 # CI-0123 ESJOJO

Problema

Plasmar diferentes conceptos de las áreas de redes y sistemas operativos en la programación un servicio que podrá ser atendido de manera redundante por varios servidores que pueden correr en máquinas distintas.

Este proyecto crea un programa que obtiene y muestra las piezas de figuras Lego para la construcción de una figura en específico para el usuario.

Específicamente debe construyen programas para:

• Servidores intermedios

• Servidores de piezas

• Clientes buscadores de piezas

Primera entrega 📒

En la primera etapa de proyecto se realiza un programa que a través de un despliegue de menú en el shell permite al usuario solicitar las figuras de Lego que desea y se le muestre la información de las piezas requeridas.

Requerimientos

1. Obtener el menú de figuras a través del servidor web
2. Obtener las piezas requeridas para la figura solicitada por el cliente a través del servidor web
3. Compilado y probado con una versión mínima de Ubuntu 22.04

4. Bibliotecas:

    1. regex de la biblioteca estandar
    2. open ssl
    3. arpa/inet

CLIENTES

Crear un cliente que utilice conexión segura SSL para solicitar a un servidor una figura específica y obtener la información de las piezas necesarias para construir el objeto solicitado.

Resolución

Clase Client

Se crea la clase Client que utiliza una conexión segura con SSL para conectarse al servidor web https://os.ecci.ucr.ac.cr/lego/ y realizar solicitudes con la finalidad de conseguir información de menú de figuras Lego disponibles y las piezas requeridas para la figura solicitada, se programa sockets con acceso seguro (SSL).

Se implementa los siguientes métodos para la correcta funcionalidad de la clase Client:

connectServer(): Establece conexión segura SSL con el servidor web

makeRequest(): Se encarga de crear un socket, si no ha sido creado, establecer conexión con el request específico y llamar a processRequest().

inAnimalArray(): se encarga de agregar un animal al arreglo interno de animales, y si está el animal dentro del arreglo entonces solo confirma si este ya se encuentra presente.

regexAnalyzer(): analiza segmentos de código dados, dependiendo de si son del menú o de una figura específica, y realiza operaciones relacionadas con lo leído.

processRequest(): Procesa la respuesta de servidor web después de que se realiza una solicitud, se utiliza la biblioteca regex para el análisis de lenguaje html que el servidor web construye, que por medio de expresiones regulares se saca la información solicitada. En caso que se haya pedido el menú, se saca los nombres de las figuras por medio de la respuesta de servidor web y los agrega en el vector de figuras que tiene el Client. Por otro lado, si se solicita las piezas de una figura, se saca y despliega la información correspondiente

Expresiones Regex (Expresiones regulares)

Para encontrar figuras en el menu principal:

```markdown
"<OPTION\\s+value=\"(?!None\")([^\"]+)\">"

1. <OPTION\\s+ -> Se busca la etiqueta [<OPTION] y luego busca 0 o más espacios en blanco

2. value=\" -> Se busca la etiqueta [value="]

3. (?!None\") -> Indica que [None"] no debe de estar seguida de [value="]

4. ([^\"]+)\"> -> Este grupo es cualquier carácter que no tenga comilla uno o más veces y se guarda este valor y debe de terminar con ">
```

Para encontrar lego y su cantidad correspondiente:

```markdown
"<TR><TD ALIGN=center> (\\d+)</TD>\\s*<TD ALIGN=center> ([^<]+)</TD>"

1. <TR><TD ALIGN=center> -> Se busca la etiqueta [<TR><TD ALIGN=center> ]
```

2. (\\d+) -> Este grupo es coge cualquier número una o más veces

3. </TD>\\s*<TD ALIGN=center> -> Busca la etiqueta [</TD>] seguido de 0 o más espacios en blanco y seguido de la etiqueta [<TD ALIGN=center>]

