Proyecto Integrador de Redes y Sistemas Operativos: Server 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

Este estudio presenta el desarrollo e implementación de un servidor de piezas de LEGO utilizando el protocolo HTTPS para recibir y proporcionar figuras específicas. El proyecto se basó en una arquitectura modular similar al patrón MVC y se crearon ocho clases para manejar la comunicación, el almacenamiento y la representación de las figuras. Además, se propone un protocolo para la comunicación entre servidores intermediarios y servidores de piezas. Se describen los resultados de las pruebas realizadas y se concluye reflexionando sobre la complejidad de desarrollar un programa con funcionalidad de red y las expectativas para las futuras etapas del proyecto.

Keywords: HTTPS, servicio de red, protocolos de comunicación

1. Introducción

Esta investigación presenta el desarrollo e implementación de un servidor de Piezas Lego que utiliza el protocolo HTTPS para recibir solicitudes de figuras específicas y proporcionar las piezas necesarias para construir el objeto solicitado. El proyecto se llevó a cabo en varias etapas, de las cuales la construcción del servidor, así como la definición de un esquema de comunicación definitivo entre servidores intermediarios y el servicio de figuras por varias partes acordantes, representan la segunda etapa. 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. Server

Al igual que el cliente, el servidor 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 obtener y devolver una figura de *LEGO*, que esta acompañada de la capa de datos, la cual consiste en un repositorio para las figuras.
- 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, buscando la figura en su repositorio y devolviéndola en formato html.

Para el desarrollo del server se elaboraron ocho clases:

2.1.1. 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.

2.1.2. FigureController

Dirige el proceso de obtener la figura deseada por el usuario y la devuelve en formato html. Busca esta figura por nombre al repositorio, el cual encapsula la obtención de la misma.

2.1.3. FigureHtmlRepository

Maneja el suministro de figuras. Esta busca las figura solicitada en su repositorio utilizando un path para obtener los datos de cada figura que se le solicita, además hace el manejo de excepciones en caso de que la figura no se encuentre en el servidor.

2.1.4. FigureHttpsServer

Implementa la funcionalidad de servidor esperando y aceptando solicitudes de clientes, para mantener múltiples conexiones hace uso hilos y de un socket pasivo, conforme establece el protocolo HTTPS.

2.1.5. Queue

Cola thread safe, para gestionar las solicitudes de los clientes con hilos.

2.1.6. Semaphore

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

2.1.7. 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`.

2.1.8. SslCtxPtr y SslPtr

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

2.2. Protocolo para Servidor Intermediario Grupo03

Se va a trabajar por tipos de mensajes que pueden enviar los servidores de piezas e intermedios. Cada mensaje cuenta con un formato establecido. Cada uno de ellos cuenta con un código de mensaje que ayuda a identificar el tipo de mensaje. Además, los mensajes enviados son cadenas de caracteres.

2.2.1. LegoDiscover

Su función es descubrir los servidores de piezas, por lo tanto es únicamente transmitido por el servidor intermedio. Se le indica la dirección IP junto con el puerto del servidor intermedio. Utiliza el protocolo UDP en broadcast, en el puerto 3141, el cual corresponde al puerto del servidor de piezas.

Código	Separador	IP
0	-	host:puerto

2.2.2. 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...
1	-	host:puerto	-	nombre	-	nombre...

2.2.3. 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
2	-	nombre

2.2.4. 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
1	-	host:puerto	-	HTML

El formato HTML de las figuras es el siguiente:

```
<HR>
<CENTER> <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> <TH> Imagen </TH> </TR>
<TR><TD ALIGN=center> pieza cantidad </TD>
<TD ALIGN=center> pieza_tipo </TD>
<TD ALIGN=center> <IMG SRC= pieza_img width=100 height=100></TD>
</TR>
. . . .
<TR><TD COLSPAN=3> <IMG SRC= figura_img width=500 height=500></TD>
</TR><TR><TD COLSPAN=2> Total de piezas para armar esta figura </TD><TD ALIGN=c
```

Cada variable del buffer significa lo siguiente:

Variable	Detalle
piezaImg/figuraImg	Especifica el url de una imagen.
piezaTipo	Especifica un tipo de pieza.
piezaCantidad	Cantidad de piezas de un mismo tipo.
piezasTotal	Número total de piezas que conforman la figura.

2.2.5. 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.

2.2.6. Características

Puerto del servidor piezas: 3141 Puerto del servidor intermedio: 3142 Si un servidor de piezas no responde al tercer llamado será declarado como fuera de servicio. La definición de este protocolo no tiene incidencia sobre el funcionamiento del protocolo HTTP utilizado entre el servidor de intermedio y los clientes

3. Resultados de Pruebas

Salida al encender el servidor:

```
[1621 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

[1621 ms][INFO]: Listening.
```

Salida al aceptar y responder una solicitud válida:

```
[1621 ms][INFO]: Listening.
[18747 ms][INFO]: Accepted connection with socket: 5
[19258 ms][INFO]: Sending response to client: 5
[19258 ms][INFO]: Handled connection with socket: 5
[31102 ms][INFO]: Exiting.
```

Resultado o salida al recibir un URL inválido:

```
[1621 ms][INFO]: Listening.
[18747 ms][INFO]: Accepted connection with socket: 5
[19258 ms][INFO]: Sending 404 response to client (caused by invalid URL Format): 5
[19258 ms][INFO]: Sending 404 response to client (caused by figureNotFound): 5
[19258 ms][INFO]: Handled connection with socket: 5
^C[31102 ms][INFO]: Exiting.
```

Resultado o salida al recibir una solicitud de una figura inválida o que no existe:

```
[1621 ms][INFO]: Listening.
[18747 ms][INFO]: Accepted connection with socket: 5
[19258 ms][INFO]: Sending 404 response to client (caused by figureNotFound): 5
[19258 ms][INFO]: Handled connection with socket: 5
^C[31102 ms][INFO]: Exiting.
```

4. Conclusión

Con base en la experiencia de desarrollar un servicio de red utilizando facilidades dispuestas por el sistema operativo, se ha logrado obtener una renovada apreciación por la multitud de partes en movimiento que cooperan para posibilitar la telecomunicación. Aún por limitada que fuera la experiencia solo de construir únicamente el mecanismo receptor de esta interacción, se ha logrado cultivar un entendimiento más profundo de la complejidad requerida para llevar a cabo un programa que presente funcionalidad de red.