Luis Gómez-Manzanilla Nieto 100472006

Ignacio Fernández Cañedo 100471955

PROYECTO FINAL

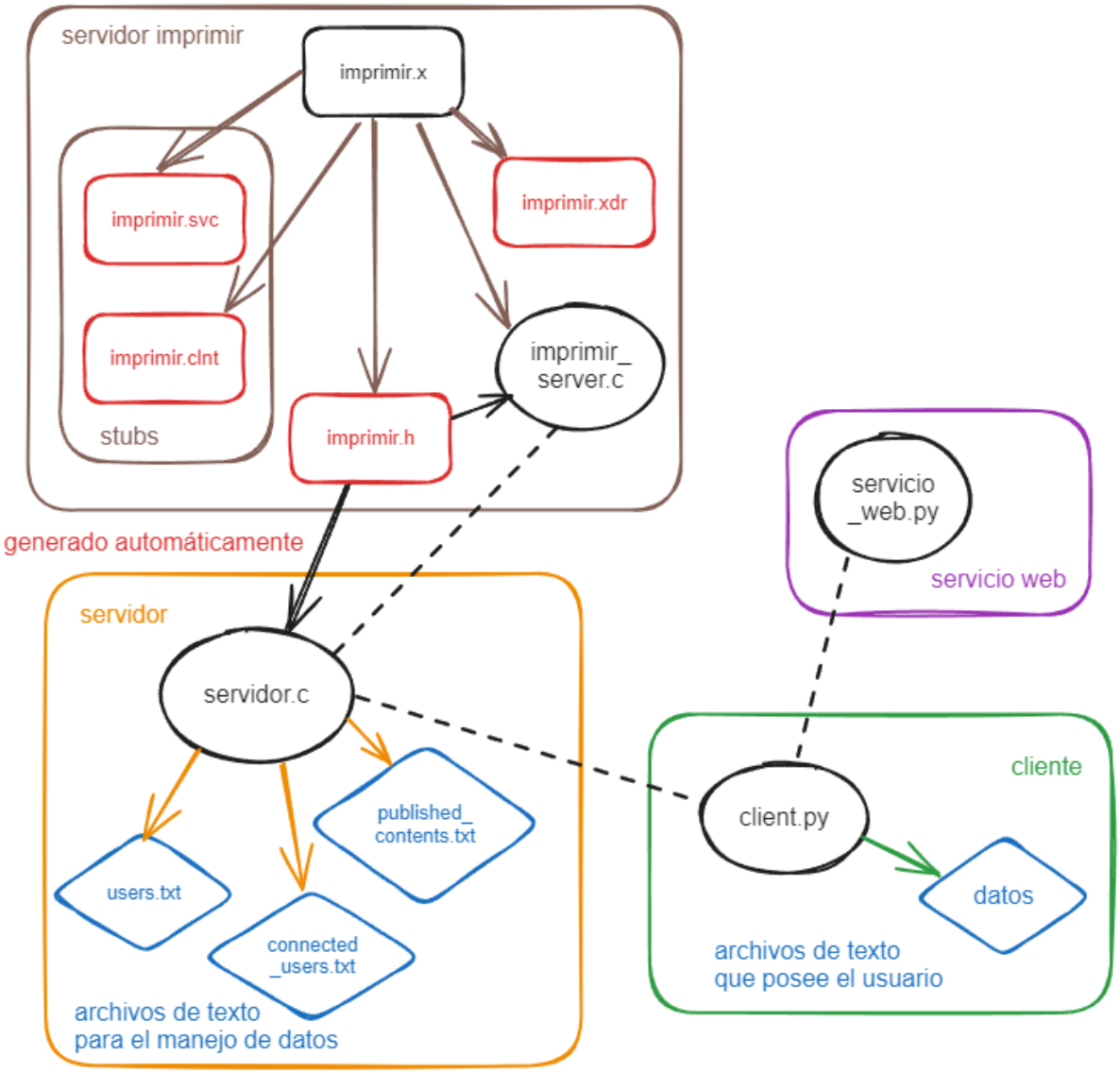
ÍNDICE

INTRODUCCIÓN

Este documento contiene la memoria del proyecto final de la asignatura Sistemas Distribuidos. Este proyecto trata de diseñar el servicio distribuido de distribución de ficheros entre clientes. Todo el proyecto está dividido en dos partes. Para la primera parte se utilizan sockets como método de comunicación cliente servidor. Para la segunda parte se ha implementado un servicio Web y otro rpc.

