Redes de Comunicaciones II

MEMORIA PRÁCTICA 1: SERVIDOR HTTP

Fecha de entrega: 15 de Marzo de 2023
Grupo 2311

Equipo 01
Ismael Iván López Murillo
Óscar Navalón Navarro

1. INTRODUCCIÓN

En esta práctica se ha desarrollado un servidor HTTP básico que permite servir archivos estáticos a través de la web. El funcionamiento de este servidor se basa en el protocolo HTTP, que es el estándar utilizado para la transferencia de información en la web.

El servidor HTTP se encarga de escuchar las peticiones realizadas por los clientes (por ejemplo, un navegador web), procesarlas y responder con la información solicitada. Para ello, el servidor utiliza una serie de algoritmos y técnicas que se han implementado en el código, como la gestión de sockets, la interpretación de cabeceras HTTP para el manejo de las respuestas y la configuración del servidor a través de ficheros auxiliares

Uno de los objetivos principales de esta práctica ha sido aprender a utilizar técnicas para desarrollar un proyecto de cierta complejidad, como es un servidor web. Se ha utilizado una metodología de desarrollo escalable implementando pieza a pieza cada parte del servidor por separado para así ir reduciendo en abstracción hasta lo más tangible.Todo esto se desarrollará con más ímpetu en los siguientes apartados de la memoria.

2. DOCUMENTACIÓN DE DISEÑO

a. Especificación de requisitos funcionales

i. El servidor será capaz de recibir y procesar peticiones HTTP.
ii. El servidor será capaz de enviar respuestas HTTP a las peticiones correspondientes.
iii. El servidor será capaz de procesar peticiones del verbo GET para obtención de recursos (imágenes, textos, archivos html, multimedia)
iv. El servidor será capaz de procesar peticiones del verbo POST para la ejecución de scripts
v. El servidor será capaz de procesar scripts tanto del lenguaje Python y PHP, ejecutando el intérprete correspondiente y devolviendo su salida en texto plano hasta “\r\n”
vi. Las variables de entrada para los scripts que el servidor ejecutará serán el Body, en caso de POST y los query parameters en caso del verbo GET, los cuales serán debidamente procesador dentro del contenido del script.
vii. El servidor será capaz de procesar el verbo OPTIONS tanto para recursos en específico como para el servidor raíz, contestando en su respuesta en la cabecera “Allow” los verbos soportados según sea el caso.
viii. El servidor será capaz de manejar múltiples conexiones simultáneas, atendiendo múltiples clientes a la vez.
ix. En caso de que se haga una petición HTTP a un recurso no existente, el servidor retornará una respuesta 404 Not Found.

x. En caso de que se haga una petición HTTP con un verbo no soportado (DELETE, PUT, OPTIONS) el servidor retornará una respuesta 405 Method Not Allowed
xi. El servidor será capaz de mantener abierta una misma conexión establecida con un cliente
xii. Las respuestas del servidor incluirán las cabeceras correspondientes de forma correcta, como lo son Content-Type, Date, Server, Content-Length, y Connection.
xiii. El servidor no permitirá el listado de directorios mediante el verbo GET, contestando con una respuesta 403 Forbidden.
xiv. El puerto en que se ejecuta el servidor, el nombre del servidor, los máximos clientes esperando en la cola de peticiones y el directorio raíz de los recursos del servidor deberán ser debidamente especificados en el archivo de configuración siguiente el formato "clave = valor"
xv. En caso de que el archivo de configuración no sea encontrado o que las claves no sean especificadas, se usarán valores por defecto,

