

Documentación práctica 1 Telemática

Telematics Web Server (TWS)

Katherine Benjumea Ortiz

Brigith Lorena Giraldo Vargas

Valentina Ochoa Arboleda

Telemática

Universidad Eafit

Escuela de ciencia y tecnología

Ingeniería de sistemas

Medellín

Abril 2023



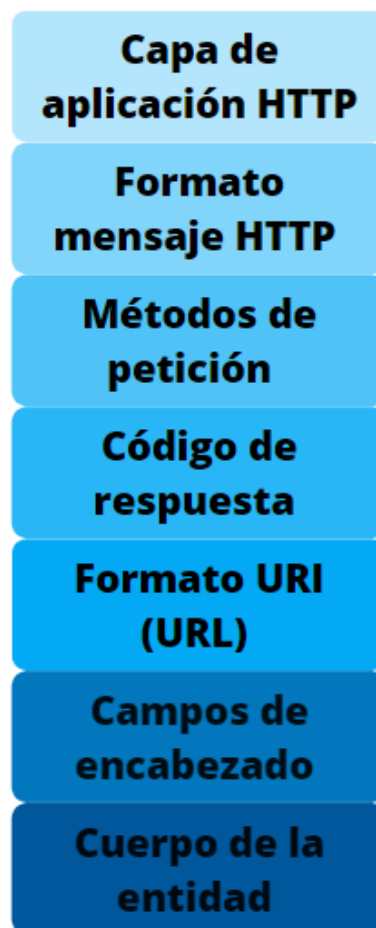
## Introducción

Este es un proyecto de aula enfocado en la aplicación de la arquitectura TCP/IP. El cual consiste en implementar un algoritmo que permita realizar un servidor web en lenguaje C, donde se procesan peticiones del protocolo HTTP/1.1 desarrolladas en la capa de aplicación de la arquitectura. El servidor retorna los recursos solicitados por el cliente.

Este proyecto nos permite analizar tres tipos de peticiones dentro del protocolo: GET, HEAD, POST, para un servidor. Este debe de recibir estas peticiones por múltiples usuarios y ser capaz de retornar lo solicitado. En caso de que las peticiones no se hagan de manera adecuada, este debe enviar los errores que se puedan presentar. Por último, el servidor posee un logger para visualizar las peticiones HTTP.

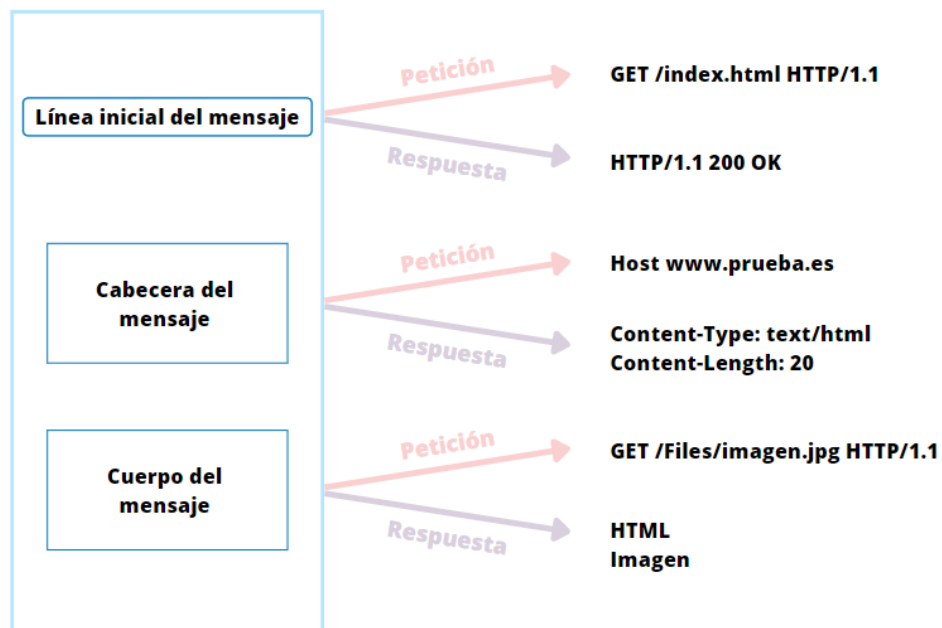
Este servidor nos ayuda a identificar el protocolo que se ejecutan en la capa de aplicación desde la cual se suministra los servicios de red en el modelo OSI.

