MEMORIA ENTREGA FINAL

Sistemas Distribuidos G84

Adam Kaawach - 100438770 100438770@alumnos.uc3m.es

Rodrigo Jiménez - 100472245 100472245@alumnos.uc3m.es

uc3m | Universidad Carlos III de Madrid

1 - Código Implementado

En esta sección, detallaremos el código implementado tanto en el servidor como en el cliente.

1.1 - Cliente

Funciones de Ayuda:

- *getServerTime*(): Realiza una solicitud al servicio web de fecha y hora y devuelve la respuesta.
- *initSocket*(): En la mayoría de las funciones del cliente, es necesario conectarse al servidor. Dado que la dirección del servidor no cambia, utilizamos esta función en cada una de esas funciones que necesitan conectarse al servidor. Inicializamos el socket y nos conectamos al servidor, luego devolvemos el socket para su uso en el envío de mensajes.
- *connectToUser*(): Siguiendo la misma metodología de la función anterior, también debemos conectarnos con otros usuarios activos para intercambiar archivos. Por esta razón, también tenemos una versión de usuario de la función initSocket que se conecta a un usuario a través de su nombre de usuario.
- *sendTuple*(): A lo largo del proceso de diseño, tuvimos muchos problemas con el envío y la recepción, lo que a menudo resultaba en un error de red. Por esa razón, en lugar de enviar cada mensaje individualmente, los combinamos en una sola cadena separada por terminadores nulos y enviamos ese mensaje para ser analizado por el servidor. Esto requiere manejo adicional de errores.
- *updateUsers*(): Cuando un cliente lista los usuarios activos, se percibe que ese cliente ahora tiene acceso a los detalles de esos usuarios activos, por lo que

esta función se utiliza para actualizar la lista de usuarios en el servidor cuando un usuario se conecta.

Funciones Principales:

Es importante señalar que todas las siguientes funciones tienen la misma estructura: Preparar la cadena de operación, conectar al servidor utilizando las funciones de ayuda, enviar los contenidos requeridos por la función al servidor utilizando las funciones de ayuda, esperar la respuesta e imprimir el resultado correspondiente. Lo que varía es el tratamiento de la respuesta y el contenido a enviar. En las siguientes afirmaciones, discutiremos los roles individuales que desempeñan para mantener la funcionalidad de los clientes.

- *register*: Envía la solicitud de registro.
- *unregister*: Envía las solicitudes de desregistro. Si la operación es exitosa, asegurará que todos los atributos se restablezcan para que el cliente pueda desconectarse correctamente y un nuevo cliente pueda conectarse.
- *connect*: Envía la solicitud de conexión. La función de conexión es responsable de crear un hilo de cliente al que otros clientes pueden conectarse cuando sea necesario. Por esa razón, inicializamos el socket del cliente, luego enviamos la información requerida al servidor. Luego se crea un hilo para manejar cualquier tráfico entrante. En el evento de que la conexión sea exitosa, establecemos el nombre de usuario del cliente al usuario conectado.
- *handle_incoming**: Primero, usamos updateUsers para enviar una actualización al servidor que contiene este usuario recién conectado. Luego esperamos la conexión de cualquier otro cliente, lo que en nuestro caso solo debería ocurrir con la función getfile. Al recibir una conexión, recibimos

1024B de datos, los decodificamos, eliminamos los terminadores nulos y, dado que sabemos que el comando siempre será el primero, verificamos si el comando es GET_FILE y solo entonces operaremos sobre los datos como se discutió en getfile.

- *disconnect*: Envía la solicitud de desconexión y asegura que el cliente se desconecte restableciendo los atributos del cliente.
- *publish*: Envía la solicitud de publicación con los parámetros requeridos.
- *delete*: Envía la solicitud de eliminación al servidor.
- *listusers*: Envía la solicitud de listusers. En el evento de que el resultado sea un éxito, readString se usa para leer la lista completa que fue enviada por el servidor pero dado que esa lista es y puede usarse en otro lugar, se analiza en un diccionario y se establece como el atributo de lista de usuarios del cliente.
- *listcontent*: Envía la solicitud de listado de contenido e imprime el resultado recibido.
- *getfile*: Se conecta al usuario pasado como argumento utilizando la función connectToUser. Abre un nuevo archivo con el nombre del argumento pasado o abre el existente y procede a copiar los datos. Del mismo modo, la función de hilo seguirá enviando los datos en fragmentos hasta que alcance el final del archivo. Esto es una copia del proceso documentado en las diapositivas.