ARQUITECTURA COMPLETA

Mediante el siguiente gráfico mostramos la arquitectura completa realizada y las comunicaciones que se realizan entre archivos (líneas discontinuas):



En el gráfico anterior hemos querido hacer una clara distinción de los cuatro componentes principales del sistema mediante la encapsulación de estos.

Los dos componentes principales son el servidor (naranja) y el cliente (verde). Estos dos componentes se comunican usando sockets. El servidor tiene tres archivos que usa para el manejo de usuarios y archivos publicados. En users.txt guardamos los usuarios registrados, en connected_users.txt guardamos los usuarios que están conectados a la vez junto con su ip y puerto, y por último en published_contents.txt guardamos los archivos que han sido publicados junto con el usuario que lo publica y la descripción del fichero puesta por el usuario que lo subió. Por parte del cliente también tenemos los archivos publicados por el usuario.

El siguiente componente es el del servicio web (rosa) que solo contiene el archivo de python que implementa el servicio y es usado por el cliente (verde) para obtener la hora.

Por último, tenemos el componente del servidor RPC (marrón) que contiene la interfaz imprimir.x que usando el comando rpcgen produce los stubs, tanto el de cliente como el de servidor, el archivo de cabecera y el archivo para el manejo de datos, y por último la implementación del servidor que sirve como logger de las operaciones que se realizan en el servidor principal.

ESTRUCTURA DE LOS MENSAJES

Para pasar los mensajes, hay cuatro comunicaciones a destacar, la de los clientes con el servidor, la de los clientes entre ellos, la del servidor con el servidor RPC y la del cliente con el servicio Web. El cliente envía al servidor una cadena con el código de operación, la fecha en la que se ha producido la operación y el usuario que ha realizado dicha operación. Como todo se pasa en una única cadena, separamos las distintas partes del mensaje y las extraemos utilizando *strtok*.

Las operaciones que pueden realizar los clientes son *“REGISTER”, “UNREGISTER”, “CONNECT”, “DISCONNECT”, “PUBLISH”, “DELETE”, “LIST_USERS”, “LIST_CONTENT” y “GET_FILE”.* Para cada una de las operaciones, el cliente envía información distinta a través del socket. La fecha y la hora se envía siempre independientemente de la operación realizada. Las operaciones

“REGISTER”, “UNREGISTER” y “DISCONNECT” se envía únicamente al usuario sobre el que se va a realizar esta operación. Para *“CONNECT”* además de dicho usuario, también se envía la ip y puerto de escucha. La función *“PUBLISH”,* a diferencia del resto, envía al usuario que está realizando esta operación, ya que al hacer publish ya hay un usuario conectado. Además de esto, se envía el archivo que se desea publicar y la descripción de este. Para la operación *“DELETE”,* es igual que la anterior pero no se le manda la descripción, ya que con el nombre ya se puede eliminar un archivo. Las operaciones *“LIST_USERS” y “LIST_CONTENT”* envían al usuario que está realizando la función, pero además en *“LIST_CONTENT”,* que se utiliza para listar todo el contenido de un usuario, también se envía al usuario sobre el que se va a realizar la operación. Por último, en *“GET_FILE”* se envía a ambos usuarios, y el archivo remoto que se quiere obtener. Esta operación se utiliza para obtener un archivo de otro usuario. Posteriormente entre los clientes se envían los archivos.

En cada caso, el servidor responde con un código de estado que indica si la operación fue exitosa o no. Si la operación fue exitosa, el cliente realiza acciones adicionales, como imprimir un mensaje de éxito, crear un archivo o recibir datos del servidor. Si la operación no fue exitosa, el cliente imprime un mensaje de error.