La arquitectura que se presenta en esta capa de aplicación está compuesta por los siguientes elementos:



*Imagen 1: Arquitectura TCP/IP*

Los cuales nos ayudan al envío de las peticiones que se le hacen al servidor y a las respuestas que este envía al cliente como se evidencia en el siguiente diagrama.



*Imagen 2: Modelo Cliente-Servidor*

Se necesitan recursos específicos para la adecuada implementación de este proyecto que se mencionarán en la sección de desarrollo.

## Objetivos

1. Desarrollar un servidor HTTP funcional en lenguaje C, utilizando sockets para establecer la conexión con los clientes y la Arquitectura TCP para la transferencia de datos.
2. Comprender y aplicar correctamente las especificaciones del protocolo HTTP para implementar los métodos HTTP GET, POST y HEAD en el servidor, implicando la capacidad de procesar y manejar solicitudes GET, POST y HEAD de los clientes, y responder adecuadamente a estas solicitudes.
3. Implementar el manejo de errores HTTP 400 (Bad Request) y 404 (Not Found) en el servidor, haciendo que tenga la capacidad de detectar y responder adecuadamente a solicitudes incorrectas o solicitudes para recursos no encontrados en el servidor.
4. Habilitar funcionalidad multihilo en el servidor para manejar varias solicitudes concurrentemente y así mejorar la escalabilidad del servidor.

## Desarrollo

En esta sección se explica la implementación del servidor con detalles para su entendimiento por parte de los usuarios.

Para efectos de pruebas se despliega el servidor implementado en la infraestructura de AWS (Amazon Web Services). Se procede a lanzar una instancia EC2 para realizar la conexión con el servidor.

En caso de tener sistema operativo de Windows se debe hacer la instalación de PuTTY, en el que se procede a agregar la dirección IP pública extraída de la instancia lanzada en AWS y el par de llaves privadas para la autenticación de la sesión, este archivo de llaves debe ser generado en el equipo en que se va a trabajar para correr el servidor por consola. Al iniciar la sesión en putty se logea con el comando Ubuntu

Se procede a descargar en consola apache2 que nos servirá para la ejecución del servidor y gcc para la compilación del lenguaje C, esto solo se realiza la primera vez que se lanza la instancia en AWS.

Se conecta un telnet al puerto 8080 con la dirección IPv4 pública que proporciona AWS para simular un browser en el que se generan las peticiones de los clientes.

Todo lo anterior se necesita para la correcta ejecución del servidor.

Para la implementación del servidor en el lenguaje C, usamos la api de Berkeley sockets para la creación del socket del servidor, siendo este el punto final de la conexión, por lo cual se ejecuta el envío de información. El socket nos permite implementar una arquitectura basada en el modelo cliente-servidor para la realización de las peticiones.

El socket necesita unos parámetros en específico los cuales son la dirección IPv4 privada y el Address del cliente que es la dirección pública, las cuales son proporcionadas por AWS.

Ya teniendo la conexión, el servidor debe comenzar a analizar las peticiones que se reciben por parte del cliente para esto se parsea la petición y de acuerdo al método que sea solicitado tendrá una respuesta en específico.

Estas peticiones contienen unos parámetros en específicos los cuales son: el método, la URL, la versión y los headers siendo estos necesarios para identificar cual recurso es el que el cliente está solicitando.

Dado el caso que el servidor no posea con los recursos que son solicitados por el cliente, esta debe de retornar un error de acuerdo al caso que se presente. En este servidor se implementó el error 404 el cual se retorna cuando el archivo solicitado no se encuentra disponible y el error 400 cuando la solicitud que ha hecho el cliente es incorrecta.

A continuación, se evidencia la manera en que el servidor recibe las peticiones y como envía las respuestas.

