Universidad Carlos III

Curso 2024-25

Sistemas Distribuidos

Proyecto Final

Marcos Romo Poveda, 100496072
100496072@alumnos.uc3m.es,
Luca Petidier Iglesias , 100496633,
100496633@alumnos.uc3m.es

Índice de Contenidos:

Introducción

En este documento se pretende explicar cómo se ha desarrollado el proyecto final de la asignatura de 'Sistemas Distribuidos' consistente en diseñar un sistema peer-to-peer (P2P) que permita el traspaso de ficheros entre clientes.

Esta práctica se ha divido en dos bloques:

A. Parte 1: En esta primera parte usando sockets TCP se van a implementar un total de 9 funcionalidades que permitirán todo el proceso de la comunicación y traspaso de ficheros entre clientes.

B. Partes 2 y 3: En estas partes se ha creado un servidor rpc, que imprime las operaciones realizadas en el servidor de la Parte 1, así como un servicio web en python al que el cliente python de la Parte 1 solicita la fecha, datos que se envían por el servidor de sockets hasta el de rpc.

Con todo esto tendremos un total de 3 servidores (Uno para cada parte de la práctica) y tantos clientes como queramos entre los que se producirá la comunicación.

Descripción del código, Parte 1

A continuación explicaremos las 9 funcionalidades implementadas para la Parte 1 de nuestro proyecto así como una explicación general del funcionamiento del servidor principal:

1. REGISTER

Si 'client.py' recibe como línea de comandos 'REGISTER user' envía al servidor un primer mensaje 'REGISTER' y después un segundo mensaje 'user'.

El servidor tras recibir el primer mensaje ya sabe que función debe de ejecutar por lo que envía a la función 'register_user' el segundo mensaje recibido correspondiente con el nombre del usuario que queremos registrar.

Esa función se encarga de crear la carpeta './Users/' en caso de que no exista, de comprobar que el usuario no estuviera previamente registrado y de registrarlo, en caso de que haya ido todo bien devuelve 0 y en caso contrario devolverá otro número.

Finalmente el servidor le devuelve al cliente el valor recibido de la función y en base a ese valor el cliente imprimirá un determinado mensaje.

2. UNREGISTER

Esta función es similar a la anterior con el añadido de que en caso de que el cliente estuviese conectado, se cierra su socket correspondiente y se devuelven al valor 'None' las variables del cliente que controla quién está conectado.

3. CONNECT

En caso de que queremos conectar el cliente ocurrirán dos cosas:

1. El servidor recibirá un nombre y un puerto, tras comprobar que el usuario está registrado pero no conectado, modifica el archivo de registro y añade una dos líneas que guardarán la IP que conoce el servidor del cliente que se le conecta y el puerto recibido como parámetro.

2. En la parte del cliente, creará un servidor (socket) en un thread, enviará el puerto asociado al servidor, si ha ido todo bien lo iniciará, modificará las variables locales que indican el usuario que está conectado y se quedará en modo 'background' esperando a recibir algún mensaje.

4. DISCONNECT

En esta función es el opuesto a la función anterior.

El cliente enviará el usuario que quiere desconectar y si ha ido todo bien y el usuario estaba conectado y registrado, se eliminan los datos del archivo de registro de la parte del servidor y en el cliente se cierra el servidor (socket) y las variables de control del usuario conectado y su socket se resetean al valor 'None'.

Cabe destacar que siguiendo las instrucciones del profesorado, sólo puede haber un cliente conectado por 'equipo / sesión' y sólo puedes intentar desconectar un usuario que no lo esté o el tuyo propio, no deberás desconectar un usuario conectado desde otro 'equipo / sesión', lo mismo ocurrirá con la función 'connect', no podrás hacer connect si tu equipo ya está previamente conectado.

5. PUBLISH

En esta función es una que tiene un mayor nivel de complejidad que la anterior.

Desde el cliente se enviará el comando 'PUBLISH', el nombre del cliente, una ruta completa de un archivo y una descripción de dicho archivo. Desde la parte del cliente, tras comprobar que el usuario está registrado, se comprueba que no exista previamente un archivo con la ruta pasada por argumento, en caso de que no exista, se crea y se guarda en dicho archivo la ruta y la descripción del archivo.

Los archivos son nombrados como '1.txt', '2.txt'... de forma consecutiva y se creará un archivo de número mayor al archivo con mayor número del directorio.

6. DELETE

En esta función es parecida al 'DELETE', el servidor recibe 'DELETE', el nombre del cliente y una ruta completa y se encarga de comprobar si el cliente había publicado un archivo con esa ruta. Si de entre todas sus publicaciones se encuentra un archivo con dicha ruta se borra y se indica al cliente que todo ha ido correctamente, si ha habido algún error de cualquier tipo, se informará al cliente quien indicará qué problema ha ocurrido.