En el caso del servicio Web, a este le llama el cliente cuando se realiza cada operación. Se realiza una llamada a la función time_sv del servicio que no recibe nada, pero devuelve una cadena en el formato *'%d/%m/%Y %H:%M:%S'* donde d,m,y corresponden a los días, meses y años y H, M, S a las horas, minutos y segundos en los que se ha llamado a esta función. Cuando el cliente ejecuta una operación y le envía los datos correspondientes al servidor, también le envía el resultado de time_sv, como hemos explicado antes. Cuando el servidor desconcatena el mensaje, obtiene la fecha y la hora por separado, que se lo enviará al servidor RPC junto a la operación y el usuario que la realiza. Posteriormente este servidor RPC se encarga de imprimir toda la información en el formato *usuario OPERACION fecha hora.*

PROTOCOLOS DE APLICACIONES

Para la comunicación cliente servidor el protocolo de aplicación utilizado es sockets TCP. Sockets TCP también se utiliza en la comunicación entre los

distintos clientes. TCP proporciona una comunicación fiable entre el cliente y el servidor. También hemos definido el protocolo de aplicación que se utiliza para enviar los mensajes, incluyendo la operación, los parámetros y el formato de respuesta del servidor. También se ha implementado el servicio Web en comunicación con el cliente y la comunicación RPC entre el servidor y el servidor RPC.

OPERACIONES DE COMPONENTE

Como se ha mencionado antes, los clientes pueden realizar distintas operaciones. Dependiendo de la operación, el servidor realizará una cosa u otra. A continuación se detalla cómo trabaja el servidor con las distintas operaciones:

register_user: Esta función registra un nuevo usuario en el sistema. Primero, verifica si el usuario ya está registrado comprobando el archivo “users.txt”. Si el usuario ya está registrado, envía ‘1’ al cliente y termina. Si el usuario no está registrado, añade el nombre del usuario al archivo “users.txt”, crea una carpeta para los archivos que este usuario publique, envía ‘0’ al cliente y termina.

unregister_user: Esta función elimina un usuario del sistema. Primero, verifica si el usuario está registrado comprobando el archivo “users.txt”. Si el usuario no está registrado, envía ‘1’ al cliente y termina. Si el usuario está registrado, elimina el nombre del usuario del archivo “users.txt”, además de sus archivos publicados, envía ‘0’ al cliente y termina.

connect_user: Esta función conecta a un usuario al sistema. Primero, verifica si el usuario está registrado comprobando el archivo “users.txt”. Si el usuario no está registrado, envía ‘1’ al cliente y termina. Luego, verifica si el usuario ya está conectado comprobando el archivo “connected_users.txt”. Si el usuario ya está conectado, envía ‘2’ al cliente y termina. Si el usuario no está conectado, añade el nombre del usuario, la dirección IP y el puerto al archivo “connected_users.txt”, envía ‘0’ al cliente y termina. El cliente guardará en una variable el usuario que está conectado en el momento.

publish_content: Esta función permite a un usuario publicar contenido. Primero, verifica si el usuario está registrado y conectado. Si el usuario no está registrado o no está conectado, envía el código de error ‘1’ o ‘2’ respectivamente. Luego, verifica si el contenido ya está publicado comprobando el archivo “published_contents.txt”. Si el contenido ya está publicado, envía ‘3’ al cliente y termina. Si el contenido no está publicado, añade el nombre del usuario, el nombre del archivo y la descripción al archivo “published_contents.txt”. También crea el archivo en la carpeta de los usuarios. Envía ‘0’ al cliente y termina.

delete_content: Esta función elimina un contenido publicado por un usuario. Primero, verifica si el usuario está registrado y conectado. Si el usuario no está registrado o no está conectado, envía el código de error ‘1’ o ‘2’ respectivamente Si el usuario no está registrado o no está conectado, envía el código de error ‘1’ o ‘2’ respectivamente.. Luego, verifica si el contenido está publicado por el usuario. Si el contenido está publicado, lo elimina del archivo “published_contents.txt” y de la carpeta del usuario y envía ‘0’ al cliente. Si el contenido no está publicado, envía ‘3’ al cliente.