4. ([^<]+)</TD> -> Este otro grupo atrapa cualqueir carácter que no sea [<] una o más veces, tiene que estar seguida de la etiqueta [</TD>]

```

**### Segunda entrega** 📒

**## Clase PiecesServer**

Es capaz de atender las solicitudes de los clientes por piezas para

armar figuras y devolver el listado de las piezas para armar la figura, estas solicitudes utilizan el protocolo HTTP. PiecesServer es un singleton.

Los servidores de piezas tendrán información del inventario por medio de archivos .txt necesario para construir cada tipo de figura. El inventario es un conjunto de piezas que indica el nombre de la pieza y la cantidad, este inventario de piezas se disminuye cuando el cliente pide armar una figura. También es importante que al principio de que los archivos .txt para agregar al inventario debe de venir al principio  "Lego source File :: group ESJOJO"

Ejemplo:

```markdown

Lego source File :: group ESJOJO

brick 2x6 plate grey

2

brick 2x4 grey

3

brick 2x2 grey

2

brick 2x2 slope inverted grey

1

```

Componentes Importantes de la Clase

Los servidores de piezas crea otros trabajadores (hilos o procesos) para realizar la

atención de las solicitudes. El proceso principal no debe realizar esa atención.

Cuenta con 2 puertos, uno para los clientes que corresponde a 2816 y otro para el browser que es 2832. Para el socket cliente se conecta por medio de SSL y el browser con TCP.

En esta clase cuenta con 4 hilos principales:

1. listeningThread: Escucha por peticiones del cliente

2. clientRequestHandler: Atiende peticiones del cliente

3. browserListenThread: Escucha por peticiones del browser (mostrar inventario únicamente)

4. browserRequestHandler: Atiende peticiones del browser (mostrar inventario únicamente)
```

También se cuenta con 2 colas con control de concurrencia para tener control de las solicitudes del cliente (clientQueue) y del browser(browserQueue).

Durante esta etapa y para las realizar pruebas, los clientes conecten

directamente con los servidores de piezas.

Funciones Importantes que realiza el Servidor de Piezas

1. startServer(): Empieza el servidor inicializando sockets, creando subprocesos y escuchando conexiones de clientes. También se encarga de hacer join al hilo clientRequestHandler y cierra el socket Cliente.

2. readLegoSourceFile(): Lea el archivo fuente de Lego y rellene el mapa de inventario de Lego.

3. stop(): Detiene el PiecesServer cerrando los sockets del servidor y haciendo join a los hilos restantes.

4. processBrowserRequests(): Procesa las solicitudes del navegador sacando clientes de la cola del navegador y atendiendo sus solicitudes.

5. processBrowserRequest(): Procesa una solicitud del navegador enviando una respuesta al cliente con una tabla de figuras de lego y sus montos.

6. processClientRequests(): Procesa las solicitudes de los clientes extrayéndolos de la cola de clientes y sirviendo sus solicitudes.

7. processClientRequest(): Procesa la solicitud de un cliente verificando si las piezas de Lego solicitadas están disponibles y respondiendo con un mensaje de éxito o falla.

8. processRequest(): Procesa la solicitud del cliente y extrae las piezas solicitadas de los datos recibidos.

Manual de Usuario 📄

Pruebas

Para la compilación de los casos de prueba, se puede usar el siguiente comando para facilitar la tarea:

`make test`

Con el fin de poder usar este comando se debe de descargar en el sistema **icdiff** que es una herramienta de comparación de archivos y directorios.

Sobre las pruebas

Para el client:

1. Prueba que el programa entre, cargue las figuras y termine la ejecucion correctamente.

2. Prueba el manejo de errores de input en el menu principal por parte del cliente y que a pesar de estos se puede continuar con la ejecucion correctamente.

3. Prueba el manejo de errores de input en el menu de una figura por parte del cliente y que a pesar de estos se puede continuar con la ejecucion correctamente.

