Proyecto Integrador de Redes y Sistemas Operativos: Server Intermedio de LEGO

A.A. Arévalo Alvarado[1], A. Badilla Olivas[1],
G. Rivera Hernández[1], J. P. Chacón González[1]

E.C.C.I., Universidad de Costa Rica.

Contributing authors: ariel.arevalo@ucr.ac.cr;
anthonny.badilla@ucr.ac.cr; geancarlo.riverahernandez@ucr.ac.cr;
jean.chacongonzalez@ucr.ac.cr;

Resumen

En esta etapa del proyecto, se agregó un servidor intermedio al sistema cliente-servidor existente. El servidor intermedio actúa como un punto de conexión entre el cliente y uno o varios servidores de piezas. Se realizaron modificaciones en el cliente y el servidor de piezas para permitir la inclusión del servidores intermediarios y seguir las directrices establecidas en el protocolo elegido por el grupo para establecer las comunicaciones entre ellos. El proyecto se basó en una arquitectura modular similar al patrón MVC y se crearon diversas clases para manejar la comunicación, el almacenamiento y la representación de las figuras según le respondiera a cada servidor. Se describen los resultados de las pruebas realizadas y se concluye reflexionando sobre la complejidad de desarrollar un programa completo con funcionalidad de red y se valora el camino recorrido y el aprendizaje obtenido a lo largo de esta y todas las etapas anteriores.

Keywords: HTTPS, servicio de red, protocolos de comunicación

1. Introducción

Esta investigación presenta el desarrollo e implementación de un servidor intermediario entre cliente y servidores de Piezas Lego, este se comunica con el cliente mediante TCP utilizando el protocolo HTTPS para recibir solicitudes de figuras específicas, mientras que establece una conexión UDP con los servidores de piezas para

solicitar las piezas correspondientes para armar la figura solicitada por el cliente, gracias al uso de una tabla de enrutamiento que indica cuáles figuras hay en cada servidor de piezas. El proyecto se llevó a cabo en varias etapas, y en esta cuarta y última etapa se enfocó en la construcción del servidor intermedio y la implementación del esquema de comunicación entre servidores intermediarios y servidores de figuras establecido en conjunto con la clase, por lo que se realizaron los cambios correspondientes en el cliente y servidor de piezas para realizar esta inclusión. El documento describe la metodología utilizada para lograr estos objetivos y presenta los resultados obtenidos. El servidor en funcionamiento fue desarrollado para entornos Linux y fue probado con éxito según los requisitos del proyecto.

2. Desarrollo

2.1. Servidor Intermediario

Al igual que el cliente y el servidor de piezas, el intermediario se llevó a cabo en `C++` siguiendo de igual manera una arquitectura modular en la cual se intenta adecuadamente separar responsabilidades en los diversos componentes. Además, la arquitectura se podría caracterizar como una variante de *MVC* de la siguiente forma:

- Modelo: Existe una clase que representa el proceso de solicitar, obtener y devolver una figura de *LEGO*, mediante la solicitud al sevidor de piezas.
- Vista: Tenemos únicamente una clase de bitácora, o registro, la cual despliega a la consola el estado del servidor.
- Controlador: La funcionalidad presente para ser parte del controlador consiste en la respuesta a la solicitud de una figura por parte de un cliente, solicitando la figura al servidor de piezas correspondiente devolviéndola en formato html al cliente una vez obtenida.

Para el desarrollo del sevidor intermedio se elaboraron las siguientes clases:

2.1.1. ProxyHttpsServer

Es la clase encargada de atrapar, analizar y descomponer los request enviados por los clientes para determinar cuál es la figura solicitada y además montar el response correspondiente una vez obtenida la figura desde el servidor de piezas.

2.1.2. ProxyHttpsController

Dirige el proceso de búsqueda en la tabla de enrutamiento para obtener la ip de algún servidor que contenga la figura solicitada por el cliente.

2.1.3. ProxyProtocolController

Maneja la comunicación con los servidores de piezas. Este se comunica con los servidores de piezas activos para que le contesten con las figuras que contiene cada uno y así poder armar su tabla de entrutamiento correspondiente, para saber a quién solicitar una figura específica en una futura solicitud de un cliente. Además, cada vez

que se enciende un nuevo servidor de piezas, este recibe el mensaje para añadir a la tabla de enrutamiento la entrada correspondiente a ese nuevo servidor.

2.1.4. ProxyRoutingTable

Es la tabla de enrutamiento encargada de mapear cada figura con las IP correspondientes a los servidores de piezas que contengan esa figura.

