-servidor y utilizando la interfaz de sockets, así como hilos.</td></tr>
<tr><td>Competencias genéricas:
Aplica los conocimientos en la práctica
Demuestra habilidad para trabajar en equipo
Demuestra capacidad de investigación
Desarrolla aplicaciones en red con base en la tecnología más adecuada</td><td>Elementos de competencia:
Programa aplicaciones en red con base en el modelo Cliente-Servidor y la interfaz de aplicaciones de sockets de flujo
Analiza los servicios definidos en la capa de transporte
Emplea el modelo Cliente-Servidor para construir aplicaciones en red
Programa aplicaciones Cliente-Servidor utilizando sockets de flujo bloqueantes</td></tr>
<tr><td colspan="2">Criterios de evaluación: La práctica 5 aportará el 10% de la unidad temática III</td></tr>
</table>

Rúbrica (analítica) para la U.A. Aplicaciones para comunicaciones en red.

Producto: Servidor HTTP

Valoración: **Novato (0-150pts), Intermedio (151-300 pts), Avanzado (301-450 pts), Experto (451-600 pts)**

ASPECTOS A EVALUAR	Excelente (100pts)	Cumplió bien (75pts)	Cumplió (50pts)	No satisfactorio(25pts)
Análisis	Entiende el problema a resolver, haciendo uso de los elementos de programación en red precisos	Utiliza los elementos de programación en red necesarios	Utiliza algunos de los elementos de programación en red	No tiene idea de cómo resolver el problema, ni que tipo de comunicación necesita implementar
Diseño	Define una arquitectura de comunicación en base al tipo de comunicación a utilizar	Define una arquitectura de comunicación que puede o no ser la mejor opción para el tipo de comunicación a utilizar	Define una arquitectura de comunicación que no se ajusta al tipo de comunicación	No define ninguna arquitectura
Implementación	Utiliza los elementos solicitados(sockets de flujo, flujos orientados a byte) y desarrolla la aplicación de acuerdo a los requerimientos	Utiliza alguno(s) de los elementos solicitados y desarrolla la aplicación de acuerdo a los requerimientos	Utiliza alguno(s) de los elementos solicitados, pero no desarrolla la aplicación de acuerdo a los requerimientos	No utiliza ninguno de los elementos y no desarrolla la aplicación en base a los requerimientos
Conocimientos	Muestra dominio de los elementos (sockets de flujo, flujos orientados a byte, hilos)	Muestra dominio de los elementos, así como de programación	Muestra dominio de algunos de los elementos	No muestra dominio de los elementos, ni de programación, ni de diseño de interfaces

Presentación	Dominio de los temas, explicación de algoritmos usados utilizados, caso de prueba bien diseñado	Explicación de algoritmos utilizados	Supo darse a entender al explicar algunos de los algoritmos	No supo darse a entender, se confundió
Trabajo colaborativo	Desde el desarrollo del proyecto se denotó el trabajo en grupo, la empatía, buena distribución de tareas, colaboración de todos. Todos saben sobre el proyecto en su totalidad.	Desde el desarrollo del proyecto se denotó el trabajo en grupo, existen algunos roces de opiniones y malentendidos , colaboración de todos. Todos saben sobre el tema en su totalidad.	Desde el desarrollo del proyecto se denotó el trabajo en grupo, la empatía, buena distribución de tareas, colaboración de todos.	Desorganización, falta de interés, el trabajo no se repartió equitativamente, huecos de ignorancia en el desarrollo del proyecto

Introducción

El protocolo de Transferencia de Hipertexto (HTTP, Hyper Text Transfer Protocol) es un protocolo de nivel de aplicación usado para la transferencia de información entre sistemas. Está basado en el modelo Cliente-Servidor y ha sido utilizado por el World Wide Web desde 1990.

Este protocolo permite usar una serie de métodos para indicar la finalidad de la petición. Se basa en estándares como el Identificador de Recurso Uniforme (URI), Localización de Recurso Uniforme (URL) y Nombre de Recurso Uniforme (URN) para indicar el recurso al que hace referencia la petición. El cliente envía una petición en forma de método, una URI y una versión de protocolo, seguida de los modificadores de la petición de forma parecida a un mensaje MIME (Extensiones de Correo Multipropósito de Internet), información sobre el cliente y al final un posible contenido. El servidor contesta con una línea de estado que incluye la versión del protocolo y un código que indica el éxito o error, seguido de la información del servidor en forma de mensaje MIME y un posible contenido.

La sintáxis de la petición es: **http://dirección[: puerto][ruta]** donde dirección es el nombre calificado o la dirección IP del servidor al que se le solicitará un recurso, el puerto es el identificador de la aplicación servidor que nos brindará el recurso y ruta especifica el nombre o ruta hacia el recurso solicitado

Mensaje HTTP: Un mensaje HTTP consiste en una petición de un cliente a un servidor y una respuesta de un servidor al cliente. Las peticiones o respuestas pueden ser simples o completas. La diferencia radica en que una petición completa incluye encabezado y contenido, mientras que una simple solo el encabezado. En el caso de peticiones simples, solo se puede usar el método GET y una respuesta simple solo incluye el contenido.

Petición: En el caso de que la petición se haga con el protocolo HTTP/1.0 ó HTTP/1.1 la petición tiene el sig. formato:

Línea de petición
Encabezado
CRLF
[Contenido]

Donde CRLF es un salto de línea y retorno de carro. La línea de petición tiene el siguiente formato:
Método <espacio en blanco> URI <espacio en blanco> Versión_protocolo CRLF
En el caso de la versión HTTP/1.0 solo soporta los métodos GET, POST y HEAD. Para HTTP/1.1 se soportan los métodos OPTIONS, HEAD, GET, POST, PUT, DELETE, y TRACE.