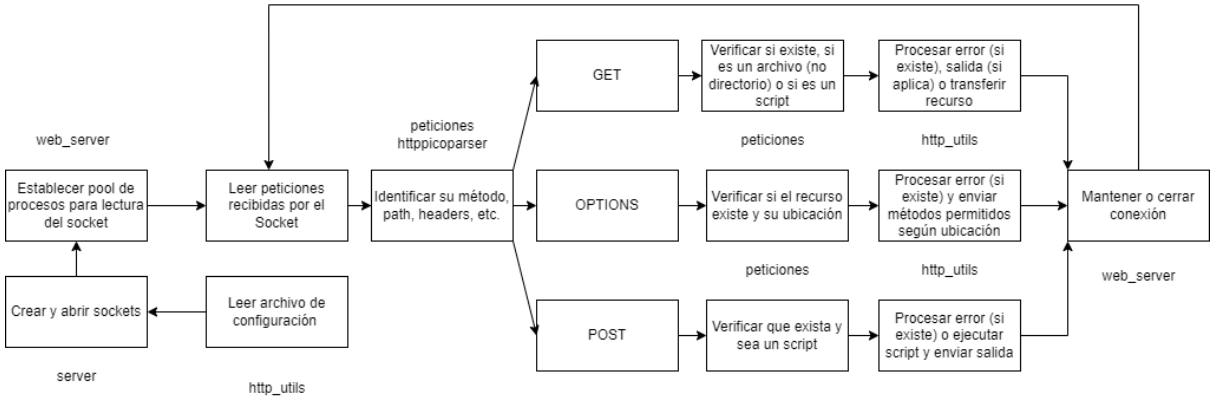
b. Especificación de requisitos no funcionales

i. Se requiere de una versión mínima de C99 para compilar y ejecutar el proyecto
ii. El servidor HTTP debe ejecutarse en una máquina con sistema operativo Linux
iii. A pesar de que no es completamente necesario, se requiere de estar conectado a internet para que el servidor pueda ser accedido remotamente dentro de la misma red.
iv. Se requiere de un espacio en disco mínimo de 200MB para guardar el código fuente, ejecutables y los recursos del servidor

c. Flujo lógico: El servidor HTTP en su forma más general sigue este procesamiento:

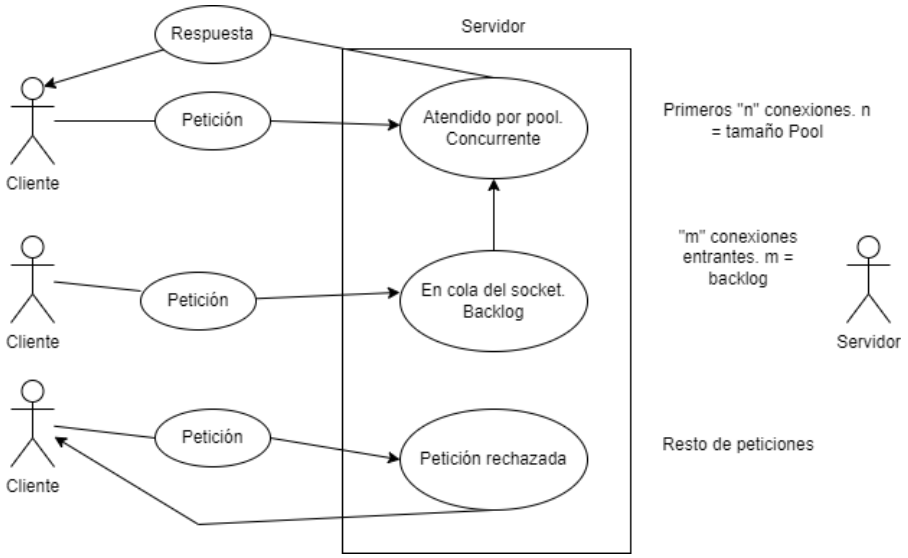
i. Crear sockets que permitan la conexión de clientes al servidor
ii. Crear una pool de procesos para manejar las conexiones entrantes al servidor
iii. Para cada conexión entrante, se leen las peticiones entrantes hasta que no exista más contenido que leer o el cliente cierre la conexión
iv. Para cada petición, se leen los headers, verbo y path (usando la librería httppicoparser)
v. Se verifica que exista el recurso o URL (si no, se procesa una respuesta 404) y que el método sea válido (si no, se procesa una respueta 405)
vi. Dependiendo el método, se verifica si el recurso es una ruta o un recurso, solo la petición OPTIONS permite acceso a rutas
vii. Para la petición GET, se transmite el binario del archivo en cuestión o se ejecuta el script en cuestión con los query params como variables
viii. Para la petición POST, se ejecuta el script en cuestión con el body como variables
ix. Para la petición OPTIONS, se envían los métodos autorizados para el path o recurso. Solamente el servidor en general, los recursos scripts o los scripts permiten POST

x. Se envía la petición, enviando los headers correspondientes y se repite el paso 3