* Función de hilo

1.2 - Servidor

Utilizamos 5 funciones de ayuda: un manejador de señales, una función para enviar el código de resultado, una función para enviar una cadena completa utilizada para enviar los comandos de lista, una función de manejo de errores y una función para configurar el servidor.

Thread Function

Handle_client es la función de hilo que se asigna a cada usuario que se conecta; utilizamos la estructura ClientData para cargar los datos del cliente y pasárselos a la función de hilo. La data simplemente incluye el socket del cliente y un puntero a la tabla de usuarios que contiene todos nuestros datos de usuario. Después de cargar los datos en la función de hilo, recibimos todos los datos enviados por el cliente en un búfer. Dado que algunas operaciones requieren la dirección y el puerto del usuario, también cargamos esos. Luego comenzamos a leer desde el búfer. Dado que el orden de los datos enviados no cambia, siempre comienza con la operación, seguido por la fecha y hora, seguido por el nombre de usuario, y luego cualquier posible combinación de argumentos dependiendo del comando enviado, por lo que estos se analizan dentro de los bloques condicionales. Para cada comando, analizamos sus argumentos y llamamos a su función manejadora. Luego imprimimos la cadena de operación.

Estructura de Almacenamiento:

El método de almacenamiento de elección es una Tabla Hash. La estructura consiste en el tamaño de la tabla hash y una lista de punteros de Usuario. Cada usuario tiene un nombre de usuario, un estado de conexión, dos puertos, uno para conectarse al servidor y el otro cuando se mantiene esa conexión, una dirección. Finalmente, un usuario tiene una lista de archivos que simplemente es una lista enlazada de archivos.

Las operaciones básicas de la tabla hash incluyen: creación, liberación, hash e impresión. También se incluyen la inserción y la eliminación para insertar nuevos usuarios y eliminarlos. Debido a la constante necesidad de verificar si un usuario existe, también tenemos una función de encontrar usuario. En cuanto a los

archivos, podemos agregar archivos a un usuario específico en la tabla hash y también eliminarlos. Por la misma razón que los usuarios, también podemos verificar si un archivo existe.

Funciones Manejadoras:

Las funciones manejadoras son las funciones que operan en nuestra estructura de datos. Cualquier función que involucre a un usuario utilizará la función toLowerCase para evitar la sensibilidad al caso entre usuarios. Register verificará si un usuario existe con el nombre del argumento y, si no, insertará a ese usuario. Unregister hará lo mismo pero utilizará eliminar en lugar de insertar. Connect encontrará al usuario y verificará su estado de conexión en caso de que ya estén conectados. Los datos pasados a la función de conexión se utilizarán para poblar los atributos de la estructura del usuario: dirección, puerto y puerto de escucha, lo que significa que la conexión es exitosa. Handle disconnect hará lo mismo pero lo inverso para el estado de la conexión. Publish utilizará la función addFile para agregar el archivo al usuario que ha sido verificado como existente. Delete hará lo inverso, usando deleteFile. List users verificará que el usuario exista y esté conectado, luego recorrerá toda la lista de usuarios y concatenar a cada usuario conectado a una cadena junto con su dirección y puerto de escucha. Luego se devuelve esa lista y se establece el resultado. List content hace exactamente lo mismo pero en lugar de verificar a los usuarios conectados, verifica los archivos dentro de todos los usuarios. Finalmente, update users, que no es un comando accesible para el cliente, es un comando intermedio que establece el puerto de escucha de un usuario recién conectado. Es útil cuando un cliente se conecta, se desconecta y luego se vuelve a conectar, ya que asegura que los datos se actualicen correctamente.

1.3 - Servicio Web

Este código es una implementación simple de un servicio web de SOAP utilizando la biblioteca Spyne en Python. El servicio, denominado "TimeGetter", proporciona un método único "get_time" que devuelve la fecha y hora actuales en formato "dd/mm/YYYY HH:MM:SS". El servicio se expone como un servicio web de SOAP y se puede acceder a través de un servidor WSGI. El servicio se configura para utilizar el protocolo SOAP 1.1 y el validador XML lxml.