4. Prueba que se puede hacer llamados continuos para revisar las figuras y que sin importar la cantidad, el programa se ejecuta sin problema alguno.

Para correr estas pruebas es necesario tener el servidor corriendo previo a la ejecucion del make test.

5. Prueba que se puede armar la figura cuando se encuentran partes suficientes, y que posteriormente, al no haber mas piezas, recibe un mensaje del servidor que no se pudo armar. (Solo hay suficientes piezas en la prueba para armar el dragon una vez)

6. Prueba que al no haber piezas para las demas figuras, no se pueden armar del todo.

Para el servidor:

No es posible realizar un make test debido a que el programa se cierra con ctrl+C el cual no es interpretado como tal dentro de un archivo input.txt. Por esto se necesita realizar las operaciones manualmente y posteriormente comparar con el archivo de output proveido.

Considerando esto, los siguientes describen los casos a probar.

1. Solo correr bin/Server y posteriormente ctrl+C. Se prueba que la secuencia de ejecucion es correcta.

2. Correr bin/Server, correr make test en la carpeta de Cliente, y al terminar las pruebas, ctrl+C. Se busca comprobar que la respuesta a pedidos desde el cliente son manejados correctamente.

Detener la Ejecución

En caso de que desee finalizar la ejecución del programa, presionaremos en nuestro dispositivo la letra Ctrl+C o con 0 en el input, tal como es indicado por la salida del programa. Esto hará que hilos finalicen su trabajo, hagan join y termine la ejecución de programa exitosamente.

Ejemplo de Ejecución 📸

Para iniciar una ejecución del programa se debe de hacer lo siguiente:

Dentro de la carpeta de proyecto lego-figure-maker del terminal, cambia de directorio a la carpeta Server con el siguiente comando:

`cd Server`

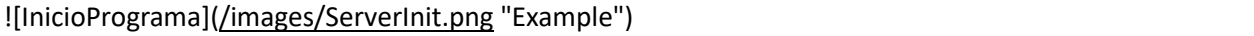
Utiliza el siguiente comando para compilar el programa de servidor de piezas y crear el ejecutable bin/Server:

`make`

Ejecuta bin/Server para comenzar con el programa:

`bin/Server`

Cuando se inicia el programa se deberá parecer a lo siguiente:



Abre un browser, y pon en el encabezado de la búsqueda "localhost:2832":

Después de ingresar al "localhost:2832" deberá aparecer el inventario del servidor de piezas que se representa con una tabla que muestra la cantidad y el nombre de la pieza, como lo siguiente:

Sin cerrar el programa de bin/Server, abre otra pestaña de terminal, dirijase a la carpeta Client

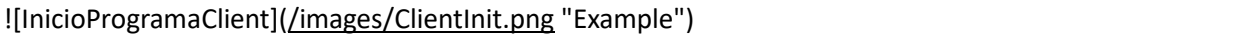
Utiliza el siguiente comando para compilar el programa de servidor de piezas y crear el ejecutable bin/Client:

`make`

Ejecuta bin/Client para comenzar con el programa:

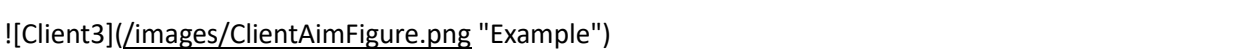
`bin/Client`

Cuando se inicia el programa deberá aparecer lo siguiente:



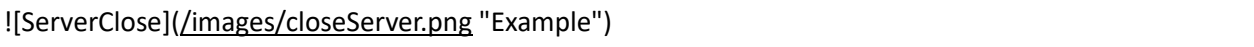
Cuando pide por la figura "dragon" deberá aparecer lo siguiente:

Cuando se pide armar la figura deberá aparecer lo siguiente:



El resultado de armar la figura correspondiente con éxito significa el cambio en el inventario del servidor de piezas, por lo que a refrescar el browser usado en el "localhost:2832" aparecerá lo siguiente:

