Luis Gómez-Manzanilla Nieto 100472006

Ignacio Fernández Cañedo 100471955

EJERCICIO EVALUABLE 3

ÍNDICE

INTRODUCCIÓN

Este documento contiene la memoria del tercer ejercicio evaluable de la asignatura Sistemas Distribuidos. Este ejercicio trata de diseñar e implementar el mismo servicio distribuido del primer y segundo ejercicios evaluables que permite almacenar tuplas. En esta aplicación, a diferencia de los ejercicios previos, se utilizarán llamadas a procedimientos remotos (RPC), más en concreto se utilizará el modelo ONC RPC.

ESTRUCTURA

La estructura del ejercicio es la siguiente:

clave_valor.x: Este es el archivo de definición de programa para RPC. Este archivo define las estructuras de datos y las funciones que se utilizarán en el programa.

claves.h: Este es el archivo cabecera que contiene la declaración de las funciones. Las funciones incluyen init, set_value, get_value, modify_value, exist y delete_key, que implementan las operaciones que puede realizar el cliente.

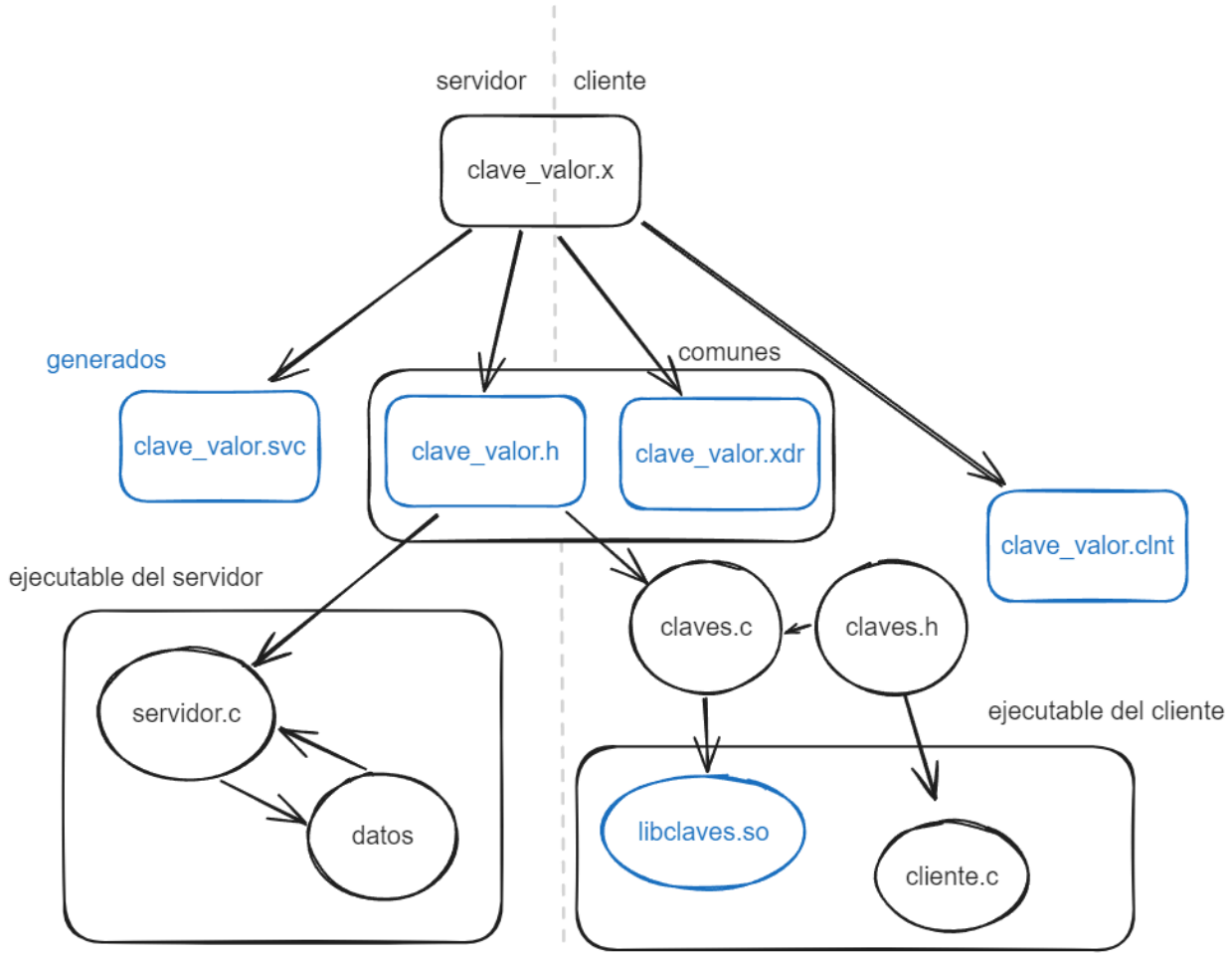
claves.c: Este archivo contiene las implementaciones de las funciones declaradas en claves.h.