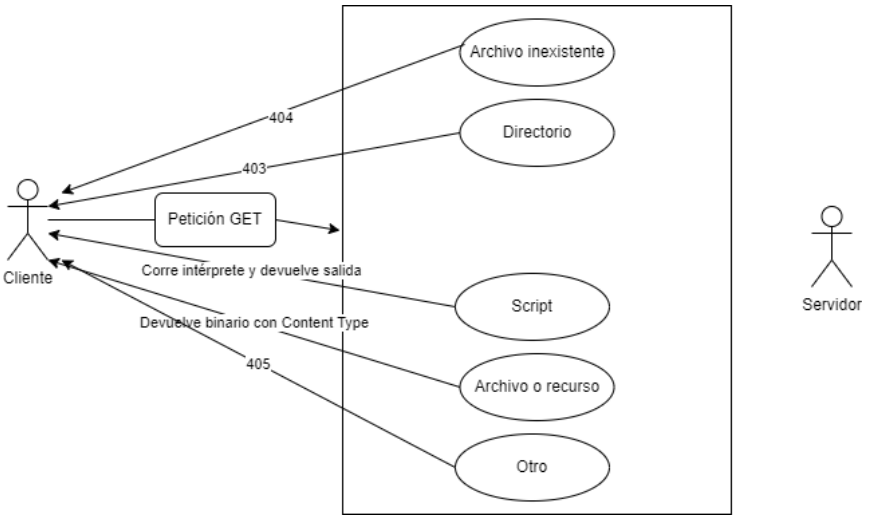


d. Casos de uso

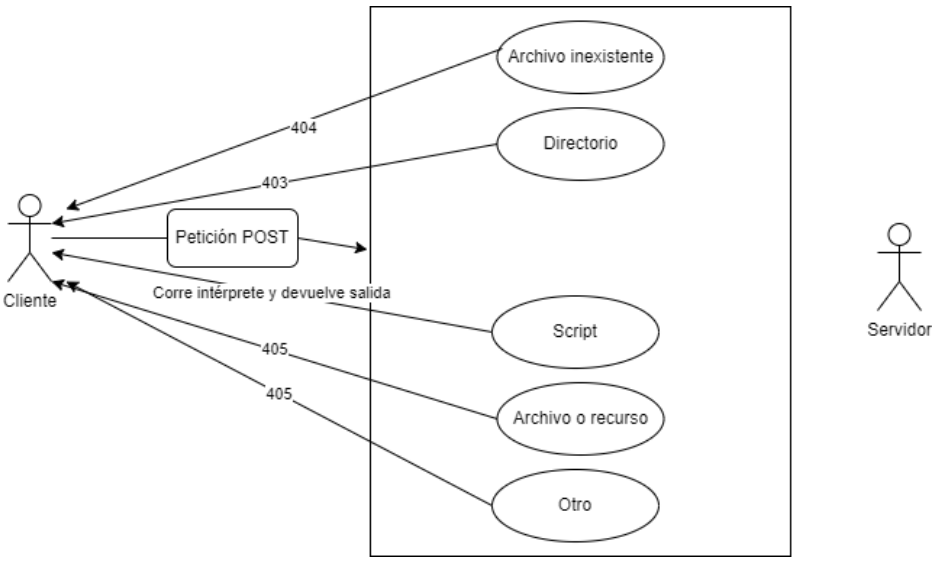
i. Peticiones de múltiples usuarios



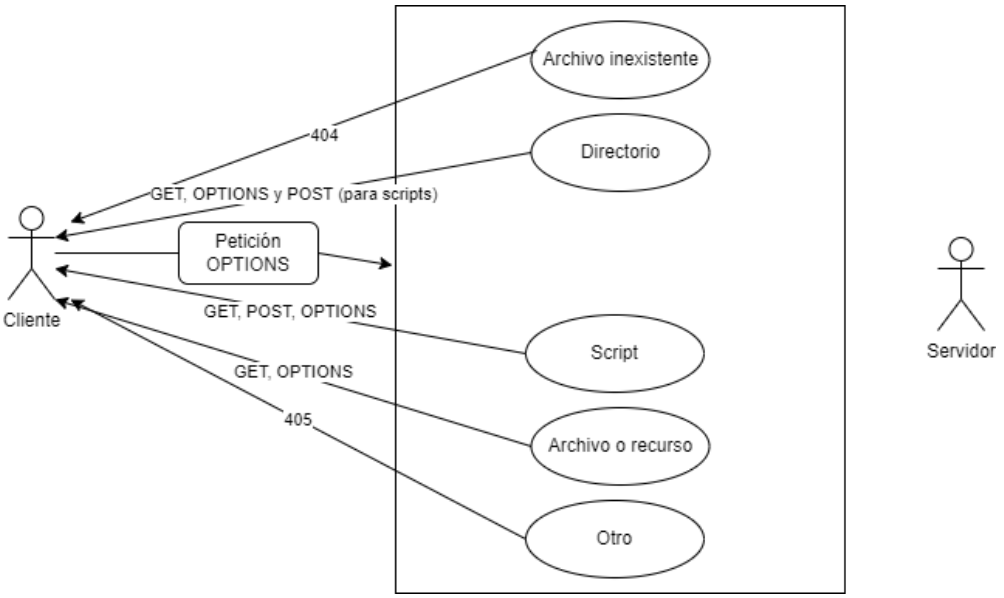
ii. Petición GET



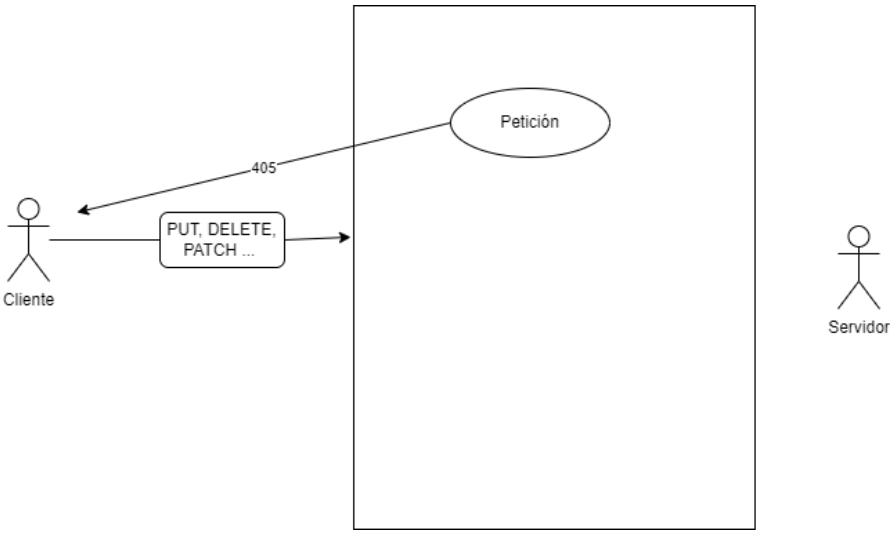
iii. Petición POST



iv. Petición OPTIONS



v. Otras peticiones



3. DESARROLLO TÉCNICO

Durante el desarrollo de este servidor HTTP, se siguieron diversas etapas para lograr su correcto funcionamiento. En primer lugar, se desarrolló la gestión de sockets para un único cliente, implementando un servidor iterativo, esto para comprobar que hemos podido establecer una conexión de forma correcta con otros equipos.

Posteriormente, se implementó tanto el servidor reactivo como uno con pool de procesos, estos serán de utilidad en el futuro para que nuestro servidor sea más eficiente. Sin embargo, fueron comentados para simplificar el desarrollo del servidor HTTP con el servidor iterativo, en donde tendremos un procesamiento secuencial que permitirá hacer una depuración más sencilla.