list_users: Esta función lista todos los usuarios conectados. Primero, verifica si el usuario que solicita la lista está registrado y conectado. Si no es así Si el usuario no está registrado o no está conectado, envía el código de error ‘1’ o ‘2’ respectivamente. Luego, lee el archivo “connected_users.txt”, cuenta el número de usuarios conectados y envía esta información al cliente en un mensaje. El mensaje tiene la estructura “0{num_users}{user1}{user2}…{userN}”, donde el ‘0’ representa el código, {num_users} es el número de usuarios conectados y {user1}, {user2}, …, {userN} son los nombres de los usuarios conectados. El código del cliente es el que se encarga de desconcatenar este mensaje e imprimirlo en el formato adecuado.

list_content: Esta función lista todos los archivos publicados por un usuario objetivo. Primero, verifica si el usuario que solicita la lista está registrado y conectado. Si no es así envía los códigoss ‘1’ o ‘2’. Luego, lee el archivo “published_contents.txt” y envía esta información al cliente en un mensaje. El mensaje tiene la estructura “0{file1}{file2}…{fileN}”, donde ‘0’ es el código correcto y {file1},

{file2}, …, {fileN} son los nombres de los archivos publicados por el usuario objetivo. El cliente al igual que en la función anterior, se encarga de desconcatenar este mensaje e imprimirlo adecuadamente.

disconnect_user: Esta función desconecta a un usuario del sistema. Primero, verifica si el usuario está conectado comprobando el archivo "connected_users.txt". Si el usuario está conectado, lo elimina del archivo "connected_users.txt" y envía '0' al cliente. Si el usuario no está conectado, envía '2' al cliente. Si el usuario no existe enviará un '1'.

get_file: Esta función permite a un usuario obtener un archivo publicado por otro usuario. Primero, verifica si el usuario que solicita el archivo está registrado. Si no es así enviará un '1'. Luego, verifica si el archivo está publicado por el usuario. Si el archivo está publicado, verifica si el usuario está conectado y envía '0' al cliente junto con la dirección IP y el puerto del usuario del que se quiere obtener sus archivos. Si el archivo no está publicado o el usuario no está conectado, envía el '2' al cliente.

En el caso del cliente, este es el que envía al servidor la operación junto a los datos correspondientes. En todas las funciones recibe un código de operación del servidor que indica si la operación se realizó de forma exitosa o no. También el cliente imprime un mensaje dependiendo del código que ha recibido.

Para la función **connect**, el cliente crea un socket y un hilo para tratar las peticiones que le lleguen desde distintos clientes cuando realicen la operación get_file. Este hilo maneja la función handle_request que espera por una conexión de otro cliente y se encarga de leer el contenido del archivo del usuario que se ha solicitado, para posteriormente enviarlo al cliente y que se cree un archivo con dicho contenido. Esta es la manera en la que funciona get_file.

La función **disconnect** modifica la variable de bloqueo del hilo para que este salga del bucle en handle request y se pueda terminar la conexión. Además también cierra el socket de escucha.

La función **get_file** se encarga de crear una conexión con otro cliente. Como hemos explicado antes, al hacer connect ese cliente se queda esperando una petición de otro cliente, esta petición llega cuando ese otro cliente realiza la operación get_file. Le manda por el socket el nombre del archivo

que quiere obtener y recibe la información de este, que posteriormente guardará en su nuevo archivo personal.

Un cliente también puede escribir **QUIT** en la shell. Este comando llamará a disconnect para desconectar a un usuario si está conectado y posteriormente termina la ejecución imprimiendo '+++ FINISHED +++'.

La tarea del servicio web es únicamente dar la hora cuando se llama a la función time_sv. Esto se realiza así para que cuando el cliente ejecute una operación, el servidor enviará al servidor RPC los datos para que éste imprima por pantalla al usuario que realiza la operación , la operación que realiza y la fecha y hora a la que lo está realizando.

