



Universidad Carlos III

Sistemas Distribuidos

Curso 2023-24

Práctica 3

RPC

Ingeniería Informática, Tercer curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)

César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)

Prof . Félix García Caballeira y Alejandro Calderón Mateos

Grupo: 81

Índice

1. Introducción	2
2. Diseño	2
2.1. Cliente	2
2.2. Servidor	2
2.2.1. Implementación en el servidor	2
2.2.2. Concurrencia del Servidor	2
2.3. Comunicación	2
2.3.1. Interfaz.x	2
2.3.2. Modificación de la plantilla	3
3. Compilación	3
3.1. Biblioteca Dinámica	3
3.2. Cliente	4
3.3. Servidor	4
4. Descripción de pruebas	4
5. Conclusiones	4

1. Introducción

El desarrollo de este proyecto consiste en implementar una aplicación cliente-servidor, donde los diferentes clientes podrán guardar información en tuplas a través del servidor, de forma transparente. La comunicación entre el cliente y el servidor se dará a través de **llamadas a procedimientos remotos**. Los mensajes transmitidos deben ser independientes del lenguaje de programación.

2. Diseño

La aplicación constará de dos partes diferenciadas: los clientes y el servidor.

2.1. Cliente

Siguiendo la misma aproximación que en el ejercicio anterior, el cliente no ha necesitado de cambios ya que la única parte afectada por los requisitos de este ejercicio es la comunicación. La comunicación entre el cliente y el servidor está completamente encapsulada en la biblioteca dinámica *libclaves.so*. Por esto, se han utilizado los ficheros proporcionados por *rpcgen* como plantilla, que sirven como código fuente de la biblioteca dinámica.

2.2. Servidor

En el servidor se encuentran las funcionalidades que se encargan de la comunicación con el cliente, en el fichero *src/servidor.c*, y la implementación de los servicios para las tuplas, en el fichero *src/imp_clave.c*. De nuevo, el único archivo que necesitó de modificaciones es *servidor.c* en la que se adaptan los contenidos de la plantilla dada por *rpcgen*.

2.2.1. Implementación en el servidor

La implementación de los servicios requeridos está contenida en el fichero *src/imp_claves.c*. Dado que los servicios coinciden exactamente con los del ejercicio anterior, este fichero no ha sufrido ningún cambio. Se compilará junto con el servidor.

Siguiendo la misma idea que en la práctica anterior, guardaremos las tuplas en ficheros dado su carácter permanente entre sesiones y las facilidades que presentan para la gestión de la concurrencia.

2.2.2. Concurrencia del Servidor

A través de *rpc* no se puede asegurar la concurrencia, ni la creación de hilos; pero la posible existencia de dos clientes concurrentes nos obliga a utilizar un código sin condiciones de carrera. Para ello no es necesario utilizar ningún mecanismo de sincronización dado que en la implementación se usan ficheros con tamaños más pequeños que una página, por lo que el sistema operativo nos asegura la inexistencia de condiciones de carrera.

2.3. Comunicación

La comunicación, tal y como especificamos anteriormente, se realiza a través de **llamadas a procedimientos remotos**. Este mecanismo permite comunicar dos procesos cualesquiera en la misma red, independientemente de la arquitectura de la máquina que los ejecuta o el lenguaje de programación en el que esté escrita la aplicación. El código proporcionado por *rpcgen* incluye *marshaling* y *unmarshaling*, por lo que no debemos preocuparnos por estos menesteres.

2.3.1. Interfaz.x

Para especificar la interfaz en lenguaje XDR se ha analizado cada operación para especificar sus argumentos de entrada y salida. Pese al parecido que tiene este lenguaje con lenguaje C, caben destacar algunas particularidades y decisiones de diseño tomadas para el desarrollo de la práctica.

En cuanto a las operaciones, *init* no recibe argumentos, *delete_key*, *exist* y *get_value* reciben únicamente un valor entero y *set_value* y *modify_value* reciben 2 punteros (*value1* y *value2*) y un valor entero. Todas las funciones devuelven únicamente un argumento de salida, un valor entero que indica si se ha realizado exitosamente, a excepción de *get_value* que devuelve además los valores

almacenados en una tupla (i.e. un array de caracteres, un número entero y un array de 32 números en coma flotante de doble precisión).

Con esto en mente, se ha especificado la interfaz de la siguiente forma.

```
typedef double vector_value2<>;
struct get_exit_args {
    char value1[256];
    int N_value2;
    double value2[32];
    int result;
};

program CLAVES {
    version CLAVESVER {
        int rpc_init() = 0;
        int rpc_set_value(int key, string value1, int N_value2,
            vector_value2 value2) = 1;
        get_exit_args rpc_get_value(int key) = 2;
        int rpc_modify_value(int key, string value1, int N_value2,
            vector_value2 value2) = 3;
        int rpc_delete_key(int key) = 4;
        int rpc_exist(int key) = 5;
    } = 1;
} = 99;
```

Para los arrays hemos utilizado una construcción que nos permite inicializarlos como memoria estática. De esta forma hemos evitado tener que gestionar memoria dinámica en la implementación. Los argumentos de salida de la función *get_value* se han encapsulado en una estructura.

