uc3m | Universidad Carlos III de Madrid

Sistemas Distribuidos

Práctica Final

Mayo 2025

Ruben Castro Pruneda
100454441@alumnos.uc3m.es
Doble Grado Informática y ADE

Alberto Molina Felipe
100454278@alumnos.uc3m.es
Doble Grado Informática y ADE

Índice

1 Diseño del proyecto

El proyecto se ha organizado en módulos según su funcionalidad. El servidor principal está en `server.c`, donde también se encuentran los handlers de comandos, definidos en `server.h`. Las funciones relacionadas con la base de datos se encuentran en `db.c` y `db.h`.

El cliente principal es `client.py`, y el servidor web que proporciona la fecha y hora se encuentra en `date.py`. La interfaz RPC está definida en `logger.x`, y su implementación en el servidor se encuentra en `logger_server.c`.

1.1 Servidor Principal

La estructura del servidor principal sigue un esquema claro: la función `main` se encarga de inicializar el cliente RPC y el socket en el puerto correspondiente, y posteriormente entra en un bucle de escucha para recibir comandos.

Cada comando recibido da lugar a la creación de un nuevo hilo que procesa la solicitud. Este hilo identifica el comando y ejecuta el *handler* correspondiente. Todos los *handlers* siguen una estructura común: primero leen los argumentos desde el `buffer` del socket, luego realizan una llamada al servidor de log mediante RPC, ejecutan las operaciones necesarias sobre la base de datos, y finalmente preparan (en mayor o menor medida) la respuesta que se enviará a través del socket.

El servidor utiliza una base de datos en memoria implementada mediante listas enlazadas. Se han definido dos estructuras principales:

- Una estructura para representar archivos publicados, con campos para el nombre del archivo, su descripción, y un puntero al siguiente archivo de la lista.
- Una estructura para representar usuarios, con campos para el nombre de usuario, estado de conexión, dirección IP, puerto de escucha, un puntero a su lista de archivos publicados, y un puntero al siguiente usuario.

Cada nodo de usuario representa a un usuario registrado en el sistema, y enlaza directamente con sus archivos publicados. Para garantizar la exclusión mutua durante el acceso concurrente, todas las operaciones sobre la base de datos están protegidas mediante un `mutex`. Las funciones implementadas devuelven códigos de resultado conforme a las especificaciones del protocolo del enunciado.

1.2 Cliente Principal

El cliente principal, implementado en `client.py`, recibe como argumentos por terminal la dirección y el puerto del servidor principal al que debe conectarse.

Este cliente está diseñado para recibir comandos tanto de forma interactiva, escribiéndolos directamente en la terminal, como a través de un `pipe`, por ejemplo mediante `cat` de un fichero de comandos, como se utiliza en los tests automatizados.

La mayoría de los comandos siguen el comportamiento definido en el protocolo: el cliente envía el comando y sus argumentos a través del socket. Antes de cada envío, se realiza una llamada al servidor web para obtener la fecha y hora actuales, que se incluye en el log mediante RPC.

A continuación, se detallan los casos especiales y algunas decisiones de diseño que hemos tomado y que no están especificadas explícitamente en el enunciado.

Transferencia de archivos: hilo de escucha y comando `GET_FILE`

La transferencia de archivos entre clientes (*peer-to-peer*) es la funcionalidad principal del proyecto. Cada cliente lanza un hilo de escucha al ejecutar el comando `CONNECT`, y lo cierra al ejecutar `DISCONNECT`. Este hilo permite atender solicitudes de otros clientes para la descarga de archivos publicados.

El comando `GET_FILE` establece una conexión directa con el cliente que posee el archivo, utilizando la IP y el puerto obtenidos de la tabla de usuarios conectados, que se ha poblado previamente mediante `LIST_USERS`. Si el archivo existe, se transfiere y guarda localmente; si ocurre un error, se muestra un mensaje y se elimina el archivo incompleto.

Además, el cliente acepta rutas tanto absolutas como relativas al publicar contenidos. La ruta que se especifica al ejecutar el comando `PUBLISH` se trata como un string. Si esta cadena corresponde efectivamente a un path válido (ya sea absoluto o relativo), la publicación será correcta y el fichero estará disponible para ser descargado por otros clientes. En caso contrario, al ejecutar el comando `GET_FILE`, si el path proporcionado no corresponde a un fichero existente en el sistema de archivos del cliente emisor se mostrará un error indicando que el fichero no existe.

Restricciones en los comandos `CONNECT` y `DISCONNECT`

Hemos decidido restringir el uso del cliente a un único usuario conectado a la vez. Aunque es posible permitir múltiples conexiones simultáneas desde el mismo cliente, utilizando varios hilos de escucha, optamos por esta solución por varios motivos: no se especifica lo contrario en el enunciado, simplifica considerablemente la implementación, nos parece más coherente con la lógica de uso del sistema, y encaja con el diseño multihilo previsto inicialmente en `client.py`.