Para el método GET:

```

Nuevo cliente conectado: 181.133.129.27:62095
Método: GET
Url: /Files/perros.jpg
Versión: HTTP/1.1
Headers:
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Postman-Token: fecc197a-9279-45d6-a599-f73563cc8336
Host: 44.203.169.78:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
El nombre del archivo final es: perros.jpg
Contenido del archivo enviado.

```

*Imagen 3: Consola Servidor.*

44.203.169.78:8080/Files/perros.jpg Save

GET 44.203.169.78:8080/Files/perros.jpg

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results 200 OK 276 ms 4.13 KB Save

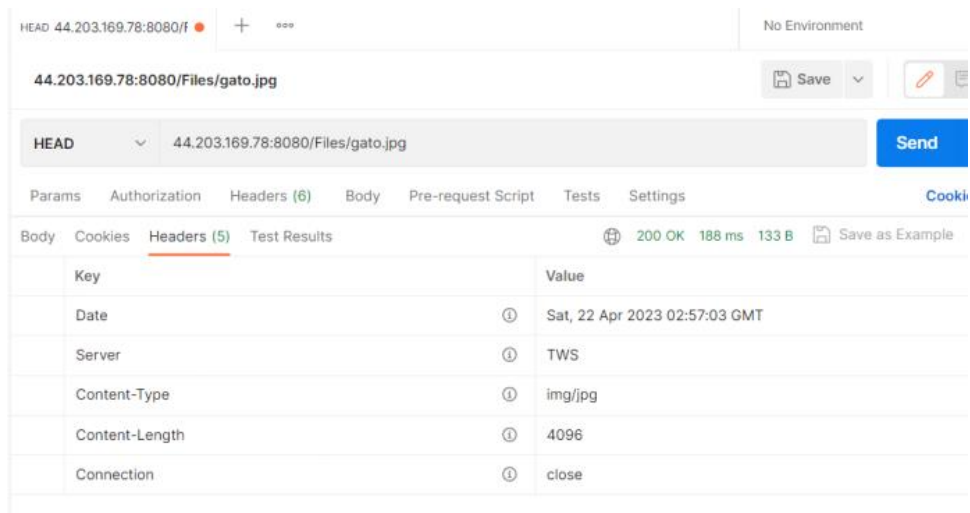
Key	Value
Date	Sat, 22 Apr 2023 02:57:03 GMT
Server	TWS
Content-Type	img/jpg
Content-Length	4096
Connection	close

*Imagen 4: Prueba en Postman.*

Para el método HEAD:

```
Nuevo cliente conectado: 181.133.129.27:62123
Método: HEAD
Url: /Files/gato.jpg
Versión: HTTP/1.1
Headers:
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Postman-Token: 4b81c4bd-ce80-47c5-8913-72491d6970ed
Host: 44.203.169.78:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
El nombre del archivo final es: gato.jpg
Confirmado.
```

*Imagen 5: Consola Servidor.*



Key	Value
Date	Sat, 22 Apr 2023 02:57:03 GMT
Server	TWS
Content-Type	img/jpg
Content-Length	4096
Connection	close

*Imagen 6: Prueba en Postman.*

Para el método POST:



```
[+]TCP server socket creado.
[+]Conectar al número de puerto: 8080
Servidor iniciado. Esperando conexiones...
Nuevo cliente conectado: 181.133.129.27:61932
Método: POST
Url: /Files/example.txt
Versión: HTTP/1.1
Headers:
Content-Type: text/plain
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Postman-Token: c3f33a64-f6f8-4c15-abfa-f3a1f7b78907
Host: 44.203.169.78:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 23
El nombre del archivo final es: example.txt
Recibido del cliente:
Hola, esta es la prueba
El body se ha guardado en example.txt
█
```

Imagen 7: Consola Servidor.