2.3.2. Modificación de la plantilla

Rpc nos ha proporcionado una plantilla dada la interfaz antes mencionada. De esta plantilla hemos desechado del lado cliente el fichero *claves_rpc_client.c* para generar un *cliente.c* y *claves.c*. En esta segunda nos hemos servido de *rpcgen* para implementar la interfaz del cliente. En cada una de las implementaciones de las funciones se utilizan los argumentos dado para llamar a la función creada por *rpc* propia de cada función, la cual realiza el correspondiente *marshalling/unmarshalling* y vuelca el contenido en la red. Respecto a la plantilla del lado servidor se ha renombrado el fichero *claves_rpc_server.c* a *servidor.c* y en cada una de las funciones proporcionadas se invoca a las funciones de la implementación.

3. Compilación

En esta sección se describe la forma de compilar el proyecto para generar dos ejecutables: cliente y servidor.

Todos los comandos que se exponen a continuación quedan recogidos en un archivo *Makefile.claves_rpc* encargado de generar ambos ejecutables, así como de compilar la biblioteca dinámica. Este archivo parte de la plantilla generada por *rpcgen* y ha sido modificado para adaptarlo a las exigencias de la práctica.

3.1. Biblioteca Dinámica

La compilación de la biblioteca dinámica ha cambiado respecto a las prácticas anteriores. Esto es porque debemos incluir los ficheros generados por *rpcgen* y adaptarnos al *Makefile* que nos otorga como plantilla.

A continuación, se adjunta el código empleado para la compilación de la biblioteca dinámica. Este se ha incluido justo antes de la generación del ejecutable *cliente*.

```
gcc $(LDLIBS) -shared -o libclaves.so $(OBJECTS_CLNT)
gcc -c -o $(SRC)cliente.o $(SRC)cliente.c
gcc -L. -Wl,-rpath=. -o $(CLIENT) $(SRC)cliente.o -lclaves $(LDLIBS)
```

Con estos comandos generamos un fichero *libclaves.so* y lo enlazamos con el ejecutable *cliente*. Cabe destacar que para poder mantener la compatibilidad con versiones de *gcc* anteriores a la 13 se deben incluir las bibliotecas (i.e. *-lclaves* y *LDLIBS*) al final del comando. También ha sido necesario modificar la plantilla para que todos los ficheros objetos se compilen con la opción *-fPIC*, para hacer el código independiente de la posición de memoria.

3.2. Cliente

El código fuente del cliente está contenido en el archivo *cliente.c*. Este se compila con el comando especificado en el apartado anterior. La única modificación de la plantilla ha sido su compilación por separado para enlazarlo con la biblioteca dinámica.

3.3. Servidor

La compilación del servidor viene dada en la plantilla generada por *rpcgen*. La única modificación necesaria ha sido añadir nuestro módulo *imp_clave.c* como fichero fuente del servidor para compilarlo junto con el resto de ficheros.

4. Descripción de pruebas

Para validar el correcto funcionamiento del sistema, hemos realizado dos bloques de pruebas.

El primero, encargado de comprobar la implementación y concurrencia del sistema, reutiliza las pruebas desarrolladas para la práctica anterior. El resultado de estas pruebas es exitoso, tal y como esperábamos ya que no hemos modificado la implementación de los servicios.

Para el segundo bloque, encargado de comprobar la comunicación y correcta serialización de los datos, hemos conectado dos máquinas distintas (una lanzando el servidor y otra varios clientes) en una misma red LAN. El resultado de estas pruebas también ha sido exitoso.

5. Conclusiones

Este ejercicio nos ha permitido explorar una nueva aproximación para el desarrollo de aplicaciones distribuidas, la llamada a procedimientos remotos. Esta forma de desarrollo nos permite una mayor abstracción, ayudándonos de herramientas de generación de código automático.

La herramienta utilizada, aunque algo tosca, nos ofrece las mismas ventajas que el desarrollo con sockets pero con una sintaxis más sencilla y reduciendo el tiempo de codificación. Sin embargo, esto no quita la necesidad de revisar el código generado y adaptarlo a las necesidades concretas de la aplicación. Además, debido a particularidades de nuestros equipos el comando *rpcgen* presentó problemas que complicaron el desarrollo.

El desarrollo de esta práctica, junto con las dos anteriores, nos ha permitido explorar diversos mecanismos de comunicación, cada uno con sus particularidades. Esto nos ha permitido afianzar las bases en la programación de aplicaciones distribuidas, dándonos las herramientas suficientes para afrontar el desarrollo de la práctica final.