7. LIST_USERS

Esta función y la siguiente tienen otro nivel de complejidad mayor y explicaremos la parte del cliente y del servidor por separado.

1. Por un lado el cliente enviará dos mensajes que son 'LIST_USERS' y el nombre del cliente y se queda esperando a recibir un mensaje. Si el mensaje es 0, es decir, que ha ido todo bien, recibirá otro mensaje que indicará cuantos mensajes más recibirá del servidor. Cada mensaje estará formado por otros tres mensajes que se guardarán en las variables locales 'name', 'IP' y 'PORT'. Finalmente se creará un archivo de nombre 'name' y extensión '.txt' desde se guardará IP y PORT. De esta manera habremos guardados en archivos '.txt' las IPs y puertos de todos los clientes conectados.

2. En la parte del cliente se comprueba primero que el cliente esté conectado y si es así se cuenta el número de clientes conectados y se informará al cliente. El servidor llamará a una función que crea una lista dinámica que recorrerá el directorio './Users' e irá guardando en la lista dinámica el nombre, la IP y el puerto de aquellos usuarios que están conectados.

8. LIST_CONTENT

En esta función es muy similar a la anterior solo que tiene un par de cambios.

En este caso el cliente enviará también el nombre del cliente cuyos archivos se quiere conocer y el servidor comprobará si el cliente actual está conectado y si el cliente del que se quiere conocer la información existe. En caso afirmativo se verá cuántos archivos tiene publicados dicho cliente y se informará al cliente actual para que sepa cuantos mensajes debe esperar recibir y en vez de recibir paquetes de 3 mensajes como en la función anterior, solo recibirá un mensaje que indica la ruta completa de todos los archivos publicados por el cliente cuya información queremos saber.

9. GET_FILE

En esta funcionalidad es la única en la que no se conecta el cliente con el servidor general.

Para ello un cliente se intentará conectar con el servidor del cliente remoto solicitado accediendo a la IP y Puerto correspondiente al cliente remoto guardado tras haber llamado previamente a la función 'LIST_USERS'.

Si la conexión ha ido correctamente el cliente remoto comprobará si el archivo solicitado con la ruta completa enviada existe y en caso afirmativo se indica al cliente que pide el archivo que sí que existe así como su tamaño. Finalmente se envían bloques de 1024 bytes en 1024 bytes hasta haber enviado completamente el archivo y haber sido recibido por el cliente que lo solicitó y guardado en la ruta local proporcionada

10. SERVIDOR

Esto no es una función como tal pero si queríamos dedicarle un apartado al funcionamiento general del servidor.

El servidor implementado está muy inspirado en los ejercicios evaluables, salvo que se ha optado por la creación de hilos por cada petición enviada, debido a que había que pasar la IP del usuario, además, creemos que la funcionalidad principal del sistema es el paso de archivos entre usuarios, por lo que nos hemos permitido tener un servidor menos eficiente ya que el servidor no debería sufrir la mayor parte de la carga de trabajo del sistema.

Es decir, contamos con un servidor concurrente, que se encarga de usar las funciones definidas en users.c, todas las previamente tratadas.

Descripción del código, Partes 2 y 3

Parte 2

Se ha desarrollado para la parte 2 un servicio SOAP en Python, siguiendo el ejemplo de las transparencias, antes de esta implementación final, adecuamos el servidor y cliente para mandar una cadena predefinida para comprobar el funcionamiento de la parte 3, que se hizo antes que esta.

Posteriormente se sustituyó esta cadena predefinida por la llamada obtener_fecha_hora, donde el servicio web devuelve la fecha y la hora en la que se ejecuta una función del cliente. Para esto se tuvo que modificar también el client.py y el server.c, pues tenía que leer una vez más para recoger la hora y fecha enviadas desde el client.py, que lo pide a ws-time-service.py, esto para todas las funciones definidas en el cliente, que requieran de conectarse con el servidor.

Parte 3

Siguiendo el ejercicio evaluable 3, definimos el archivo .x para definir el único procedimiento remoto al que vamos a llamar, que imprimirá la información proporcionada por el servidor de la parte 1. En este caso, el servidor de la parte 1 actúa de cliente en esta parte, y onc_rpc_server, el servidor.

Así que adecuamos server.c para ir recogiendo la información necesaria y recoger la IP de la variable de entorno LOG_IP_RPC, para mandar toda la información y que se imprima en el servidor. La única función definida en el servidor es print_operation, que recibe el nombre de usuario, la operación, la fecha y la hora, que se imprime en onc_rpc_server.

Instrucciones para la Compilación y Ejecución

Estas instrucciones están detalladas también en el archivo 'README.txt' pero aquí las vamos a volver a explicar con un poco más de detalle.