1.4 - RPC

La estructura PrintArgs se utiliza para encapsular los argumentos de la operación de impresión. Incluye campos para el tipo de operación, el nombre de usuario del usuario que inicia la operación, la fecha de la operación y el nombre de un archivo asociado con la operación para las operaciones de DELETE y PUBLISH. Cada uno de estos campos puede contener cadenas de hasta 255 caracteres.

El programa SERVER_OPERATION define la estructura y la versión para el servicio RPC. Dentro de este programa, se define el procedimiento print, que no devuelve ningún valor y depende únicamente de la estructura PrintArgs para la entrada.

La función del lado del servidor print_1_svc verifica el tipo de operación pasada en la struct. Si la operación es "PUBLISH" o "DELETE", la función imprime toda la información proporcionada, incluido el nombre del archivo. Si la operación es específicamente cualquier otra, el nombre del archivo se omite en la salida.

2 - Compiling

Para compilar y ejecutar la aplicación en el puerto de preferencia(en este ejemplo **8080**) los pasos son los siguientes:

1- Ejecutar comando ***make*** en la carpeta root del proyecto
2- Comenzar el servidor webservice con el comando: **python3 ws-time.py**
3- Iniciar servidor rpc con el comando: **./servidor_rpc**
4- Iniciar servidor principal con el comando: **./servidor 8080**.
5- Iniciar cliente de python con el comando: **python3 client.py -s localhost -p 8080**

Todos estos comandos deben ejecutarse en terminales diferentes.
Si se quiere ejecutar en otro puerto, se debe cambiar 8080 por el otro número de puerto escogido.

3 - Tests

Para probar nuestro servidor, hemos realizado diversas pruebas como las siguientes:

- Conectar más de 2 clientes a la vez y comprobar que todo funciona. Hemos conectado 3, 4 y 5 clientes y el servidor funcionaba correctamente. El sistema de publicación de archivos tenía una correcta ejecución, al igual que el get_file. Además, el RPC loguea bien las operaciones, haciendo la distinción entre delete y publish(donde añade el nombre del archivo) y las demás operaciones. En la siguiente imagen podemos ver una sesión con varios usuarios conectados:

```
c> list_users
c> LIST_USERS OK
d           127.0.0.1   51407
g           127.0.0.1   43929
a           127.0.0.1   40165
b           127.0.0.1   32869

c>
```

- Si intentamos realizar cualquier operación antes de conectarnos con un usuario, el resultado es el siguiente:

```
c> register b
c> REGISTER OK
c> list_users
c> LIST_USERS FAIL, USER NOT CONNECTED
c>
```

- Si intentamos registrarnos con un usuario ya registrado el resultado es el siguiente:

```
c> register a
c> USERNAME IN USE
c>
```

- Al intentar conectarnos como un usuario si ya hay otro usuario conectado en la misma terminal el resultado es un error. En cambio, si primero desconectamos el usuario y después nos intentamos conectar con el otro usuario, sí está permitido. Esto es el comportamiento esperado. Se puede ver en la siguiente imagen:

Estas son algunas de las pruebas que hemos realizado para probar nuestro servidor. El comportamiento ha sido el esperado y en los casos extremos que hemos pro

4-Conclusión

Con esta práctica hemos aprendido cómo sería crear una versión básica de un sistema de compartimiento de archivos peer to peer. Hemos utilizado muchos de los recursos en sistemas distribuidos aprendidos durante este curso, como RPC o sockets.

Algunos de los problemas más notorios que nos hemos encontrado ha sido cómo pasar los argumentos de las requests desde python a c. Al principio nuestra manera de hacerlo era una pero nos daba bastantes problemas y lo tuvimos que cambiar a la que finalmente tenemos.

Nos ha parecido interesante trabajar en esta práctica y creemos que nos puede ser bastante útil para un futuro. Aunque no usemos las tecnologías concretas que hemos usado aquí, las técnicas de programación distribuida en general nos pueden ser de mucha utilidad.