ST0255 TELEMÁTICA
PROYECTO N1
Programación en Red

Fecha de publicación: marzo 9 de 2024
Última actualización: marzo 9 de 2024.
Fecha de Entrega: abril 9 de 2024.

1. Introducción

El objetivo de este proyecto es desarrollar las competencias necesarias en el diseño y desarrollo de aplicaciones concurrentes en red. Para lograr esto, se empleará la API Sockets con el fin de escribir un protocolo de comunicaciones que permita a una aplicación cliente comunicarse con una aplicación servidor.

Para el caso particular, se requiere implementar un proxy inverso para HTTP y balanceador de carga. Estos son elementos intermedios en el funcionamiento y operación de aplicaciones cliente/servidor.

En este sentido, un proxy inverso se entiende como aquel que recibe (intercepta) cada una de las peticiones del cliente y la envía a un servidor con la capacidad de procesar la petición para finalmente enviar la respuesta al cliente. Por otro lado, un balanceador de carga, es la entidad encargada de distribuir las peticiones entrantes por parte de los clientes hacia un conjunto de servidores. Para cada petición, debe posteriormente, debe retornar la respuesta al cliente.

2. Antecedentes

2.1. Sockets

Tradicionalmente, a los desarrolladores de aplicaciones, las APIs de desarrollo de los diferentes lenguajes de programación, les realizan abstracciones que les ocultan los detalles de implementación de muchos aspectos, y el proceso de transmisión de datos no es la excepción.

Un socket es una abstracción a través de la cual las aplicaciones pueden enviar y recibir datos. Al respecto, el socket se puede entender como el mecanismo que utilizan las aplicaciones para “conectarse” a la red, específicamente con la arquitectura de red o “stack” de protocolos y de esta forma comunicarse con otras aplicaciones que también se encuentran “conectadas” a la red. De esta forma, la aplicación que se ejecuta en una máquina escribe los datos que dese transmitir en el socket y estos datos los puede leer otra aplicación que se ejecuta en otra máquina.

2.2. Tipos de Sockets

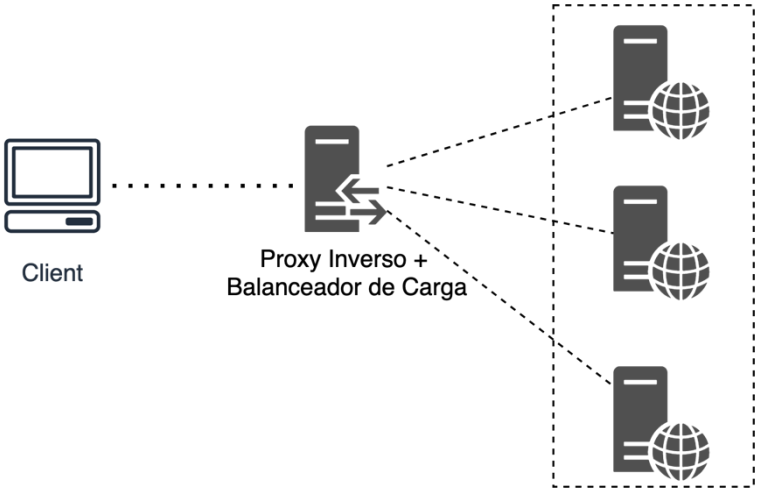
Existen diferentes tipos de sockets. Para este proyecto vamos a utilizar la API Sockets Berkeley, específicamente los sockets tipo “Stream (SOCK_STREAM)” o “Datagram (SOCK_DGRAM)”. En el marco de este proyecto, acorde al protocolo que está implementando usted debe decidir qué tipo de socket va a emplear y justificar a la luz de los requerimientos de su aplicación.

2.3. HTTP Proxy + Balanceador de Carga

En la figura 1 se puede observar la arquitectura a implementar en la cual se observa la aplicación cliente, así como la aplicación servidor. La solución por implementar debe contar con los siguientes componentes:

- La aplicación cliente.
- El servidor HTTP Proxy + Balanceador de Carga.
- Tres servidores de aplicación web. Se debe desplegar una página web la cual se debe replicar en los tres servidores. Para efectos de esta práctica, debe implementar la misma página web (html+css+js) en los tres servidores. A nivel de servidor web, se puede emplear tanto NGINX como Apache.