COMPILACIÓN Y EJECUCIÓN

Las instrucciones de compìlación y ejecución del proyecto también se incluyen en el fichero README incluido con el resto de archivos, son las siguientes:

Para compilar el código en C ejecutamos el siguiente comando:

make -f Makefile.imprimir

Para iniciar el servidor RPC ejecutamos los dos siguientes comandos:

sudo mkdir -p /run/sendsigs.omit.d/

sudo /etc/init.d/rpcbind restart

Las siguientes ejecuciones se tienen que hacer en terminales distintas.

Para ejecutar el servicio web en python ejecutamos el siguiente comando: *python3 servicio_web.py & python3 -mzeep http://localhost:8000/?wsdl*

Para ejecutar el servidor RPC: *./imprimir_server*

Para ejecutar el servidor, primero se debe declarar una variable de entorno con la ip del servidor rpc: *export IMPRIMIR_SERVER=localhost*

Para ejecutar el servidor: *./servidor -p 8080*

Para ejecutar el cliente, se puede ejecutar varios clientes en terminales diferentes:

python3 ./client.py -s localhost -p 8080

Cabe destacar que para simplificar el manejo de los usuarios registrados y los archivos que se publican, el servidor crea un directorio por usuario registrado y los archivos que se publican o se suben deben localizarse en dichos directorios, por lo que al ejecutar el código se irán creando directorios o archivos.

PRUEBAS

Para las pruebas, éstas se realizan con el servidor RPC, el servidor y el servicio Web en ejecución, ya que no hay que realizar ninguna acción por el usuario en estos sistemas y por lo tanto no requieren ninguna prueba.

Por lo tanto, cuando se estén ejecutando ambos servidores y el servicio Web, las pruebas se realizarán con los clientes:

1ª prueba - Funcionamiento correcto

La primera prueba consistirá en realizar una ejecución normal. Un usuario se registrará, se conectará y posteriormente intentará obtener el archivo de otro usuario. Para ello, se debe ejecutar lo siguiente en la shell del cliente (T1 significa la terminal 1 y T2 la terminal 2):

T1.1- REGISTER user1

T1.2- CONNECT user1

Ahora abrimos otra terminal con otro usuario en la que publicaremos dos archivos para que el user1 pueda obtener uno de ellos

T2.1- REGISTER user2

T2.2- CONNECT user2

T2.3- PUBLISH prueba.txt archivo de prueba 1

T2.4- PUBLISH prueba2.txt archivo de prueba 2

En el archivo prueba.txt se puede escribir algún contenido para ver como posteriormente el otro usuario tendrá en su archivo dicho contenido.

Ahora en la terminal del primer usuario obtendremos los usuarios conectados y el contenido publicado de user2

T1.3- LIST_USERS

user1 127.0.0.1 38357

user2 127.0.0.1 52039

T1.4- LIST_CONTENT user2

prueba.txt "archivo de prueba1"

prueba2.txt "archivo de prueba2 "

T1.5- GET_FILE user2 prueba.txt miprueba.txt

Como podemos ver, ahora el user1 en su carpeta personal tiene el archivo miprueba.txt con el contenido de prueba.txt del user2.

Ahora el user2 eliminará el archivo.

T2.5- DELETE prueba.txt

T2.6- LIST_CONTENT user2

prueba2.txt "archivo de prueba2 "

Se puede ver que ya no existe el archivo que se ha eliminado

Ahora se pueden desconectar ambos usuarios

T1.6- DISCONNECT user1

T2.7- DISCONNECT user2

Ambos usuarios desaparecen del archivo connected_users y al hacer list_users ya no aparecen

Ahora realizaremos unregister de alguno de ellos para que desaparezca de users.txt

T2.8- UNREGISTER user2

Se puede observar que ahora user2 no está en el fichero de users.txt y su carpeta personal se ha eliminado, mientras que user1 si está y tiene en su carpeta personal el archivo que ha obtenido del otro usuario.

Abriendo la terminal RPC, podemos ver que han ido apareciendo las operaciones realizadas por ambos usuarios junto al tiempo en el que lo han realizado. El servidor también contiene salidas en la terminal en la que podemos ver dichas operaciones y como la información se ha ido pasando del cliente al servidor y viceversa.