Respuesta: La respuesta tiene el siguiente formato:

Línea de estado
Encabezado
CRLF
[Contenido]

Donde la línea de estado tiene el siguiente formato:
Versión_protocolo<espacio en blanco>código_de_estado<espacio en blanco>frase_explicativa CRLF
El código es un número de 3 dígitos que indica si la petición ha sido atendida satisfactoriamente o no. La frase explicativa es solo eso, una frase corta que explica el código.

Encabezados: Existen encabezados generales que pueden ser usados tanto para peticiones, como para respuestas, encabezados de respuesta y encabezados e peticiones. A continuación se enlistan:

- **Cache-Control**, son directivas que se han de tener en cuenta a la hora de mantener el contenido en una caché.
- **Connection**, permite especificar opciones requeridas para una conexión.
- **Date**, representa la fecha y la hora a la que se creó el mensaje.
- **Pragma**, usado para incluir directivas de implementación.
- **Transfer-Encoding**, indica la codificación aplicada al contenido.
- **Upgrade**, permite al cliente especificar protocolos que soporta.
- **Via**, usado por pasarelas y proxies para indicar los pasos seguidos.

Encabezados de petición: Permiten al cliente pasar información adicional al servidor sobre la petición y el propio cliente:

- **Accept**, indican el tipo de respuesta que acepta.
- **Accept-Charset**, indica los conjuntos de caracteres que acepta.
- **Accept-Encoding**, que tipo de codificación acepta.
- **Accept-Language**, tipo de lenguaje de la respuesta que se prefiere.
- **Authorization**, el agente de usuario quiere autentificarse con el servidor.
- **From**, contiene la dirección de correo que controla en agente de usuario.
- **Host**, especifica la máquina y el puerto del recurso pedido.
- **If-Modified-Since**, para el GET condicional.
- **If-Match**, para el GET condicional.
- **If-None-Match**, para el GET condicional.
- **If-Range**, para el GET condicional.
- **If-Unmodified-Since**, para el GET condicional.
- **Max-Forwards**, indica el máximo número de elementos por los que pasa.
- **Proxy-Authorization**, permite que el cliente se identifique a un proxy.
- **Range**, establece un rango de bytes del contenido.
- **Referer**, indica la dirección donde obtuvo la URI de la petición.
- **User-Agent**, información sobre el agente que genera la petición.

Encabezados de respuesta: Permiten al servidor enviar información adicional al cliente sobre la respuesta, el propio servidor y el recurso solicitado:

- **Age**, estimación del tiempo transcurrido desde que se creó la respuesta.
- **Location**, se usa par a redirigir la petición a otra URI.
- **Proxy-Authenticate**, ante una respuesta con el código 407 (autentificación proxy requerida), indica el esquema de autentificación.
- **Public**, da la lista de métodos soportados por el servidor.
- **Retry-After**, ante un servicio no disponible da una fecha para volver a intentarlo.
- **Server**, información sobre el servidor que maneja las peticiones.
- **Vary**, indica que hay varias respuestas y el servidor ha escogido una.
- **Warning**, usada para aportar información adicional sobre el estado de la respuesta.
- **WWW-Authenticate**, indica el esquema de autentificación y los parámetros aplicables a la URI.

Encabezados de entidad: Aportan información sobre el contenido del mensaje o si no hay contenido, sobre el recurso al que hace referencia el URI de la petición:

- **Allow**, da los métodos soportados por el recurso designado por la URI.
- **Content-Base**, indica la URI base para resolver las URI relativas.
- **Content-Encoding**, indica una codificación adicional aplicada al contenido (a parte de la aplicada por el tipo).
- **Content-Language**, describe el idioma del contenido.

- **Content-Length**, indica el tamaño del contenido del mensaje.
- **Content-Location**, da información sobre la localización del recurso que da el contenido del mensaje.

Recursos y/o materiales	
• Manual de prácticas de laboratorio de Aplicaciones para Comunicaciones en Red • Plumones • Bibliografía	• Internet • Computadora • IDE de desarrollo • Apuntes

Instrucciones

En esta práctica debes implementar un servidor HTTP que sea capaz de reconocer y servir peticiones de recursos mediante los métodos GET, POST, HEAD, PUT Y DELETE (Ver RFC 2616)

Desarrollo de la práctica

A partir del programa ServidorWeb que te será proporcionado por el profesor deberás realizar las siguientes modificaciones:

- El programa ServidorWeb implementa un servidor HTTP con funcionalidad limitada en donde solo es capaz de atender peticiones de tipo GET. Basándote en la especificación del protocolo HTTP deberás implementar los métodos de respuesta para peticiones de tipo POST, HEAD, PUT y DELETE. El servidor también deberá ser capaz de atender peticiones por lo menos 3 tipos MIME distintos.
- También deberás enviar códigos de respuesta basándote en la especificación de HTTP (ej. 200 para una operación exitosa, 500 para un error interno del servidor).
- Una vez terminado el servidor, deberás probarlo con páginas o aplicaciones web que hagan uso de dichos métodos de envío.
- Implementa una alberca de hilos para que el servidor HTTP pueda atender una cantidad máxima de 100 clientes cocurrentemente.

Cierre de la práctica

Preguntas:

1. ¿Qué ventajas tiene el uso del método POST vs GET o HEAD?
2. ¿Qué modificaciones harías al servidor para que fuese capaz de interpretar scriptlets JSP?