Como se puede observar, se requiere que usted disponga de una máquina para desplegar su aplicación HTTP Proxy + Balanceador de Carga y una para cada servidor web (NGINX/Apache).



HTTP esta concebido para ser un protocolo que responde al patrón cliente/servidor. De esta forma, la aplicación cliente se comunica directamente con la aplicación servidor. Sin embargo, en muchos escenarios, es útil la implementación de un elemento intermedio denominado Proxy. En este caso, el cliente envía la petición al proxy, este la intercepta y la envía al servidor web que aloja el recurso. Observe que en este caso, el proxy se va a comportar como el cliente para los servidores web. De esta forma, el proxy recibirá la respuesta por parte del servidor y la enviará al cliente. Dentro del conjuto de razones por las cuales se emplea un proyx a nivel de http, se resaltan aspectos de desempeño, filtrado y transformación de contenido, entre otros aspectos.

3. Requerimientos.

Se requiere que usted implemente tanto una aplicación cliente como servidor. De esta forma el cliente, HTTP Proxy + Balanceador de Carga + Servidor Web comunicarán a través de los mensajes definidos para el protocolo HTTP/1.1.

3.1. Cliente HTTP

Se requiere que usted implemente una aplicación que permita realizar peticiones HTTP a cualquier servidor HTTP, así como con su servidor HTTP Proxy + Balanceador de Carga y visualizar/almacenar la respuesta proveniente del servidor. Es así como se requiere armar la petición HTTP sin el uso de ningún tipo de librería.

La aplicación cliente se debe ejecutar de la siguiente forma:

./httpclient </path/log.log> <url:port>

- httpclient: es el nombre del archivo ejecutable.
- /path/log.log: representa la ruta y nombre del archivo que almacena el log.
- < url:port >: URL y puerto donde se localiza el recurso a solicitar.

Su aplicación cliente debe permitir registrar todas las peticiones que se generen. Para tal efecto, el archivo de log debe permitir el registro de la siguiente información:

<date> <time> <http_request> <http_response>

- date: indica la fecha en que se efectuó la transacción.
- time: indica la hora en que se realizó la transacción.
- http_request: indica el tipo de solicitud HTTP que se está realizando.
- http_response: la respuesta HTTP que se recibe del servidor/proxy.

Es importante resaltar que la aplicación cliente, puede consultar cualquier recurso web vía HTTP. Esto quiere decir, por ejemplo, que se puede solicitar un recurso html; éste se debe descargar y almacenar en el filesystem del host que ejecuta la aplicación y, para el caso particular de este tipo de recursos, se debe permitir la visualización de este (texto). Para el caso de cualquier otro recurso como imágenes (png, jpeg), documentos (pdf,docx,ppt,xls), videos (mpeg, mov), etc, solo se requiere el almacenamiento de este.

Las peticiones HTTP se deben transportar sobre protocolo de transporte TCP y emplearán como se mencionó anteriormente la versión HTTP/1.1. Su aplicación cliente debe ser capaz de realizar peticiones empleando los métodos GET, HEAD y POST.

Finalmente, el cache es una funcionalidad muy útil para este tipo de aplicaciones, por tal razón se le solicita que implemente dicha funcionalidad. Para esto, se deben almacenar las últimas consultas realizadas. Esto implica, que cuando se quiera realizar una consulta que ya se ha efectuado previamente, se debe verificar si está ya no se encuentra en cache. Si es así, solo se debe consultar si se ha realizado alguna modificación del recurso (método HEAD).

Por favor, implemente el comando flush en su aplicación cliente, el cual debe permitir borrar totalmente el cache de la aplicación.

3.2. Servidor HTTP Proxy + Balanceador de Carga

Para cumplir los requisitos de funcionamiento de esta subsección, se requiere que usted implemente un servidor HTTP Proxy + Balanceador de Carga que permita interceptar una petición HTTP/1.1 y enviarla a un conjunto de servidores web que opera a nivel de back end.