clave_valor_client.c, clave_valor_clnt.c, clave_valor_server.c, clave_valor_svc.c, clave_valor_xdr.c, clave_valor.h: Todos estos archivos son los archivos que se generan con el procedimiento rpcgen de RPC, algunos de ellos son necesarios para que funcione correctamente el programa RPC, otros son simples esqueletos para utilizarlos como punto de partida.

Servidor.c: Este archivo contiene el código del servidor. El servidor incluye las funciones para leer y escribir en el archivo, además en este archivo están implementadas las funciones que corresponden a las peticiones de los clientes. Las peticiones pueden ser para añadir una nueva tupla, obtener los valores de una tupla existente, modificar los valores de una tupla existente, comprobar si existe una tupla con una determinada clave o eliminar una tupla. Todas devuelven el resultado. El sistema RPC se encarga de utilizar este resultado para enviarselo a los clientes.

cliente.c: Este archivo contiene el código del cliente. El cliente puede enviar peticiones al servidor mediante las distintas funciones declaradas en la cabecera.

datos: Fichero en el que se almacenarán todas las tuplas.



DISEÑO

Para almacenar los elementos clave-valor1-valor2 hemos decidido utilizar ficheros como estructura de almacenamiento de datos. A diferencia de las listas anidadas, gracias a utilizar ficheros en nuestra implementación, los datos se guardarán entre ejecuciones. Además, tenemos un vector donde se cargan las tuplas existentes de ejecuciones anteriores. Las consultas principalmente afectan al vector y posteriormente este vector se carga en el fichero.

El archivo clave_valor.x define las estructuras de datos y las funciones que se utilizarán en el programa. En primer lugar se define un nuevo tipo de dato double_array que se utilizará en las estructuras. Posteriormente se define una estructura de datos para utilizar como argumento para las funciones set y modify, y otra para get. Después se define programa RPC llamado CLAVE_VALOR. Este programa incluye varias funciones, cada una con un número único que la identifica. Este archivo se utilizará para generar el resto de archivos RPC.

En el caso de claves.c, las funciones incluidas en este archivo comienzan con la creación del cliente RPC que se conectará al servidor, para ello utiliza la función clnt_create, pasando como argumentos la dirección del servidor o el protocolo de transporte. Si esto falla, se muestra un mensaje de error y se termina el programa.

Posteriormente, la función realiza la llamada RPC utilizando la función correspondiente. Esta función se comunica con el servidor y solicita la respuesta asociada a dicha función. Si esta función falla también se devolverá un mensaje de error.

Posteriormente, si la llamada RPC tiene éxito (devuelve 0), se realiza lo esperado por la función correspondiente y se devuelve si la operación fue exitosa o no.

Respecto al archivo servidor.c, inicialmente se define el número máximo de tuplas a almacenar y se declaran varias variables globales. Las variables globales incluyen los mutex para la sincronización de hilos, el array de tuplas, el array de claves, el número actual de tuplas y el nombre del archivo donde se almacenan las tuplas.

