Universidad Carlos III
Curso Sistemas Distribuidos 2023-24
Práctica final

Rubén de Arriba Viejo (100471975) 100471975@alumnos.uc3m.es
Óscar Hontoria Herrador (100471920) 100471920@alumnos.uc3m.es

Índice

1. Introducción

Este documento presenta los razonamientos y pasos seguidos para el diseño e implementación del servicio distribuido para la gestión de archivos entre usuarios. En este sistema, uno o más clientes mediante el uso de SOCKETS, podrán acceder al servidor que almacena los usuarios conectados y permite que estos publiquen ficheros, vean los otros usuarios y realizen distintas operaciones sobre los usuario o una operación entre dos usuarios.

Para hacerlo se ha implementado un servidor para la gestión de los ficheros y usuarios, un cliente para solicitar operaciones y un servicio web.

2. Componentes

En esta práctica se han generado cuatro componentes. Cada uno de ellos cumple una función fundamental para el funcionamiento global del sistema. Los componentes son los siguientes:

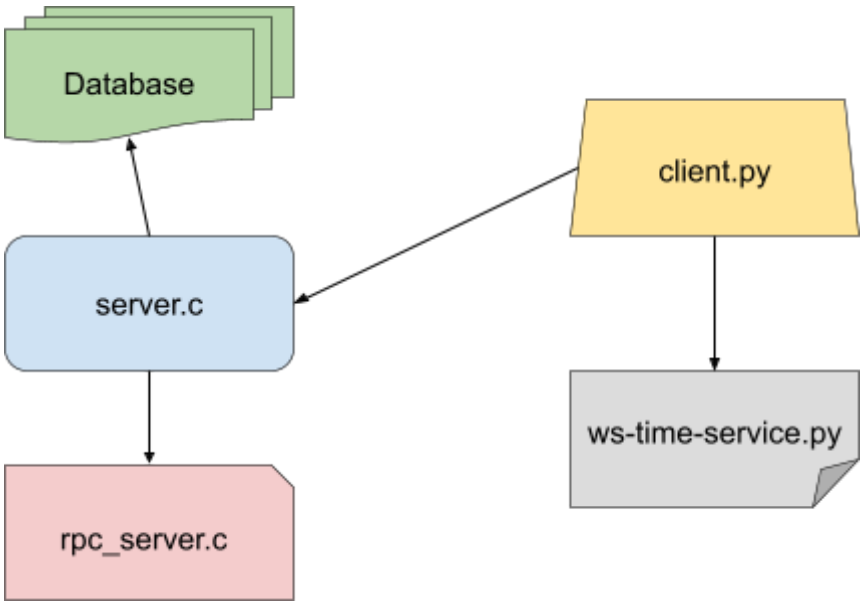
- Server.c: este fichero contiene el código correspondiente al servidor principal del sistema. Se encarga de almacenar los datos proporcionados por los clientes en la base de datos y, mediante RPC, realizar funciones definidas en rpc_server.c.

- Client.py: este fichero contiene el código correspondiente a la parte cliente del sistema. Se encarga de recoger los datos de entrada, transformarlos de forma correcta y enviarlos al servidor. Los clientes también se conectan a un servicio web.

- Ws-time-service.py: este fichero contiene el código correspondiente a un servicio web que proporciona a quien lo solicite la hora actual. Este servicio es utilizado por los clientes.

- Rpc_server.c: este fichero contiene la función que ejecuta el fichero server.c mediante el uso de RPC.

La arquitectura final quedaría de la siguiente manera:



3. Comunicaciones

Los clientes llaman a la función del servicio web para obtener la hora que proporcione este servicio. De esta operación, el cliente recibe la fecha y hora a la que realizó la solicitud, el mensaje se envía en forma de cadena de caracteres.

Para realizar las operaciones con el servidor, el cliente genera una cadena de caracteres con toda la información correspondiente a la operación. Esta se envía al servidor mediante el uso de sockets. Al ser cadenas de caracteres, no es necesario hacer ninguna transformación a la hora de enviar ni recibir la información.