Finalmente, se debe retornar la respuesta al cliente vía el servidor proxy. Para lograr esto se requiere que usted considere los siguientes aspectos:

- Su servidor HTTP Proxy + Balanceador de carga debe ser escrito en lenguaje de programación C.
- No se debe emplear ningún tipo de librería para el procesamiento de las solicitudes HTTP.
- Se debe emplear la API Sockets.
- Se requiere que se procesen peticiones para la versión de protocolo HTTP/1.1.
- Se requiere que su servidor escuche peticiones en el puerto 8080. Una vez reciba la petición de un cliente (p.ej., browser, terminal de consola, postman, su aplicación cliente implementada en el literal anterior), se debe iniciar un nuevo socket cliente para comunicarse con el servidor web destino elegido, y enviar la petición a éste.
- Una vez envié la petición al servidor, debe esperar la respuesta y enviarla al cliente web que la solicito que solicite el recurso web.
- Se requiere que la aplicación HTTP Proxy + Balanceador de Carga implemente un archivo de "log" donde se registren todas las peticiones que recibe. En este sentido, el log debe permitir registrar todas las peticiones que se reciben y debe visualizar la petición que se hace y la respuesta que se entrega. Esto se debe visualizar por la salida estándar, y de igual forma, se debe implementar el registro en un archivo.
- Su HTTP Proxy + Balanceador de Carga es responsable de implementar los siguientes métodos:
    - GET y HEAD.
- Una vez el proxy reciba la petición del cliente y realice el "parsing", puede enviar la petición a uno de los servidores destino.
- Su servidor HTTP Proxy debe modificar la petición que proviene del cliente:
    - El proxy siempre debe enviar la petición al servidor origen empleando la url relativa y empleado el encabezado Host. Por ejemplo, si la petición se está realizando del cliente al HTTP Proxy de la siguiente forma:

      GET www.mipagina.com HTTP/1.1

      El proxy debe enviar la petición al servidor origen de la siguiente forma:

      GET / HTTP/1.1
      Host: www.mipagina.com
      Connection: close
      (Encabezados adicionales si aplica)

    - Agregar el encabezado Via. El encabezado HTTP Via lo agregan los servidores proxy y puede estar presente tanto en solicitudes como en respuestas HTTP.

- La función de proxy debe permitir el caché para los diferentes recursos que se soliciten por parte de los clientes. Para esto debe considerar lo siguiente:
    - Para todos los recursos solicitados en las peticiones hecha por los clientes, la respuesta (el recurso solicitado) debe ser almacenada en un archivo en el disco del servidor. De esta forma se garantiza que el cache persista en caso tal se presente una falla en el servidor HTTP Proxy + Balanceador de Carga. Así, la próxima vez que se realice la petición de este recurso, se debe acceder desde al disco y enviar la respuesta desde aquí hacia el cliente.
    - Los recursos para almacenar en el cache deben ser localizados en el directorio donde se ejecuta la aplicación principal del HTTP Proxy + Balanceador de Carga.

    - Se debe implementar un mecanismo para implementar un TTL para cada recurso que se mantenga en el cache. Esto debe ser un parámetro que se pase al momento de lanzar la aplicación.
- Para cualquier error de funcionamiento que se presente entre el cliente y el proxy, este debe enviar un mensaje con código de estado (status code) 500 Internal Server Error. Por ejemplo, si un cliente realiza una petición con un método no implementado, este debe ser el mensaje de error que se debe entregar. Lo mismo aplica para cualquier petición que no esté armada de la forma correcta.
    - En cualquier otro caso, el HTTP Proxy simplemente debe hacer el reenvío de la respuesta que envía el servidor origen al cliente.
- Para efectos de distribución de la carga de las peticiones, la política a implementar es Round Robin.

Para efectos de ejecutar su servidor HTTP Proxy + Balanceador de Carga, lo debe realizar de la siguiente manera:

./httproxy <port> </path/log.log>

- ./ httproxy: es el ejecutable de la aplicación.
- port: es el puerto en el cual se escucharán las peticiones por parte de los clientes. Para efectos de este proyecto debe ser el puerto 8080.
- /path/log.log: representa la ruta y nombre del archivo que almacena el log.