Esta restricción se manifiesta de la siguiente forma: si se intenta conectar un nuevo usuario mientras ya hay otro conectado desde ese cliente, el comando se cancela y se muestra el siguiente mensaje:

```
c> CONNECT FAIL, <usuario_actual> IS ALREADY CONNECTED, DISCONNECT FIRST
```

Además, solo el usuario actualmente conectado puede ejecutar el comando `DISCONNECT`. Consideramos que no tendría sentido permitir la desconexión de un usuario conectado desde otra instancia del cliente.

Cabe mencionar que en el comando `LIST_CONTENT`, además de mostrar el nombre de los ficheros publicados, hemos decidido incluir también su descripción, a pesar de que esto no se especifica explícitamente en el enunciado. Consideramos que esta información es útil para el usuario y coherente con el resto de funcionalidades del sistema.

1.3 Servidor Web

El servidor web, implementado en `date.py`, responde en la ruta `/datetime` con la fecha y hora actuales en texto plano. Escucha en `127.0.0.1:8000` y se usa para obtener la marca temporal antes de enviar comandos.

1.4 Servidor RPC de Log

El servidor RPC, definido en `logger.x` e implementado en `logger_server.c`, recibe estructuras `log_entry` con el nombre de usuario, la operación y la fecha. La función `log_operation` imprime cada entrada por consola y devuelve un código de éxito.

2 Compilación y ejecutables

El proyecto incluye un `Makefile` sencillo que genera los binarios del servidor principal y del servidor RPC de log. Al ejecutar `make`, los ejecutables se compilan en el directorio `bin/` para mantener limpio el repositorio.

Los comandos de ejecución de cada componente son los siguientes:

- `env LOG_RPC_IP=<ip_logger_server> ./bin/server -p <puerto> [-v]`
- `./bin/logger_server`
- `python3 client.py -s <ip_server> -p <puerto>`
- `python3 date.py`

3 Batería de pruebas

Para facilitar la ejecución de los distintos componentes y realizar pruebas, hemos incluido en la entrega el script `debug.sh`. Lo hemos utilizado copiando nuestro proyecto dentro del repositorio `u20-docker`, y preparando el entorno para que cada componente se ejecute en un contenedor específico.

El script está diseñado para que el servidor principal se ejecute en el primer contenedor, el servidor RPC en el segundo, y en cada uno de los contenedores restantes se inicie un cliente junto con su correspondiente servidor web.

Las direcciones IP de los contenedores se extraen del fichero `../machines`, y se utiliza el puerto `6677`, elegido arbitrariamente para todas las comunicaciones. Los ficheros que se pasan al cliente en modo `-c` son directamente scripts de prueba que contienen los comandos a ejecutar.

A continuación se muestra el resultado del test más simple de transferencia de fichero.

Cliente 1

```
./debug.sh -c commands1.txt
c> REGISTER USER_1
c> REGISTER OK
c> CONNECT USER_1
c> CONNECT OK
c> PUBLISH id_ed25519 TOP SECRET
c> PUBLISH OK
c> File id_ed25519 sent successfully
```

Log server

```
USER_1 REGISTER 01/05/2025 14:11:20
USER_1 CONNECT 01/05/2025 14:11:20
USER_1 PUBLISH id_ed25519 01/05/2025 14:11:20
USER_2 REGISTER 01/05/2025 14:11:27
USER_2 CONNECT 01/05/2025 14:11:27
USER_2 LIST_USERS 01/05/2025 14:11:27
USER_2 LIST_CONTENT 01/05/2025 14:11:27
```

Cliente 2

```
./debug.sh -c commands2.txt
c> REGISTER USER_2
c> REGISTER OK
c> CONNECT USER_2
c> CONNECT OK
c> LIST_USERS
c> LIST_USERS OK
    USER_2     172.30.0.5 55289
    USER_1     172.30.0.4 46765
c> LIST_CONTENT USER_1
c> LIST_CONTENT OK
    id_ed25519  "TOP SECRET"
c> GET_FILE USER_1 id_ed25519 key_copy
c> GET_FILE OK
```

3.1 Unittest

Para cada uno de los comandos del cliente comprobamos todos los casos y si interación con el servidor principal.

register.txt

Éxito
Ya registrado

```
c> REGISTER register
c> REGISTER OK
c> REGISTER register
c> USERNAME ALREADY IN USER
```

unregister.txt

Éxito
No existente

```
c> REGISTER unregister
c> REGISTER OK
c> UNREGISTER unregister
c> UNREGISTER OK
c> UNREGISTER unregister
c> USER DOES NOT EXIST
```

connect.txt

No existente
Éxito
Ya conectado

```
c> REGISTER CONNECT
c> REGISTER OK
c> CONNECT NO_CONECT
c> CONNECT FAIL, USER DOES NOT EXIST
c> CONNECT CONNECT
c> CONNECT OK
c> CONNECT CONNECT
c> USER ALREADY CONNECTED
```

disconnect.txt

Éxito
Ya desconectado
No existente

```
c> REGISTER disconnect
c> REGISTER OK
c> CONNECT disconnect
c> CONNECT OK
c> DISCONNECT disconnect
c> DISCONNECT OK
c> DISCONNECT disconnect
c> DISCONNECT FAIL,
    YOU CAN ONLY DISCONNECT
    THE USER CURRENTLY CONNECTED
    IN THIS CLIENT
c> DISCONNECT no_disconnect
c> DISCONNECT FAIL, YOU CAN ...
```