Compilación

Para la compilación de nuestro proyecto hay que realizar los siguientes pasos:

- ★ **Paso 1:** Realizar rpcgen -NMa print_rpc.x, esto crea todos los archivos necesarios para la comunicación rpc de la Parte 3, print_rpc_clnt.c, print_rpc_svc.c y print_rpc_xdr.c.
- ★ **Paso 2:** Ejecutar make en terminal, compilará los archivos necesarios para crear los ejecutables 'server' y 'onc_rpc_server', el servidor socket y rpc. Para onc_rpc_server:onc_rpc_server.c print_rpc_svc.c print_rpc_xdr.c, y para server: server.c print_rpc_clnt.c print_rpc_xdr.c users.c lines.c.
- ★ **Paso 3:** Abrir los archivos 'ws-time-service.py' y 'ws-calc-client.py' e instalar los paquetes necesarios para realizar los imports.

Ejecución

Tras haber compilado nuestro proyecto, es necesario hacer una serie de pasos para la ejecución del mismo, recomendamos por claridad tener un terminal abierto por cada ejecutable y archivo .py:

- ★ **Paso 1:** Ejecutar en un terminal **./one_rpc_server**, el servidor rpc que imprimirá la operación que hacemos junto con el usuario y la fecha y hora.
- ★ **Paso 2:** Ejecutar en otro terminal **./server -p 4200**, lanza el servidor socket que ejecuta las funciones definidas en la parte 1.
- ★ **Paso 3:** Ejecutar en un tercer terminal **python3 ws-time-service.py**, el servicio web que proporciona la hora y fecha cuando se realiza una operación.
- ★ **Paso 4:** Ejecutar en un último terminal **python3 client.py -s localhost -p 4200**, el cliente python de este sistema, desde el que se escribirán los comandos para usar el sistema.

Batería de pruebas

Para comprobar el correcto funcionamiento de todo nuestro proyecto hemos realizado las pruebas descritas a continuación:

1. REGISTER, CONNECT, DISCONNECT Y UNREGISTER

La primera prueba consiste en comprobar que nuestro programa funciona correctamente y que permite realizar registros de usuarios y guarda correctamente la información, para ello hemos ejecutado el siguiente código.

```
Python
REGISTER A

CONNECT B

CONNECT A

DISCONNECT B

DISCONNECT A

UNREGISTER A
```

Tras la ejecución de este código comprobamos si se ha creado correctamente el usuario en la parte del servidor, si se ha guardado correctamente la IP y el PORT del socket de servicio creado en la parte de cliente y si finalmente se han borrado correctamente los datos en la parte de servidor y se ha cerrado el socket de servicio.

Por otro lado también se pretende comprobar si devuelve correctamente los errores al intentar hacer un ‘connect’ y un ‘disconnect’ del usuario B que nunca ha sido registrado y por lo tanto tampoco conectado.

Como hemos comentado anteriormente, solo se deberá probar a conectar o desconectar un usuario registrado por ti mismo o un usuario que sepamos que no ha sido registrado por otra persona en otro equipo, esto es para evitar que un usuario registre B y haga un ‘connect’ y otro usuario haga otro ‘connect’ o intente hacer ‘disconnect’.

2. PUBLISH

En esta prueba se ha querido comprobar que se guardan los archivos enviados mediante 'PUBLISH' correctamente al servidor a su vez que no permite archivos con rutas absolutas duplicadas y que solo permite hacer 'PUBLISH' a usuarios conectados.

```
Python
REGISTER B

PUBLISH /home/ENTREGA/README.txt Esto es una prueba

CONNECT B

PUBLISH /home/ENTREGA/README.txt Esto es una prueba

PUBLISH /home/ENTREGA/README.txt Esto es una prueba

DISCONNECT B
```

Con esta prueba registramos un usuario pero no lo conectamos antes de hacer el 'PUBLISH', por lo que nos da un error, lo conectamos, hacemos y enviamos el mismo archivo dos veces lo cual nos dará un error indicando que no puede haber archivos duplicados.

3. DELETE

Para este apartado hemos probado una vez ya teniendo un usuario conectado, a borrar un archivo que no existe y después publicarlo y borrarlo para ver si el servidor se comportaba correctamente.

```
Python
DELETE /home/ENTREGA/README.txt

PUBLISH /home/ENTREGA/README.txt Esto es una prueba

PUBLISH /home/ENTREGA/PEPELU.txt Esto es una prueba 2

DELETE /home/ENTREGA/PEPELU.txt

DELETE /home/ENTREGA/README.txt
```

Tras realizar la prueba podremos comprobar que el servidor actúa de forma correcta publicando los archivos y eliminándolos cuando existen y mandando un mensaje de error cuando no existen.