Con esta prueba verificamos que todas las operaciones funcionan correctamente, así como el resto de archivos y protocolos implementados.

2ª prueba - Usuarios no existen

En esta prueba se va a comprobar que el código trata correctamente el manejo de los errores cuando se realizan operaciones que no deberían

En primer lugar, vamos a intentar conectar a un usuario que no existe

T1.1- CONNECT prueba2

c > CONNECT FAIL, USER DOES NOT EXIST

Como podemos ver, se ha tratado bien el error y la shell permite volver a introducir un comando.

Ahora vamos a verificar qué ocurre si hacemos UNREGISTER de un usuario que no existe

T1.2- UNREGISTER prueba2

c > USER DOES NOT EXIST

Se puede observar que también se trata bien el error

Ahora intentaremos hacer publish de un usuario que no se ha conectado y no existe

T1.3- PUBLISH prueba.txt archivo de prueba

c > PUBLISH FAIL, USER DOES NOT EXIST

También trata bien el error ya que no hay ningún usuario conectado y por lo tanto no existe usuario para hacer publish

Ahora comprobaremos list_user y LIST_CONTENT

T1.4- list_users

c > LIST_USERS FAIL, USER DOES NOT EXIST

T1.4- list_content prueba2

c > LIST_CONTENT FAIL, USER DOES NOT EXIST

En ambas operaciones se realiza bien el tratamiento de errores, ya que no hay ningún usuario conectado y no se puede saber quien ha realizado la operación

Por último comprobaremos GET_FILE

T1.6- GET_FILE prueba2 prueba.txt prueba2.txt

c > GET FILE FAIL, FILE NOT EXIST

T1.7- DISCONNECT prueba2

c > DISCONNECT FAIL, USER DOES NOT EXIST

Como nuestra función get_file primero comprueba el archivo, ese es el error que muestra, ya que no existe dicho archivo. Disconnect muestra que el usuario no existe.

Podemos ver que el manejo de errores en el caso en el que los usuarios no existen funciona correctamente

3ª prueba - Usuario existe pero no está conectado

En esta prueba comprobaremos distintos tipos de errores que pueden ocurrir en el código cuando el usuario sí existe pero no está conectado.

En primer lugar registramos a un usuario

T1.1- REGISTER prueba3

Ahora registramos a otro usuario con ese mismo nombre

T1.2- REGISTER prueba3

c > USERNAME IN USE

Podemos ver que se ha tratado bien el error y en el archivo users.txt no se ha guardado otro user.

Ahora conectaremos al usuario

T1.3- CONNECT prueba3

Se ha conectado correctamente, vamos a intentar conectarlo de nuevo

T1.4- CONNECT prueba3

c > USER ALREADY CONNECTED

Se ha mostrado bien el mensaje y no se ha guardado en connected_users otra vez. Ahora vamos a desconectar a este usuario y realizar el resto de operaciones con un usuario que si está registrado pero no conectado

T1.5- DISCONNECT prueba3

T1.6- DISCONNECT prueba3

c > DISCONNECT FAIL, USER NOT CONNECTED

Vemos que si se trata la opción cuando el usuario no está conectado. Ahora realizaremos el resto de funciones

T1.7- PUBLISH prueba.txt archivo de prueba

c > PUBLISH FAIL, USER DOES NOT EXIST

T1.8- DELETE prueba.txt

c > DELETE FAIL, USER DOES NOT EXIST

T1.9- list_users

c > LIST_USERS FAIL, USER DOES NOT EXIST

T1.10- list_content prueba2

c > LIST_CONTENT FAIL, USER DOES NOT EXIST

T1.11- GET_FILE prueba2 prueba.txt prueba2.txt

c > GET FILE FAIL

Como podemos ver, el mensaje de error que se muestra es el de que el usuario no existe, en lugar de no conectado. Esto se debe a que la forma en la que hacemos nosotros la comunicación, el servidor recibe el usuario cuando este se conecta, por ello, como estas operaciones no reciben ningún usuario, el servidor no sabe quien és el usuario que las realiza porque no sabe quién de todos los usuarios registrados es el que la está realizando, ya que no se ha conectado.