list_users.txt

(usando publishN.txt)
Varios usuarios

```
c> REGISTER list_users
c> REGISTER OK
c> CONNECT list_users
c> CONNECT OK
c> LIST_USERS
c> LIST_USERS OK
    list_users 172.30.0.4 59283
    publish_1  172.30.0.5 36151
    publish_2  172.30.0.5 49223
```

list_content.txt

(usando publish1.txt)
Varios usuarios

```
c> REGISTER list_content
c> REGISTER OK
c> CONNECT list_content
c> CONNECT OK
c> LIST_USERS
c> LIST_USERS OK
    list_content 172.30.0.4 58285
    publish_1  172.30.0.5 37723
c> LIST_CONTENT publish_1
c> LIST_CONTENT OK
    unittest/publish2.txt  "dumb_file_12"
```

publish.txt

Éxito
Fichero duplicado
Desconectado

```
c> REGISTER publish
c> REGISTER OK
c> CONNECT publish
c> CONNECT OK
c> PUBLISH publish.txt dumb_file
c> PUBLISH OK
c> PUBLISH publish.txt dumb_file
c> PUBLISH FAIL,
    CONTENT ALREADY PUBLISHED
c> DISCONNECT publish
c> DISCONNECT OK
c> PUBLISH publish.txt dumb_file
c> PUBLISH FAIL,
    USER NOT CONNECTED
```

delete.txt

Éxito
Fichero no existente
Desconectado

```
c> REGISTER delete
c> REGISTER OK
c> CONNECT delete
c> CONNECT OK
c> PUBLISH delete.txt dumb_file
c> PUBLISH OK
c> DELETE delete.txt
c> DELETE OK
c> DELETE delete.txt
c> DELETE FAIL,
    CONTENT NOT PUBLISHED
c> DISCONNECT delete
c> DISCONNECT OK
c> DELETE delete.txt
c> DELETE FAIL, USER NOT CONNECTED
```

get_file.txt

Usuario no existente
Fichero no existente
Éxito

```
c> REGISTER get_file
c> REGISTER OK
c> LIST_USERS
c> LIST_USERS OK
    get_file   172.30.0.4 60941
    publish_1  172.30.0.5 60529
c> GET_FILE no_user no_file remote_copy
c> GET_FILE FAIL, USER NOT FOUND
c> GET_FILE publish_1 no_file remote_copy
c> GET_FILE FAIL, FILE NOT EXIST
c> GET_FILE PUBLISH_1 publish2.txt remote_copy
c> GET_FILE OK
```

3.2 Casos específicos

Fallos de conexión

- Cliente principal sin servidor web

```
c> REGISTER no_web_server
c> ERROR getting datetime from Web Service: [Errno 111] Connection refused
c> REGISTER FAIL, COULD NOT GET DATETIME
```

- Cliente principal sin servidor sockets

```
c> register no_socket_server
c> REGISTER FAIL [Errno 111] Connection refused
```

- Servidor principal sin servidor RPC

```
s> init server 0.0.0.0:6677
s>
RPC failed: RPC: Unable to receive; errno = Connection refused
```

- Transferencia de fichero cuando se cierra el cliente, pero no se desconecta con el servidor

```
c> REGISTER USER_1
c> REGISTER OK
c> CONNECT USER_1
c> CONNECT OK
c> PUBLISH test_data/id_ed25519 TOP SECRET
c> PUBLISH OK
c> quit
--------------
c> REGISTER USER_2
c> REGISTER OK
c> CONNECT USER_2
c> CONNECT OK
c> LIST_USERS
c> LIST_USERS OK
    USER_2     172.30.0.4 53305
    USER_1     172.30.0.4 45131
c> LIST_CONTENT USER_1
c> LIST_CONTENT OK
    test_data/id_ed25519  "TOP SECRET"
c> GET_FILE USER_1 test_data/id_ed25519 test_data/key_copy
c> GET_FILE FAIL [Errno 111] Connection refused
```

3.3 Casos no contemplados

Si un usuario se registra y conecta en un cliente, pero cierra el ciente antes de desconectarse será imposible volverse a conectar como este usuario. Ningún otro cliente podrá conectarse porque el servidor responderá `USER ALREADY CONNECTED` y nosotros no contemplamos que un usuario se desconecte desde un cliente en el que no esta ya conectado.

Se podría solucionar haciendo que tanto el comando quit como un handler para sigkill manden el comando de disconnect.

4 Conclusiones

Nos ha encantado desarrollar este proyecto. Combinar sockets, RPC y HTTP en un mismo sistema nos ha ayudado a entender la complejidad real que pueden alcanzar las aplicaciones distribuidas.

Lo único que se nos ha hecho más complicado ha sido la parte de testing. Ya lo habíamos visto en las prácticas de empresa, pero probar microservicios o sistemas distribuidos no es tan directo como probar programas secuenciales. Llegamos a debatir si hacer mock servers para automatizar mejor los tests, pero lo descartamos por la carga de trabajo.