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 el protocolo HTTP 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 el protocolo HTTP 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 usando la dirección IP y el puerto 80.

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

inAnimalArray(std::string animal): 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(bool requestMenu, std::string& line): analiza segmentos de código dados, dependiendo de si son del menú o de una figura específica, y realiza operaciones relacionadas con lo leído.

processRequest(bool requestMenu): 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:

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

-<OPTION\s -> Se busca la etiqueta [<OPTION] y luego busca 0 o más espacios en blanco -value=" -> Se busca la etiqueta [value="] -(?!None") -> Indica que [None"] no debe de estar seguida de [value="] -([^"]+)\ -> Este grupo es cualquier carácter que no tenga comilla uno o más veces y se guarda este valor

Para encontrar lego y su cantidad correspondiente:

" (\d+)\s* ([^<]+)"

- -> Se busca la etiqueta [ ] -(\d+) -> Este grupo es coge cualquier número una o más veces -\s* -> Busca la etiqueta [] seguido de 0 o más espacios en blanco y seguido de la etiqueta [ ] -([^<]+) -> Este otro grupo atrapa cualqueir carácter que no sea [<] una o más veces, tiene que estar seguida de la etiqueta []

Manual de Usuario □

Compilación y Ejecución del código

Para poder compilar y correr el código, se provee un archivo Makefile que asiste en la compilación y construcción del programa. Con esto, para la compilación es solo necesario el siguiente comando:

```
make
```

Este comando corre el makefile y crea el ejecutable dentro de la carpeta bin/. El ejecutable será del nombre de la carpeta común, en cuyo caso, a como es proveído, sería ‘lego-figure-maker’.

Para poder correr el programa desde la carpeta común sería entonces el siguiente comando:

```
bin/lego-figure-maker
```

De querer borrar el ejecutable y todos los archivos relacionados generados, utilizar el siguiente comando:

```
make clean
```

Posterior a la ejecución solo es necesario seleccionar las opciones dadas por el output en consola para poder navegar dentro del programa.

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

```
make test
```

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.

Ejemplo de Ejecución □

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

Cuando el usuario pide por la figura "blacksheep" y luego solicita cerrar el programa se deberá parecer a lo siguiente:

PeticionYCierre

Protocolo

Establecer los esquemas de comunicación:

◦ Entre los clientes y los servidores intermedios se comunican por medio de una red pública con el puerto 80 (HTTP)

◦ Entre los servidores intermedios y los servidores de piezas se comunican por medio de una red privada, en un puerto diferente a 80

Cliente: Solicita el menú de figuras y las piezas necesarias a través de la solicitud GET de protocolo HTTP que se envía a través de la URL

Servidor Intermedio: Contiene el mapa de rutas. Este mapa de rutas se debe actualizar cuando identifica que se agrega un nuevo servidor de piezas.

Servidores de piezas: Realiza una revisión de los modelos que almacena y es quien brinda las piezas solicitadas por el cliente.

Valorar el uso de datos encriptados para las comunicaciones Encriptar datos con AES para el envío de datos en lo posible para toda conexión.

Protocolo de comunicación para adicionar o eliminar servidores de piezas a servidores intermedios o viceversa (interacción):

Puertos:

Puertos para intermediario:

Puerto: 2304 (clientes)

Puerto: 2432 (otros servidores intermediarios)

Puerto: 2560 (servidores de piezas)

Puertos para servidor de partes:

Puerto: 2816 (servidor intermediario)

Sucesion de eventos:

1. Primer Caso: Servidor intermediario se levanta antes que los servidores de pieza

```
Hace un broadcast a todos los puertos 2560 con su IP dentro del servidor y empieza a escuchar por el puerto 2304 a los clientes.
Al no haber ningun servidor de piezas entonces no recibe respuesta.
Se mantiene escuchando en el puerto 2304 a que algun servidor de piezas anuncie su levantamiento.
Al levantarse un servidor de piezas, este realiza un broadcast a todos los puertos 2304 dentro de la red, conjunto con su IP.
El servidor intermediario recibe el IP, y lo guarda en un mapa local para poder accesarlo cuando sea necesario.
```

2. Segundo Caso: Servidor de pieza se levanta primero que el servidor intermediario

```
El servidor de piezas realiza un broadcast con su IP a todos los puertos 2560 dentro de la red.
Al no encontrarse ningun servidor intermediario, no recibe respuesta.
Se levanta un servidor intermediario, y este realiza un broadcast a todos los puertos 2560 con su IP dentro del servidor y empieza a escuchar por el puerto 2304 a los clientes.
El servidor de piezas recibe el IP del servidor intermediarios y este responde a este IP con su IP.
```

3. Tercer Caso: conexion de un servidor intermediario con otro servidor intermediario

```
El servidor intermediario se levanta y tambien emite un broadcast con su IP a todos los puertos 2432 dentro de la red.
El otro servidor intermediario recibe el IP, lo agrega a su mapa local de IPs y responde al IP del otro servidor intermediario con su IP.
El primer servidor intermediario recibe el IP del otro servidor y lo agrega al mapa local de servidores intermediarios.
```

4. Cuarto Caso: se borra un servidor intermediario

```
Se deja de recibir peticiones por el puerto 2304.
Se finalizan de procesar las peticiones del servidor.
Se le envian mensajes a los otros servidores intermediarios con su IP y codigo de borrado en el ultimo byte.
Los otros servidores mandan un mensaje confirmacion de borrado.
De recibir un mensaje de confirmacion de todos, se borra.
De no ser asi, se vuelve a mandar el mensaje al menos 10 veces mas esperando respuesta antes de borrarse de todas maneras y reporta el error.
```

5. Quinto Caso: se borra un servidor de partes

```
Espera a recibir un mensaje del servidor intermediario que lo esta utilizando confirmando que se finaliza el checkeo o utilizacion, en el caso que esta siendo utilizado.

El servidor de partes realiza un broadcast con el codigo de borrado de servidor de piezas a todos los puertos 2560.

Todos los servidores intermediarios reciben el mensaje, borran el IP de sus mapa locales y mandan una confirmacion de borrado.

Al recibir los mensajes de confirmacion, el servidor local reduce un contador local de servidores intermediarios.

Al llegar este contador a 0 se borra.

De no llegar a 0, en una cantidad arbitraria de tiempo, este reporta el error antes de borrarse.
```

Nota: si es necesario diferenciar entre un broadcast y un mensaje de respuesta, se puede agregar un byte al final de cada mensaje con IP y en este se puede tomar el 00000000 como un broadcast, el 00000001 como un mensaje normal, 00000010 cuando se manda un mensaje de borrado de servidor intermediario, y 00000011 cuando se manda un mensaje de borrado de servidor de partes, el resto de los digitos pueden utilizarse para agregar mas funcionalidad funcionalidades en el futuro. Otro uso podria ser reservar los primeros 32 valores para mensajes generales y despues asignar 112 a mensajes de servidores de piezas y otros 112 a mensajes de servidores intermediarios, para mantener organizacion.

Paso de datos:

Se puede definir un formato propietario diferente al de html para los datos enviados. Se puede también definir que al enviarse los datos y que estos, a la mitad de un contenido, no caben dentro del mensaje enviado, se indique donde ocurrió tal interrupción, o simplemente guardarlo para este ser enviado dentro del siguiente mensaje.

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