Sistemas distribuidos

Grado en Ingeniería Informática

Ejercicio Evaluable 3: RPC

Darío Caballero Polo

(100451112, grupo 81, 100451112@alumnos.uc3m.es)

Raúl Miguel Carrero Martín

(100451286, grupo 81, 100451286@alumnos.uc3m.es)

Tabla de contenidos

Tabla de contenidos..... 2
1.- Introducción..... 3
2.- Interfaz de servicio..... 3
3.- Servidor..... 4
4.- Cliente..... 5
5.- Makefiles..... 5
6.- Ejecución..... 6

1.- Introducción

Se pide modificar el programa de tipo cliente/servidor desarrollado en la anterior práctica de forma que el código de comunicación mediante sockets TCP se generen mediante RPC. Igual que en las anteriores entregas, se almacenan tuplas conteniendo una clave (*key*), un mensaje de texto (*value1*), un número entero conteniendo la cantidad de elementos que habrá en el siguiente elemento de la tupla (*N_value2*) y un array de doubles. Estas tuplas se almacenan en un archivo (*tuplas.txt*) con cada elemento separado por un espacio:

```
1  71 value1_71 11 1005574202.845406 353120746.572108 1369332813.447221 2027373939.138244 1133021284.020646
2  117 value1_117 16 1906336361.502959 1138498024.453847 1821128919.784909 1604407580.006846 523537453.47166
3  6 value1_6 21 989184034.567984 543059730.306885 524384737.174631 504710574.630722 64590066.316518 1074475
4  282 value1_282 4 477332536.755588 647797221.647852 148583778.862261 643197129.949608
5  642 value1_642 14 1661333.368127 1558497579.228892 1682767471.364579 328182071.801045 148078976.749047 21
6  707 value1_707 4 1802589605.294669 1466075105.146361 1875814574.776127 401806917.765328
7  150 value1_150 2 1940028575.831650 584693616.627714
```

Cabe destacar que las líneas 1, 2, 3 y 5 de la imagen han sido truncadas para facilitar la lectura.

2.- Interfaz de servicio

```
struct args
{
    int key;
    string value1<256>;
    int N_value2;
    double V_value2[32];
};

struct get_value_result
{
    string value1<256>;
    int N_value2;
    double V_value2[32];
    int status;
};

program CLAVES {
    version CLAVESVER {
        int rpc_init(void) = 1;
        int rpc_set_value(struct args arg) =2;
        get_value_result rpc_get_value(int key) = 3;
        int rpc_modify_value(struct args arg)=4;
        int rpc_delete_key(int key) = 5;
        int rpc_exist(int key)=6;
        } = 1;
} = 99;
```

En el archivo “*claves.x*” se especifica en formato XDR la interfaz de las funciones para ser generadas mediante RPC. Como se ve, se han creado dos estructuras: *args* y *get_value_result*. La primera de ellas encapsula los argumentos de entrada de las funciones que originalmente tenían más de un parámetro. No es necesario realizar este encapsulamiento, pero puede resultar más cómodo. En cuanto a la segunda estructura, esta se utiliza para devolver el resultado de la función rpc*_get_value()*, ya que además de devolver el código de estado, necesita devolver *value1*, *N_value2* y *V_value2*. Como devuelve más de un resultado, es necesario encapsularlos en una estructura.

3.- Servidor

En el lado del servidor tenemos un archivo llamado *“servidor.c*”. Para desarrollar su código, se partió del servidor generado por *RPCgen* (“*claves_server.c*”) y se añadieron las correspondientes llamadas a las funciones del servidor especificadas en *“funciones_servidor.h*” y *“funciones_servidor.c*”. Esas llamadas se rodearon de un *lock* y *unlock* para permitir concurrencia. Los archivos de funciones del servidor se incluyeron en la carpeta “*funciones_servidor*”.

4.- Cliente

En el lado del cliente, tenemos dos archivos principales: "*cliente_tests.c*" y "*cliente_concurrente.c*". Estos son los mismos que en las anteriores entregas, puesto que las interfaces de las funciones a las que llaman, definidas en "*claves.h*" son independientes a la implementación de la comunicación.

En el directorio "*claves*" se incluye el código de la librería dinámica "*libclaves.so*". Esta se compone del anteriormente mencionado "*claves.h*" y "*claves.c*". El primero de los dos archivos es la cabecera que incluyen los clientes en su código, la cual es independiente a RPC. Sin embargo, "*claves.c*" incluye el archivo "*claves.h*" del directorio raíz (distinto al de la carpeta "*claves*"). Ese archivo de cabecera, en cambio, sí que contiene referencias a RPC ya que es el generado por RPCgen.

De igual forma que en las anteriores entregas, "*cliente_tests.c*" se encarga de probar todas las posibles respuestas de cada función, de forma similar a unos test unitarios. Así se comprueban todos los posibles errores y el funcionamiento correcto de cada función. Los tests no se especificarán en el documento, pues se sobrepasarían las páginas permitidas. Sin embargo, se puede ejecutar el archivo y ver en consola los detalles de cada test. En resumen, se ha probado que la ejecución de cualquier función antes de realizar *init()* resulta en error, que después de realizarlo todas las funciones no dan error y su comportamiento es el esperado (incluyendo la comprobación de los valores de la tupla recuperada con *get_value()*) y que todo ello se mantiene cierto con más de una tupla a la vez.

El otro archivo cliente es "*cliente_concurrente.c*". Este se puede ejecutar las veces que se deseen en distintas terminales, ya que funcionará como clientes diferentes. Estos clientes entran en un bucle infinito enviando continuamente peticiones. El objetivo de estos clientes es probar la ejecución concurrente.

5.- Makefiles

Se trabajó inicialmente con dos archivos *Makefile*: "*Makefile.claves*" y "*Makefile*". El primero es el generado por el comando `rpcgen -NMa claves.x` para la compilación de los archivos necesarios para la comunicación mediante RPC. El segundo es el Makefile de los archivos desarrollados por nosotros. Por lo tanto, había de ejecutarse `make -f Makefile.claves` y después `make`.

Sin embargo, se han combinado ambos *Makefiles* en uno solo: "*Makefile*" para facilitar la compilación.

6.- Ejecución

Igual que en el anterior ejercicio, en este se han facilitado tres *scripts* para ejecutar el servidor, el cliente de tests y el cliente concurrente. Estos son: "*run_servidor.sh*", "*run_cliente_tests.sh*" y "*run_cliente_concurrente.sh*". De esta forma, no es necesario establecer manualmente las variables de entorno antes de correr los programas.