Finalmente, tenga en cuenta que las aplicaciones en red son concurrentes. En este sentido, el servidor HTTP Proxy + Balanceador de Carga que se está implementando no es la excepción. De esta forma, se requiere que su servidor se capaz de soportar de manera concurrente múltiples peticiones HTTP de manera simultánea. Para efectos de esta práctica, se permite el uso de hilos para soportar la concurrencia del servidor.

4. Detalles de Implementación y Uso de la Aplicación.

A continuación, se detallan algunos aspectos que se deben considerar para la realización de su proyecto.

- La aplicación cliente puede ser escrita en el lenguaje de preferencia del grupo que soporte la API sockets tal como: Python 3.X, Java, C, C++, C#, Go, etc.
- La aplicación servidor **SOLO** debe ser implementada en lenguaje C.
    - **Nota**: No se recibe ningún proyecto, en su totalidad, con la aplicación servidor desarrollada en un lenguaje diferente a C.
- No se permite utilizar (tanto para cliente como para servidor) ninguna clase existente o personalizada de sockets. Se debe utilizar la API de Berkeley.
- La aplicación servidor se debe desplegar y ejecutar en un servidor en nube. Para esto utilice la cuenta de AWS academy.
- Para documentar el diseño e implementación de su solución, por favor utilice herramientas como UML o cualquier otra con el fin de ilustrar las funcionalidades, así como el funcionamiento de este.

5. **Aspectos Para Considerar para el Desarrollo:**
    - **Equipo de trabajo:** El proyecto debe ser desarrollado en grupos de 3 personas como máximo. **NO** debe ser desarrollado de manera individual.
    - **Cronograma**: Defina un plan de desarrollo, hitos, victorias tempranas, hasta la finalización del proyecto.
        - Tenga presente que tiene un mes calendario para desarrollar el proyecto. Se espera que el 60% del tiempo lo invierta en el análisis, diseño, implementación, pruebas y documentación para la aplicación servidor. El otro 40% restante, para la aplicación cliente.
    - **Punto de Chequeo 1:**
        - **20 – 25 de Marzo**. Ya para esta fecha, usted debería tener la aplicación Servidor totalmente implementada.
    - **Tiempo**: Debe gestionar muy bien su tiempo para poder alcanzar el objetivo final del proyecto. Empiece a trabajar apenas le publiquen su enunciado.
    - **Consultas/dudas**: Si tiene alguna duda, por favor, acuda a su profesor y/o coordinador de la asignatura lo mas pronto posible.
    - **Cliente o Servidor:** Se sugiere que empiece por desarrollar la aplicación servidor y luego continue con la aplicación cliente

6. **Entrega**

    - **Repositorio:** Trabaje con git. Cree un repositorio privado para su proyecto. Recuerde que usted no puede compartir el código de su trabajo con nadie.
    - **Documentación:** La documentación se debe incluir en el repo en un archivo README.md. En este archivo se requiere que usted incluya los detalles de implementación donde como mínimo se esperan las siguientes secciones:
        - Introducción.
        - Desarrollo.
        - Aspectos Logrados y No logrados.
        - Conclusiones.
        - Referencias.
    - **Video**: Entregue y sustente al profesor mediante un video creado por el grupo, donde explique el proceso de diseño, desarrollo y ejecución (no más de 20 mins). Posteriormente se le citará a una sustentación presencial. Se debe ser claro en el video el funcionamiento de la solución, tanto para el cliente como para el servidor.
    - **Fecha de entrega Final:** abril 9 de 2024.
    - **Mecanismo de entrega:** Por el buzón de Interactiva virtual se debe realizar la entrega. La entrega debe incluir un enlace a un repositorio en github. A partir de esta fecha y hora, no se puede realizar ningún commit al repositorio.

7. **Evaluación**

    - Se realizará un proceso de sustentación para la verificación de la práctica acorde a lo entregado por el buzón y con lo explicado inicialmente en su video.
    - Posteriormente será actualizado y publicado en este documento los criterios de evaluación.

8. **Referencias**

- https://beej.us/guide/bgnet/
- https://beej.us/guide/bgc/
- https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
- RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing (ietf.org)