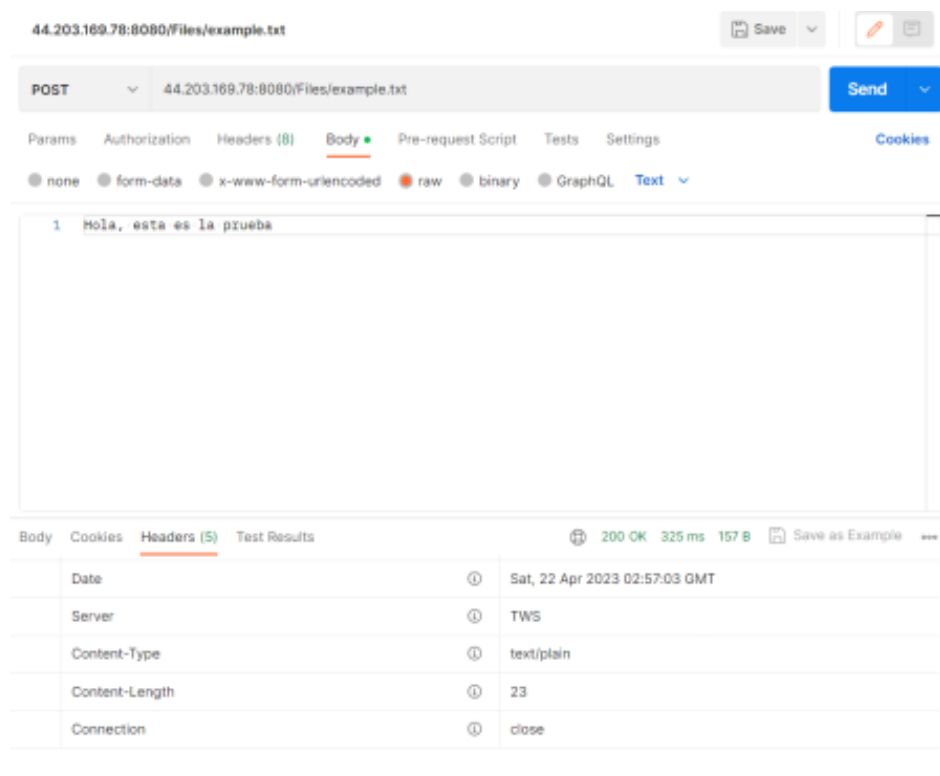


Imagen : Prueba en Postman.

Conclusiones:

1. En la implementación de sockets se requiere tener claro como es el funcionamiento de la capa de aplicación en la arquitectura TCP en el modelo OSI para ejecutar de manera correcta las peticiones del protocolo HTTP teniendo en cuenta las especificaciones de este.
2. Al implementar un tipo de socket Stream para el servidor se obtiene un flujo de datos bidireccional, el cual es orientado a la conexión.
3. Para futuros proyectos se podría implementar la optimización de las peticiones con opciones como, la compresión de datos y la optimización de imágenes.
4. El equipo de trabajo obtuvo mucho aprendizaje en el manejo de C para la creación del servidor, se logró comprender más el manejo correcto de peticiones HTTP para la comunicación en la web.
5. No se logró cumplir con todos los objetivos, ya que se tenían carencias en el conocimiento respecto al lenguaje implementado, aun así, se hizo la correcta indagación de recursos disponibles en la web que permitieran el desarrollo del servidor y su estructura, con el uso de sockets.

## Referencias

Team, K. (2022, September 6). ¿Qué es un Socket? | KeepCoding Bootcamps. *KeepCoding Bootcamps*. <https://keepcoding.io/blog/que-es-un-socket/>

*Generalidades del protocolo HTTP - HTTP / MDN*. (2022, November 26). <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>

*RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*. (1999, 1 junio). IETF Datatracker. <https://datatracker.ietf.org/doc/rfc2616/>

*Postman*. (s. f.). <https://www.postman.com/>

*Programación en C/Sockets - Wikilibros*. (s. f.). [https://es.wikibooks.org/wiki/Programaci%C3%B3n\\_en\\_C/Sockets](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Sockets)

De Expertos En Ciencia Y Tecnología, E. (2022, 16 noviembre). Explicando la arquitectura de protocolos TCP/IP. VIU. <https://www.universidadviu.com/co/actualidad/nuestros-expertos/explicando-la-arquitectura-de-protocolos-tcpip>