2.1.5. ProxySslClient

Es el encargado de enviar la figura al cliente en formato HTML una vez montado el response.

2.1.6. LegoProxy

Es el main asociado a este servidor, encargado de inicializar las instancias necesarias para encender el servidor y apagarlo de manera adecuada.

2.2. Cambios al Lego Client

El lego client únicamente se modificó para que ahora solicitara la IP correspondiente al servidor intermedio al que se desea realizar la solicitud.

2.3. Cambios al Servidor de Piezas Lego

El servidor de piezas se modificó para que ahora atendiera solicitudes de servidores Intermediarios en lugar de solicitudes de clientes. Se agregaron dos clases para gestionar directivas del protocolo establecido, mostradas a continuación.

2.3.1. FigureProtocolController

Esta clase se encarga de que cada vez que se encienda el servidor, enviar un mensaje por broadcast a todos los servidores intermediarios activos, para avisarles que él ahora está activo, además busca en su repositorio de figuras cuáles contiene y las adjunta al mensaje de presente para que los servidores intermediarios puedan actualizar su tabla de enrutamiento. De igual manera, esta clase se encarga de recibir mensaje de servidor intermedios que se encienden posteriormente, para responderles directamente por unicast cuáles figuras contiene.

2.3.2. FigureSSLController

Es la clase encargada de retornar una figura HTML al servidor intermedio solicitante.

2.4. Clases comunes entre servidor intermedio, de piezas y cliente

Para mejorar la calidad de código y evitar redundancia se definieron las siguientes clases comunes para todas las entidades del proyecto mostradas a continuación:

2.4.1. Logger

Se encarga de toda la impresión de la información que debe desplegar el programa, información tanto de errores (con un *traceback* del lugar en el que ocurrió el error), como de excepciones. Así mismo, se encarga de llevar el control del tiempo que cada método tarda en completar su ejecución, así como el estado actual del servidor.

2.4.2. Queue

Cola thread safe, para gestionar las solicitudes con hilos.

2.4.3. Semaphore

Semáforo implementado para hacer que la cola sea thread safe y evitar esperas activas u otros inconvenientes al utilizar hilos.

2.4.4. Ipv4SslSocket

Maneja una conexión SSL sobre TCP sobre IPv4. Facilita la apertura de esta conexión, así como la lectura y escritura a través de la misma. Hace uso de `sys/socket.h`.

2.4.5. SslCtxPtr y SslPtr

Consisten en wrappers RAII para los punteros de sistema `SSL` y `SSL_CTX`.

2.4.6. Ipv4UdpSocket

Maneja una conexiones UDP sobre IPv4. Facilita la apertura de esta conexión, así como la lectura y escritura a través de la misma.

2.4.7. ProtocolClient

Esta clase se encarga de montar todos los mensajes correspondientes y establecidos en el protocolo, para poder establecer comunicación entre intermediarios y servidores de piezas, y que así puedan mapear quienes están presentes en la red.

2.4.8. ProtocolServer

Se encarga de analizar y descomponer los mensajes establecidos en el protocolo para determinar qué tipo de mensaje es y poder actuar o responder en consecuencia a ello.

2.4.9. ProtocolHeader

Este es un header creado por todo el grupo, el cuál define las constantes o códigos definidos para cada mensaje del protocolo.

2.4.10. ProtocolServer

Es la clase encargada de mantenerse escuchando mensajes de protocolo y responder a ellos según corresponda

2.4.11. SslServer

Clase común para atender solicitudes Https.

2.5. Especificación del protocolo implementado para la comunicación

2.5.1. Características generales

Toda conexión TCP se establece sobre SSL. Por ende, el Cliente en NachOS queda descartado para formar parte de la comunicación.

2.5.2. Protocolo de conexión

El descubrimiento de nuevos servidores se realiza de manera asincrónica al levantarse un servidor, ya sea intermediario o de piezas. Al levantarse un servidor, este se identifica a todos los presentes (que les interese) y estos responden para poder completar sus tablas internas o proporcionar información. Si una solicitud de figura por parte de un cliente no se encuentra en la tabla de rutas del servidor intermedio, no es necesario que el servidor intermedio se comunique con otros servidores de piezas, puede indicarle al cliente que su solicitud es inválida.

2.5.3. LegoDiscover (Realizado por Intermediario)

Se realiza un broadcast a través de UDP, el cual solicita a los servidores de piezas que reciban este mensaje que se reporten respecto al servidor emisor. Los únicos emisores de este mensaje pueden ser los servidores intermedios, y los únicos receptores pueden ser servidores de piezas.

