uc3m | Universidad Carlos III de Madrid

Sistemas Distribuidos

Ejercicio 3 - RPC

Abril 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

Diseño

Este proyecto es una evolución de un sistema anterior basado en sockets, con el mismo comportamiento funcional, pero migrado a una arquitectura RPC usando rpcgen. La lógica del sistema de tuplas se mantiene igual, y solo cambia la forma de comunicación entre cliente y servidor.

El sistema se divide en tres componentes:

- Cliente (proxy): implementado en `proxy-rpc.c`, ofrece las funciones públicas del sistema de tuplas y se compila como una biblioteca compartida (libclaves.so) e internamente, llama a las funciones RPC generadas automáticamente.
- Servidor: contiene las funciones `*_rpc_1_svc` en `servidor-rpc.c`, que reciben las peticiones remotas y delegan en la lógica de claves.c, la cual no ha cambiado respecto a la versión por sockets.
- Interfaz RPC: se define en `claves_rpc.x`, donde se declaran las estructuras y operaciones. A partir de ahí se generan automáticamente los ficheros de cliente y servidor (*_clnt.c, *_svc.c, *_xdr.c, etc.).

El uso de RPC simplifica considerablemente la serialización de datos y la gestión de la comunicación entre procesos, eliminando gran parte del código de red manual que era necesario en la versión pasada con sockets (como el envío y recepción de buffers, el manejo de tamaños, o la reconstrucción de estructuras). En lugar de construir manualmente un protocolo, la interfaz se define declarativamente en el archivo .x, y rpcgen se encarga de generar automáticamente el código necesario para la comunicación.

Aunque esta aproximación introduce una dificultad inicial relacionada con la configuración correcta de la interfaz y la comprensión del sistema generado, una vez establecida, la solución resulta más eficiente, mantenible y fácil de entender. Las estructuras de datos y funciones quedan explícitamente documentadas, y la separación entre lógica y transporte es más clara.

Compilación

Como indica el enunciado se compila el proxy como una biblioteca compartida y se generan dos binarios `bin/cliente` y `bin/servidor` con el comando `make`.

El binario del servidor se genera a partir del código generado automáticamente por rpcgen, concretamente el fichero `claves_rpc_svc`, pero la implementación de la lógica está separada en el archivo servidor-rpc.c, que se compila y se enlaza junto con los archivos generados.

```
$ ./bin/servidor
```

Para el cliente es hace falta tener las variables de entorno `IP_TUPLAS` y `PORT_TUPLAS` definidas.

```
$ env IP_TUPLAS=localhost ./bin/cliente
```

El sistema de librerías es más restrictivo en NixOS, especialmente en lo relativo a la carga dinámica de bibliotecas compartidas, todo el sistema ha sido probado dentro de un contenedor Docker con Ubuntu, donde el entorno es más estándar y compatible con el modelo de compilación y ejecución utilizado por rpcgen.

Tests

Los tests son los mismos que en el ejercicio anterior.

- Guardar un valor y recuperarlo para comprobar que coincide.
- Intentar obtener un valor de una clave que no ha sido almacenada.
- Intentar almacenar datos con un formato inválido para verificar que se detectan errores.
- Comprobar que no se puede sobrescribir una clave ya establecida.
- Verificar que una clave existe antes y después de eliminarla.
- Almacenar un valor, modificarlo y recuperarlo para confirmar los cambios.
- Intentar modificar una clave que no existe para comprobar que se genera un error.