Para cerrar el programa, dirijase al terminal que se encuentra en el directorio de Server y pon ctrl+c, aparecerá lo siguiente:



 # Protocolo grupal de comunicación PIRO 2023

Esquema de comunicación

- Los servidores intermedios y de piezas deben estar en la misma red

- Los Servidores Intermedios y de Piezas usarán el protocolo HTTP

- Los Servidores Intermedios no se comunican con otros Servidores Intermedios

- Los servidores intermedios deben conectarse con los servidores de piezas y escuchar por solicitudes de clientes

- Los servidores intermedios contienen un mapa de ruta, el cual indica los servidores de piezas que contienen las piezas necesarias para construir cierta figura

- Los servidores de piezas indican a los servidores intermedios las figuras que contienen

- Los clientes establecen una conexión con los servidores intermedios, nunca con los servidores de piezas

URI

Los servidores intermedios redirigen las solicitudes de los clientes a los servidores de piezas, de modo que las URIs utilizadas deben ser estandarizadas, por lo cual se ha acordado utilizar la forma “/piece”, donde “piece” específica la figura que desea el cliente. Por ejemplo si se desea la figura de Jirafa, la solicitud será de la siguiente manera:

```
GET /giraffe
```

Puertos

Se utilizarán 3 diferentes puertos para la comunicación en la red. Los servidores intermedios deben comunicarse con los clientes, así como con los servidores de piezas, para lo cual se ha acordado lo siguiente:

1. En el puerto 4850, el Servidor Intermedio va a enviar un GET de broadcast, y los Servidores de Piezas van a mandar un 200 OK con su tabla de figuras si se encuentra levantado.

2. En el puerto 4849 el Servidor Intermedio va a redirigir la request del cliente a los Servidores de Piezas, si es que lo encuentra en su tabla.

3. En el caso en el que el Servidor Intermedio no encuentre la figura solicitada en dentro de su tabla de rutas con los Servidores de Piezas, entonces hará de nuevo un broadcast por si algún nuevo Servidor de Piezas se levanta y tiene la figura pero no estaba en la tabla de rutas del Servidor Intermedio

Conexiones

Los servidores intermedios deben esperar el envío de bytes por parte de los Servidores de Piezas. Si estos, luego de cierta cantidad de tiempo no reciben bytes, deben eliminar la dirección IP de ese servidor de su mapa de rutas, puesto que el Servidores de Piezas no está activo.

Si el cliente le solicita una figura al servidor intermedio, cuyo servidor no tiene en su tabla de rutas, el servidor debe enviar un mensaje de broadcast a través del puerto 4850, con tal de encontrar dicho servidor. El servidor de piezas, entonces, responde con un mensaje indicando su dirección IP y las figuras que contiene.

En caso de que otro servidor intermedio reciba el mensaje, este simplemente descarta la solicitud.

Mensajes

Los mensajes entre cliente y servidor intermedio, y entre servidor intermedio y de piezas, se realizan sobre el protocolo HTTP 1.0. El contenido de los mensajes es sensible a la naturaleza de la solicitud a un servidor, y el puerto donde se realiza la conexión.

En particular, existen estas categorías de mensajes:

1. Mensajes de contenido entre cliente y servidor intermedio:

- Los clientes pueden realizar solicitudes GET para obtener el listado de figuras, y el listado de piezas para alguna figura en particular.

- Los servidores intermedios son responsables de responder con el listado de figuras en HTML al cliente.

- La distinción entre una solicitud del listado de figuras y el listado de piezas de alguna figura se realiza mediante el URI asociado al método GET.

- Un servidor intermedio debe devolver la respuesta HTTP con el listado de piezas de una figura que le respondió un servidor de piezas al cliente que las solicitó.

- De no poder contactar un servidor de piezas, el servidor intermedio podrá responder con su propia respuesta HTTP.

- De manera semejante, de obtener una solicitud mal formada, o una pieza inexistente, se debe responder con su propia respuesta HTTP.