4. LIST_USERS

Esta prueba sirve para comprobar si tras ejecutar este comando se guardan en el ordenador del usuario archivos con el nombre de los usuarios conectados con su IP y puerto guardados en dos líneas diferentes y a su vez comprobar que no se guardan los usuarios no conectados.

```
Python
REGISTER AA

REGISTER BB

CONNECT BB

LIST_USERS
```

Tras ejecutar esta prueba vemos que solo nos guarda un único archivo 'BB.txt' que es el único usuario conectado por el momento.

5. LIST_CONTENT

En esta prueba suponiendo de nuevo que el usuario actual está conectado intentaremos listar los contenidos publicados por los usuarios A que no ha subido nada, el usuario B que ha subido 3 archivos y el usuario Z que no está registrado.

```
Python
LIST_CONTENT A

LIST_CONTENT B

LIST_CONTENT Z
```

Tras ejecutar esto podemos ver que para el usuario A simplemente nos indica OK pero no imprime nada, para el usuario B nos imprime OK y la ruta de los 3 archivos que ha publicado y para el usuario Z nos indica que ese usuario no existe.

6. GET_FILE

Para la última función ejecutaremos dos códigos en dos equipos diferentes con la intención de que el equipo ‘B’ vea los archivos publicados por el equipo ‘A’ y transfiera uno de esos archivos a su equipo:

Equipo A:

```python
REGISTER A

CONNECT A

PUBLISH /home/README.txt Esto es una prueba del README

PUBLISH /home/autores.txt Esto es una prueba de los autores
```

Equipo B:

```python
REGISTER B

CONNECT B

LIST_USERS

LIST_CONTENT A

GET_FILE A /home/README.txt /home/README2.txt
```

Tras ejecutar este código podremos observar como en el equipo ‘B’ se ha descargado correctamente el archivo enviado desde el equipo ‘A’.

Con estas pruebas ya habríamos comprobado todas las funcionalidades implementadas para la parte 1 de la práctica. Hemos comprobado que el cliente envía los datos correctamente, que el servidor los recibe y procesa, que dependiendo de qué argumentos recibe realiza una u otras funciones y que finalmente el usuario en función de qué quería realizar y de los argumentos que había enviado recibirá un mensaje de error o por contrario una serie de mensajes con información si así lo esperaba.

A continuación vamos a explicar también una breve prueba que hemos realizado para comprobar si hemos implementado correctamente la parte 2 y parte 3 de nuestro proyecto.

7. Otras pruebas, Parte 2 y Parte 3

Para este apartado se ha hecho una prueba simple encargada de comprobar si el servidor desarrollado en RCP imprime lo que debería, para ello hemos ejecutado en un cliente los siguientes comandos:

```
Python
REGISTER AA

REGISTER BB

CONNECT BB

LIST_USERS
```

Y vemos que el ‘onc_rpc_server’ imprime de forma correcta lo siguiente:

```
Python
AA REGISTER 08/05/2025 16:56:45

BB REGISTER 08/05/2025 16:56:50

BB CONNECT 08/05/2025 16:56:53

BB LIST_USERS 08/05/2025 16:56:57
```

Conclusiones

I. **Problemas Encontrados:** Durante la elaboración de esta práctica no nos hemos encontrado ningún problema relevante a excepción de algún error a la hora de compilar o alguna interpretación errónea del enunciado que nos ha llevado a hacer algún apartado de forma errónea.

II. **Soluciones a los Problemas Encontrados:** Como ya hemos mencionado, los únicos problemas encontrados han sido a la hora de compilar y como mencionamos en el siguiente apartado, generalmente hemos resuelto estos problemas con el uso de las IAs Generativas, concretamente de Chat GPT.

III. **Opiniones Personales:** Ambos compañeros creemos que esta práctica y en concreto esta asignatura ha sido una asignatura útil y bien planteada ya que nos ha enseñado aspectos y conocimientos sobre la comunicación entre clientes y procesos usando diferentes técnicas lo cual nos puede ayudar en un futuro, por ello, estamos satisfechos con esta práctica y con la asignatura en general.

Declaración de uso de IA

Para la elaboración de este documento no se ha realizado una IA Generativa en ningún momento ni para la generación de ideas, generación de texto o corrección del mismo.

Sin embargo, para la elaboración de la práctica se ha usado el uso de la IA Generativa de forma exclusiva y puntual para la resolución de algunos problemas y su interpretación que no conseguimos resolver con el material de clase ni con la información encontrada en internet.

Material adicional

También hay que comentar que se ha usado el material proporcionado por el profesor Alejandro Calderon en su github, así como tutoriales y materiales online para alguna parte muy en concreto del código que supera nuestros conocimientos.