Posteriormente se crean funciones para escribir y para leer las tuplas en un archivo de texto. Ambas funciones utilizan mutex para asegurarse que solo un hilo puede leer o escribir en el archivo a la vez.

A continuación se declaran las funciones que implementan las operaciones RPC para el almacenamiento clave-valor, en primer lugar se encuentra la función init, que inicializa el servidor, establece el nombre del archivo donde se almacenarán las tuplas y llama a las funciones para leer y escribir las tuplas. El resto de funciones bloquean los mutex al comenzar y se desbloquean al finalizar.

Al utilizar un servicio ONC RPC, la mayoría del código se genera automáticamente al realizar rpcgen. Este código se incluye en los archivos xdr que contiene las funciones de serialización y deserialización que se utilizan para convertir las estructuras de datos, clnt que contiene las funciones stub del cliente que se utilizan para hacer llamadas al servidor y svc que contiene las funciones stub del servidor que se utilizan para recibir llamadas de los clientes.

PRUEBAS

Para ejecutar el ejercicio, en primer lugar se deberá hacer un “make -f Makefile.clave_valor”, después, para ejecutar el servidor basta con escribir “./servidor”. Para ejecutar el cliente se debe escribir “./cliente”. No hace falta incluir la dirección del servidor como argumento ya que esta se adquiere como variable de entorno.

Para realizar pruebas de concurrencia y del funcionamiento del servidor atendiendo peticiones de diferentes clientes, hemos creado dos ficheros de cliente, cliente.c y cliente2.c. Cada uno tiene su ejecutable correspondiente. La diferencia entre estos dos es que cliente es quien se tiene que ejecutar primero ya que es el que realiza el init y en cambio el cliente2 no realiza el init pero utiliza el resto de funciones.

Además, el archivo de cliente.c tiene implementadas todas las funciones para mostrar el correcto comportamiento secuencial (ejecutar las funciones y ver cómo se realizan correctamente). Esto lo hacemos llamando a las funciones y en caso de que se realicen correctamente y se reciba un 0, se imprimirá por pantalla una confirmación.

En cliente.c primero realizamos un init para inicializar el sistema. Posteriormente usamos set_value() para guardar dos tuplas diferentes. Hasta aquí podemos comprobar que se ha generado el archivo datos.txt y contiene las dos tuplas creadas. Posteriormente realizamos un get_value de la segunda tupla e imprimimos los elementos del vector obtenido para comprobar que son los mismos que hemos seteado. A continuación, modificamos la tupla que acabamos de obtener con modify_value() y volvemos a obtenerla e imprimirla para ver como ha cambiado. Por último, usaremos delete_key() de la primera tupla y a continuación usaremos exist() para comprobar si las dos tuplas que

introducimos inicialmente existen. El resultado de esto será que la primera no existe pero la segunda si. Finalmente en datos.txt de las dos tuplas que hemos añadido inicialmente solo queda la segunda y tendrá los valores del vector modificados.

Por otro lado, el cliente2 está pensado para comprobar el comportamiento concurrente. Para ello, una vez compilado el proyecto, inicialmente ejecutaremos el cliente como se explica anteriormente para realizar el init. Posteriormente ejecutamos el siguiente comando “./cliente2 & ./cliente2”, con este creamos dos procesos que ejecuta el cliente2 concurrentemente.

En este archivo lo que hacemos es set_value() primero dos valores, con key 3 y 4 respectivamente. En caso de que se realice correctamente se mostrará un mensaje de confirmación y en caso contrario uno de error. Como podemos observar al ejecutar, por cada set_value() aparecen los dos mensajes y esto se debe a que un hilo realiza el set correctamente y genera el mensaje correcto y el otro hilo como ya se ha realizado el set, este da error y se muestra el mensaje de error. Posteriormente, comprobamos la existencia de las dos claves introducidas y eliminamos la 3, solo se produce un mensaje de que se ha eliminado ya que en el otro no existe por lo que da error. Por último, se comprueba si la clave que hemos eliminado existe y como podemos observar se muestra dos veces que no existe la clave.