Código	Separador	IP
DISCOVER	-	host:puerto

2.5.4. LegoPresent

Su función es que los servidores de piezas se puedan identificar a los servidores intermedios levantados. Le indica su dirección IP y el puerto junto a un conjunto de figuras que ofrece el servidor de piezas. Utiliza UDP, si este apenas se levanta debe realizarlo en forma broadcast, de lo contrario sería una respuesta unicast hacia el LEGO DISCOVER hecho por el servidor intermedio si se levanta después, en este caso los mensajes UDP van dirigidos al puerto 3142, el cual corresponde al puerto utilizado por el servidor intermedio.

Código	Separador	IP	Separador	Piezas	Separador	Piezas...
PRESENT	char(29)	host:puerto	char(29)	nombre	char(29)	nombre...

2.5.5. LegoRequest

Su función es solicitar una figura al servidor de piezas. Utiliza el protocolo TCP, estableciendo la conexión por el puerto 3141, el cual corresponde al servidor de piezas.

Código	Separador	Figura
REQUEST	char(29)	nombre

2.5.6. LegoResponse

Su función es responder a un servidor intermedio con una figura solicitada. Utiliza el protocolo TCP, estableciendo la conexión por el puerto 3142, el cual corresponde al servidor intermedio.

Código	Separador	Figura	Separador	.Buffer
RESPONSE	char(29)	host:puerto	char(29)	HTML

El formato HTML de las figuras es el siguiente:

```
<HR>
<CENTER> <H1> Nombre: NombreFigure </H1>
<H2> Lista de piezas para armar la figura del final </H2> </CENTER>
<TABLE BORDER=1 BGCOLOR= lightblue CELLPADDING=5 ALIGN=CENTER>
<TR> <TH> Cantidad </TH> <TH> Descripcion </TH> </TR>
<TR><TD ALIGN=center> pieza cantidad </TD>
<TD ALIGN=center> pieza_tipo </TD>
</TR>
. . . .
</TR>
<TR>
<TD COLSPAN=2> Total de piezas para armar esta figura </TD><TD ALIGN=center>
<TD ALIGN=center>
      <H2> piezasTotal </H2>
    </TD>
</TR></TABLE>
```

Cada variable del buffer significa lo siguiente:

Variable	Detalle
piezaTipo	Especifica un tipo de pieza.
piezaCantidad	Cantidad de piezas de un mismo tipo.
piezasTotal	Número total de piezas que conforman la figura.

2.5.7. LegoRelease

Su función es indicar a un servidor intermedio que el servidor de piezas emisor dejará de estar disponible. Utiliza el protocolo TCP, estableciendo la conexión por el puerto 3142, el cual corresponde al servidor intermedio.

Código	Separador	IP
REALEASE	char(29)	host:puerto

2.5.8. Características

Servidor Intermedio:
UDP: Espera comunicación en 3141.
TCP: Espera comunicación en 3142.
Servidor de Piezas:
UDP: Espera comunicación en 4849.
TCP: Espera comunicación en 4850.
Se entiende que se necesita un hilo independiente para la escucha de mensajes en TCP y UDP, es decir, un hilo por cada socket.

3. Resultados de Pruebas

Salida al encender el servidor de piezas:

```
[107 ms][INFO]: ProtocolServer: Listening
[1108 ms][INFO]: SslServer: Listener certificates:
Server certificates:
Subject: /C=CR/ST=ALAJUELA/L=ALAJUELA/O=ECCI/OU=ECCI/CN=PIRO
Issuer: /C=CR/ST=ALAJUELA/L=ALAJUELA/O=ECCI/OU=ECCI/CN=PIRO

[1111 ms][INFO]: SslServer: Listening at 192.168.127.183
```

Salida al aceptar y responder una solicitud válida en el servidor de piezas:

```
[100828 ms][INFO]: ProtocolServer: Received message on worker 0
[100828 ms][INFO]: ProtocolServer: Handling LEGO_DISCOVER on worker 0
```

Resultado o salida al recibir un URL inválido en el servidor de piezas:

```
[2374 ms][INFO]: Listener certificates:
Server certificates:
Subject: /C=CR/ST=ALAJUELA/L=ALAJUELA/O=ECCI/OU=ECCI/CN=PIRO
Issuer: /C=CR/ST=ALAJUELA/L=ALAJUELA/O=ECCI/OU=ECCI/CN=PIRO

[2374 ms][INFO]: Listening.
[5927 ms][INFO]: Accepted connection with socket: 5
GET /leg/unicorn HTTP/1.1
Host: localhost:7777
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/11
...

[6439 ms][INFO]: Sending 404 response to client
(caused by invalid URL Format): 5
```