- Aquellas respuestas de los casos de error de redirección a un servidor de piezas deben indicar que no se pudo encontrar la pieza. Se sugieren las respuestas 404 o 500 para mensajes de error HTTP.

2. Mensajes de contenido entre servidor intermedio y servidor de piezas:

- Los servidores intermedios deben enviar la solicitud HTTP de un listado de piezas de una figura enviada por un cliente hacia el servidor de piezas correspondiente.

- Al obtener la respuesta apropiada a esa solicitud, deben redirigirla hacia el cliente que la solicitó.

3. Mensajes de descubrimiento desde el servidor intermedio hacia servidores de piezas:

- Un servidor intermedio comienza desconociendo los servidores de piezas.

- Cuando un cliente solicita una pieza para la cual el servidor intermedio desconoce el servidor de piezas apropiado, éste debe preguntar mediante broadcast cuáles servidores servidores contienen esa pieza.

- Un servidor de piezas al recibir una solicitud de descubrimiento de piezas de un servidor intermedio debe responder apropiadamente. Se le puede preguntar cuáles figura soporta, o si soporta una pieza o no.

- Cuando un cliente solicita una pieza, y el servidor intermedio conoce el servidor de piezas apropiado, pero éste no responde tras un tiempo de timeout, a un tiempo no mayor a 2 segundos, el servidor intermedio se desconoce de aquel servidor de piezas y responde al cliente con una respuesta indicando que la pieza no se encontró.

El formato de los mensajes HTTP:

1. Solicitud de listado de figuras:

Método: “GET”

URI: “/”

Versión HTTP: “HTTP/1.0”

Cuerpo: Ignorado

2. Respuesta del listado de figuras (encontrado):

Estado: “OK” 200

Versión HTTP: “HTTP/1.0”

Cuerpo: Dirección IP del servidor, un cambio de línea (CRLF), junto a lista separada por comas de figuras que maneja el servidor, terminando en un punto.

3. Solicitud de listado de piezas para una figura:

Método: “GET”

URI: “/” seguido del nombre de la figura

Versión HTTP: “HTTP/1.0”

Cuerpo: Ignorado

4. Respuesta del listado de piezas para una figura (encontrado):

Estado: “OK” 200

Versión HTTP: “HTTP/1.0”

Cuerpo: Listado de piezas para una figura en HTML

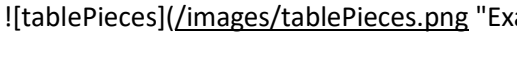
Notas:

- El listado de figuras y el listado de piezas para una figura se envían HTML.

- El contenido de las respuestas y solicitudes no tienen encoding particular (ni chunked). Se asume el final del mensaje como el final de la respuesta / solicitud HTML. Se sugiere considerar un timeout en la lectura.

- Ambos listados son libres de ser implementados según desee cada grupo, siempre y cuando le permita al cliente formar las solicitudes de figuras en el formato apropiado.

FIGURA LEGO: CHIKI



Integrantes 👥

• Esteban Porras Herrera - C06044

• Joseph Stuart Valverde Kong - C18100

• Johana Wu Nie - C08591

Evaluaciones 🌐

Evaluado por: Esteban Porras Herrera

• Esteban Porras Herrera - C06044 : 100/100

• Joseph Stuart Valverde Kong - C18100 : 100/100

• Johana Wu Nie - C08591 : 100/100

Evaluado por: Joseph Stuart Valverde Kong

• Esteban Porras Herrera - C06044 : 100/100

• Joseph Stuart Valverde Kong - C18100 : 100/100

• Johana Wu Nie - C08591 : 100/100

Evaluado por: Johana Wu Nie

- Esteban Porras Herrera - C06044 : 100/100

- Joseph Stuart Valverde Kong - C18100 : 100/100

- Johana Wu Nie - C08591 : 100/100