4ª prueba - El resto de mensajes de error

A continuación probaremos el resto de casos de error que tienen que tratar las distintas operaciones. Para ello vamos a registrar a un usuario y conectarlo

T1.1- REGISTER prueba4

T2.2- CONNECT prueba4

Para la operacion publish, vamos a publicar dos veces lo mismo

T1.3- PUBLISH 1.txt contenido 1

T1.4- PUBLISH 1.txt contenido 1

c > PUBLISH FAIL, CONTENT ALREADY PUBLISHED

Como podemos ver, no permite publucar dos veces el mismo archivo. Ahora si borramos un archivo que no se ha publicado.

T1.5- DELETE 2.txt

c > DELETE FAIL, CONTENT NOT PUBLISHED

No elimina nada ya que ese archivo no se ha publicado. Ahora intentaremos hacer un list_content de un usuario que no existe

T1.6- LIST_CONTENT usuarionoexiste

c > LIST_CONTENT FAIL, REMOTE USER DOES NOT EXIST

Verifica si el usuario remoto existe y si no es así muestra ese mensaje. En el caso de get_file, mostrará el mensaje GET_FILE FAIL siempre que el resultado no sea el esperado o que el servidor devuelva un 1. Si se intenta conectar un usuario en una terminal en la que ya hay un usuario conectado,

se imprimirá USER ALREADY CONNECTED y no lo permitirá. Para conectar a otro usuario se deberá hacer desde otra terminal u otro dispositivo.

CONSIDERACIONES

Al hacer un list_content de un usuario que ha hecho get_file de otro cliente, no se mostrará este archivo, ya que no ha sido él quien ha publicado el archivo, sino el otro cliente. Esto lo hemos decidido así ya que en la memoria indica que con list_content se muestran solo los archivos publicados por el usuario.

Para simular el funcionamiento de un servidor real y poder visualizarlo, hemos implementado que por cada usuario que se registra, se crea un directorio asociado a él y es donde se localizan sus archivos.

La función get_file requiere incluir un usuario que esté conectado para poder obtener alguno de sus archivos. No se podrá realizar esta función de un usuario que no esté conectado aunque tenga archivos publicados. Esto lo hemos entendido así del enunciado de la práctica.

Como hemos indicado anteriormente en el apartado de las pruebas, de la forma en la que está implementado nuestro sistema, las operaciones que no reciben ningún usuario utilizan al usuario que se ha conectado porque al conectarse este se almacena como una variable de entorno. Es por esto que para los casos en los que el usuario se ha registrado pero no está conectado, el mensaje que se mostrará siempre es el de USER DOES NOT EXIST en lugar de USER NOT CONNECTED. Esto es debido a que al no realizar un connect el sistema no sabe cual de todos los usuarios que hay registrados es el que está conectado, ya que se podría registrar a varios usuarios en el sistema en una ejecución de la shell.

CONCLUSIÓN

En conclusión, en este proyecto hemos creado un sistema distribuido de gestión y distribución de archivos entre usuarios. El sistema permite que distintos usuarios puedan hacer varias operaciones distintas, como conocer los archivos de otro usuario u obtener alguno de ellos. Cada operación maneja los distintos errores y casos que pueden ocurrir. Además el sistema

lleva un registro de las operaciones que se realizan y la hora a la que se realizan en un servidor RPC aparte. Esta práctica nos ha ayudado a ampliar nuestros conocimientos de algunos aspectos de la asignatura, como la programación de sockets tanto en Python como en C, la gestión de múltiples conexiones a través de hilos, o la implementación de distintos protocolos como RPC o el servicio Web. Por último, esta práctica nos ha hecho ver que un sistema distribuido puede hacer uso de diferentes métodos de comunicación, aprovechando las ventajas de cada uno y enfocándose en propósitos determinados, y no solo tiene porqué reducirse a usar solo un tipo de comunicación.