Se comenzó con el manejo del navegador a través de una response pre-montada que mostraba un "Hello World" en el navegador, esto para comprobar la idea de cómo proseguiremos (manejo de respuestas, headers, etc,).

Luego, con la ayuda de la librería picohttpparser, la cual es una librería muy ligera y sencilla para procesar los headers HTTP, se desarrolló la estructura principal para el procesamiento de peticiones, comprobando la prueba de concepto de lectura de Headers, de body, del path y verbo en cuestión, y recepción correcta de peticiones HTTP en general.

A continuación, se implementó el verbo GET únicamente para las solicitudes HTML, es decir, solo un archivo de texto, esto para tener un avance más visual. En paralelo, se desarrolló el archivo server.conf y se implementó su lectura y procesamiento y asignamiento de datos a las configuraciones correctas del servidor, ya que estas funcionalidades resultan independientes en su desarrollo, uno puede continuar con las peticiones sin tener el archivo de configuración, y el archivo de configuración podría ser desarrollado con la respuesta pre-montada.

Se procedió a la implementación del verbo GET para cualquier tipo de recurso para continuar con la visualización de resultados y para proceder con el resto de verbos (POST y OPTIONS) y al control de errores en el servidor, especialmente el error 404 y 403, que son importantes previos al manejo de más solicitudes. .

Parte del desarrollo del método POST fue implementar la ejecución de scripts .py y .php, los cuales se hicieron primeramente de forma aislada, probando que se podía ejecutar un script desde C, esto para simplificar la depuración, para después ser procesado en el servidor, tomando el algoritmo que se comprobó que servía y agregándole a la petición de scripts..

Finalmente, se llevó a cabo la última etapa de limpieza de código y ultimación del mismo, en donde se identificaron áreas de mejora dentro del servidor y posibles errores que fueron solucionados, como la opción de listar directorios o el reuso de direcciones de un socket.

Cada una de estas etapas fue esencial para el correcto funcionamiento del servidor HTTP y para lograr cumplir con los objetivos propuestos, por lo cual se tuvo en

general un enfoque secuencial paralelizado en algunos procesos (como lectura de configuración).

4. CONCLUSIONES

a. Conclusiones técnicas

En este proyecto se ha logrado implementar un servidor HTTP básico capaz de manejar solicitudes HTTP/1.1 y servir archivos a los clientes.
Se han respetado todos los requisitos funcionales para el desarrollo.

Se ha logrado un manejo adecuado de las excepciones y errores, lo que aumenta la robustez del servidor y su capacidad para seguir funcionando incluso en situaciones imprevistas.

En conclusión, este proyecto ha permitido profundizar en el funcionamiento de los servidores HTTP y en el manejo de las solicitudes y respuestas, lo que ha resultado en una implementación básica pero funcional de un servidor HTTP.

Es posible seguir extendiendo este mismo proyecto para soporte de otros verbos, manejo de autenticación, entre otras funcionalidades de HTTP, ya que se cuenta con una base sólida.

b. Conclusiones personales

En cuanto a las conclusiones personales, podemos decir que hemos aprendido mucho trabajando juntos en este proyecto. Uno de los mayores desafíos fue asegurarnos de que estuviésemos alineados en cuanto a los objetivos y la forma en que debíamos abordar el trabajo.

Además, pudimos experimentar con diferentes enfoques y técnicas de programación, lo que nos permitió mejorar nuestras habilidades y aprender cosas nuevas. Nos sentimos satisfechos de haber logrado construir un servidor HTTP completamente funcional que cumple con los requisitos establecidos.

Por otro lado, también nos enfrentamos a algunos problemas técnicos y tuvimos que trabajar duro para superarlos. Aunque esto a veces fue frustrante, sentimos que aprendimos mucho al enfrentar estos desafíos y resolver los problemas, sobre todo al manejar un lenguaje como lo es C en donde no tenemos tantas simplicidades y toca trabajar más duro.

Como equipo tuvimos una comunicación asertiva y pudimos aprender mucho del otro, siempre manteniendo una carga de trabajo similar y mejorando el proyecto continuamente entre pares.