La comunicación entre server.c y rpc_server.c se realiza mediante RPC, por lo que hemos generado la estructura *usuario_peticion* que tiene cuatro campos diferentes: usuario, operacion, fecha y file_name. Esta estructura almacena la información que deberá imprimir rpc_server.

El almacenamiento se realiza de forma dinámica por el fichero server.c, almacenando la información en listas enlazadas. Esto implica que, si el servidor deja de estar activo, se pierda la información almacenada durante su ejecución.

Para la compartición de ficheros entre usuarios, los clientes, al ejecutar la función correspondiente, reciben del servidor la dirección IP y el puerto al que deben dirigirse para poder recibir el fichero solicitado. Una vez conozcan esta información, deberán realizar otro mensaje para enviarlo a dicha ubicación, donde se encontrará el cliente que tiene el fichero. De esta forma, el segundo cliente envía al primero el fichero solicitado como cadena de caracteres de la misma forma, utilizando sockets.

4. Justificación

Respecto al paso de mensajes se ha elegido el envío de cadenas de caracteres porque, a parte de que la mayor parte de los parámetros enviados son caracteres, no hace falta hacer ninguna transformación de little-endian a big-endian o viceversa, ya que al ser bytes se representan de la misma forma independientemente de la arquitectura.

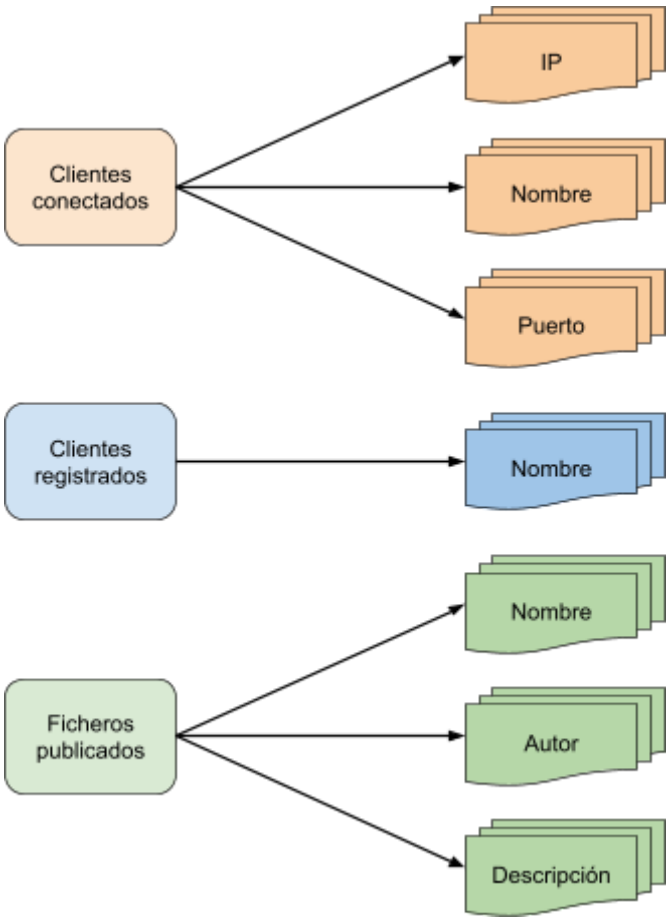
En el caso de RPC se ha elegido utilizar una estructura porque era compatible entre los ficheros que utilizan esta herramienta, es decir, ambos lenguajes son c. Además, internamente, RPC serializa las estructuras, por lo que es un trabajo que no hace falta que realicemos nosotros.

En el caso del servicio web, al ser únicamente una función que retorna un valor y ser lenguaje python, hemos decidido simplemente llamar a dicha función y almacenar el resultado en una variable que posteriormente se serializará con el resto de componentes del mensaje para enviarlo al servidor.

En cuanto al almacenamiento de datos, el uso de listas enlazadas nos resulta muy sencillo a la hora de controlar el número de elementos dentro de la lista como a la hora de acceder a los datos dentro de las mismas. Se han creado 7 listas enlazadas diferentes: para los clientes registrados, para los clientes conectados, para las IP de los clientes conectados, para los sockets de los clientes conectados, para los nombres de los ficheros publicados, para las descripciones de los fichero publicados y para los autores de los ficheros publicados. Estas

siete listas enlazadas pueden agruparse en tres, ya que se acceden de forma paralela siempre. Estas tres agrupaciones son: los clientes conectados con las IP de los clientes conectados y con los socket de los clientes conectados, el nombre de los ficheros publicados con las descripciones y autores de los mismos y los clientes registrados.