Resultado o salida al recibir una solicitud de una figura inválida o que no existe en el servidor de piezas:

```
[1694 ms][INFO]: Listener certificates:
Server certificates:
Subject: /C=cr/ST=sanjose/L=sanjose/O=ECCI/OU=UCR/CN=localhost
Issuer: /C=cr/ST=sanjose/L=sanjose/O=ECCI/OU=UCR/CN=localhost

[1694 ms][INFO]: Listening.
[5773 ms][INFO]: Accepted connection with socket: 5
[6285 ms][INFO]: Client request:
GET /lego/u
Host: 127.0.0.1
[6285 ms][INFO]: Sending 404 response to client (caused by FigureNotFound): 5
```

Resultado o salida al encender el servidor intermedio:

```
[30 ms][INFO]: ProtocolServer: Listening
```

Resultado o salida al encender el servidor intermedio:

```
[30 ms][INFO]: ProtocolServer: Listening
```

Resultado o salida del servidor intermedio al recibir un present de un servidor de piezas:

```
[32 ms][INFO]: ProtocolServer: Received message on worker 5
[32 ms][INFO]: ProtocolServer: Handling LEGO_PRESENT on worker 5
[32 ms][INFO]: ProtocolController: Adding entry:
         Figure: blacksheep
         IP: 192.168.127.183
[32 ms][INFO]: ProtocolController: Adding entry:
         Figure: dragon
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: duck
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: elephant
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: fish
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: giraffe
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: horse
         IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
         Figure: lion
         IP: 192.168.127.183
```

```
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: monkey
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: penguin
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: roadrunner
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: shark
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: squid
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: swan
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: turtle
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: unicorn
        IP: 192.168.127.183
[33 ms][INFO]: ProtocolController: Adding entry:
        Figure: whitesheep
        IP: 192.168.127.183
[1030 ms][INFO]: SslServer: Listener certificates:
Server certificates:
Subject: /C=CR/ST=SanJose/L=SanJose/O=ECCI/OU=PIRO/CN=PIROSERVER
Issuer: /C=CR/ST=SanJose/L=SanJose/O=ECCI/OU=PIRO/CN=PIROSERVER
```

Resultado o salida del cliente al solicitar una figura al intermediario:

```
[1097 ms][INFO]: HORSE
5 pieces of brick 2x2 yellow
4 pieces of brick 2x4 white
2 pieces of brick 2x4 yellow
1 pieces of brick 2x6 yellow
1 pieces of brick 2x3 yellow
2 pieces of brick 1x1 yellow eye
2 pieces of brick 2x1 yellow
1 pieces of brick 2x2 white
Total amount of parts for horse is: 18
[1097 ms][INFO]: Finished.
```

Resultado o salida del intermedio al recibir una solicitud del cliente:

```
[764950 ms][INFO]: SslServer: Accepted connection
[765459 ms][INFO]: HttpsServer: Received request on socket 7
[765488 ms][INFO]: SslClient: Redirecting request for: horse
[766014 ms][INFO]: HttpsServer: Client request on socket 7: GET /lego/horse
Host: 192.168.127.183


[766014 ms][INFO]: HttpsServer: Sending response to client on socket 7
[766014 ms][INFO]: HttpsServer: Handled connection on socket 7.
```

Resultado o salida del servidor al recibir una solicitud de figura del intermedio:

```
[866279 ms][INFO]: SslServer: Accepted connection
[866788 ms][INFO]: SslServer: Received request on socket 7
[866812 ms][INFO]: SslServer: Handled connection on socket 7
```

4. Conclusión

Durante el desarrollo de este proyecto, hemos obtenido un valioso aprendizaje al enfrentar y superar diversas barreras y dificultades. Desde el diseño inicial hasta la implementación final, obtuvimos una noción sobre la gran complejidad que representan los sistemas de red y la interacción de sus componentes. Además, fortalecimos habilidades de resolución de problemas y exploramos nuevas áreas de conocimiento debido a la amplia investigación que realizamos para poder desarrollarlo. Este proyecto nos ha proporcionado una experiencia significativa y una base sólida para futuros desafíos en el desarrollo de software y las redes.