Se ha decidido de esta forma para ahorrarnos el tener que trabajar con un puntero extra de las estructuras que podrían haberse generado. La estructura de la base de datos quedaría:



5. Operaciones

Las operaciones se realizan en dos partes principales: la creación del mensaje y la interpretación y ejecución correspondiente.

La creación del mensaje se realiza en el cliente (client.py), que recoge los datos de entrada y los ordena adecuadamente para enviarlos al servidor. Durante esta operación, el cliente accede al servicio web (ws-time-service.py), que proporciona la fecha y hora al cliente. Esto se hace para que los clientes reciban la información de fecha y hora de un único sitio y en el servidor se puedan ordenar adecuadamente las operaciones.

Una vez enviado el mensaje, el servidor (server.c) deserializa la información del mensaje y decide qué operación realizar. Al tener los datos correctamente deserializados, envía una parte de la información al servidor de RPC (rpc_server.c) para que este lo muestre por pantalla.

Dentro del propio servidor (server.c) se realizan todas las operaciones que tienen relación con el almacenamiento de datos y gestión de usuarios. Como todos los datos se guardan en forma

de cadenas de caracteres, se realizan operaciones malloc y realloc dependiendo de la operación solicitada. También, dependiendo de la operación, se deben comprobar distintos aspectos del usuario, por ejemplo que no se registre si ya está registrado o que no se conecte si ya está conectado.

Los ficheros que se generan en la operación GET_FILE se guardan en la carpeta llamada ficheros.

6. Guía de ejecución

Para la ejecución de los programas primero es necesario ejecutar el Makefile con la operación *make* en terminal. Tras ejecutar el Makefile se crearán dos ejecutables diferentes: *server* y *rpc_server*.

Para ejecutar correctamente cada una de las partes es necesario el uso de 4 terminales: una para ejecutar *web-time-service.py*, otra para ejecutar *client.py*, otra para ejecutar *server* y otra para ejecutar *rpc_server*. Para la ejecución de *client.py* es necesario dar como parámetros la IP y el puerto de escucha del servidor. Este puerto se selecciona al ejecutar *server* ya que se debe ejecutar pasando como parámetro el puerto que se desea utilizar. Los otros dos ficheros se ejecutan sin pasar ningún tipo de parámetro de entrada.

La ejecución quedaría de la siguiente manera:

- Terminal 1: python3 web-time-service.py
- Terminal 2: python3 client.py -s [IP_SERVER] -p [PORT_SERVER]
- Terminal 3: ./server [PORT_SERVER]
- Terminal 4: ./rpc_server

El fichero web-time-service.py debe ejecutarse el primero para evitar errores.

En cuanto a la parte de RPC, es posible eliminar todos los ficheros que comiencen de la forma rpc_conexion a excepción del fichero rpc_conexion.x y volver a generarlos. Se generarían los ficheros de la forma *rpcgen -C rpc_conexion.x*.

7. Pruebas

Para organizar mejor la memoria se ha decidido poner las pruebas de ejecución en otro documento pdf. Este documento tiene el nombre de *pruebas.pdf*.

8. Consideraciones adicionales

La ejecución de RPC tarda alrededor de 10 segundos por cada ejecución. Debido a esto, todas las pruebas se han realizado sin utilizar este servicio, salvo una en la que se demuestra su funcionamiento. Desconocemos el origen de este retraso a pesar de haber optimizado el código. Además, este error no ha ocurrido de forma continua, algunas veces aparecía y otras no.

9. Conclusiones

Consideramos que este trabajo sirve como ejemplo de la realización de un sistema distribuido completo, con diferentes componentes que interactúan entre ellos. Además, consideramos muy adecuado que varias formas de comunicación se utilicen en distintas partes de la arquitectura, puesto que así podemos encontrar y solucionar problemas que anteriormente no nos habíamos encontrado.