Sistemas distribuido Grupo 84

GRADO EN INGENIERÍA INFORMÁTICA

Curso 2023-24

Ejercicio evaluable 2

Autores: Kaixuan Zhang Chen
NIA: 100472080
Correo: 100472080@alumnos.uc3m.es

Miguel Domínguez Gómez
NIA: 100451258
Correo: 100451258@alumnos.uc3m.es

Introducción

El objetivo de este ejercicio es desarrollar el mismo servicio de los ejercicios 1 y 2 pero en este caso utilizando llamadas a procedimientos remotos (RPC). Para ello se utilizará el modelo ONC RPC visto en clase.

Diseño e implementación

La implementación básica y como funciona las funciones como init ya se explicó en las anteriores entregas de memoria.

file.c no cambia, clave.h no se cambia, cliente.c tampoco.

En claves.c se cambió la implementación para utilizar RPC (Remote Procedure Call) para la comunicación entre el cliente y el servidor. En este enfoque, se establece una conexión mediante el cliente RPC (CLIENT *clnt), y se llaman a funciones RPC específicas (init_tuplas_1, set_value_tuplas_1, etc.) para realizar operaciones en el servidor. Además, se utilizan variables para almacenar los resultados de estas llamadas RPC. Se importará también tuplas.h para usar las implementaciones que se ha hecho.

En tuplas.x se ha definido una especificación para un programa RPC llamado "TUPLAS". Esta especificación incluye métodos como init_tuplas(), set_value_tuplas(message msg), get_value_tuplas(int key), modify_value_tuplas(message msg), delete_key_tuplas(int key), y exist_tuplas(int key), cada uno asociado con un identificador único en la versión TUPLASVER. Se creó también una estructura llamada message que se lo asignan a los métodos que necesiten más datos que la key como set_value y como variable de respuesta todos recibirán un int excepto get_value que recibirá la estructura message de vuelta.

Se crearon tuplas_svc.c, tuplas_client.c, tuplas_clnt.c, tuplas.h y tuplas_xdr.c que no se tocaran. El makefile que se genera no se utilizará y se usará la implementada por nosotros.

Lo único que se modificará es tuplas_server.c, Se implementa la lógica del servidor para manejar las operaciones definidas en tuplas.x. Primero, se establece un mutex para garantizar que las operaciones se realicen de manera segura en un entorno multihilo. Cada función del servidor corresponde a una operación definida en tuplas.x, como inicializar, establecer valor, obtener valor, modificar valor, eliminar clave y verificar existencia.

Cada función adquiere el mutex antes de realizar la operación respectiva y lo libera después. Esto asegura que las operaciones se ejecuten de manera exclusiva y evita conflictos de acceso a datos compartidos. Los parámetros de entrada de las funciones se manejan localmente y se pasan a las funciones correspondientes del sistema para su ejecución. Los resultados se recopilan nuevamente en estructuras adecuadas para enviar respuestas al cliente.

Se utiliza el tipo bool_t para indicar el éxito o fracaso de las operaciones. Además, se manejan los errores que puedan surgir durante la ejecución de las operaciones, garantizando una respuesta adecuada al cliente en caso de problemas. Para procesar las funciones se importará file.c.

El protocolo de aplicación utilizado para la comunicación entre el cliente y el servidor se basa en un modelo de solicitud-respuesta, donde el cliente envía una solicitud al servidor y espera una respuesta.

El diseño general de la comunicación entre claves.c y el servidor es el siguiente:

Inicialización del Cliente y Servidor:

1. Cliente:
   - El cliente inicia llamando a la función `init()` para establecer la conexión con el servidor.
   - Se obtiene la dirección IP y el puerto del servidor desde las variables de entorno.
   - Se crea un socket y se conecta al servidor utilizando la dirección IP y el puerto obtenidos.

2. Servidor:
   - El servidor espera a que los clientes se conecten y envíen solicitudes.
   - Utiliza la función `init_tuplas_1_svc` para inicializar las tuplas y permitir que los clientes realicen operaciones.

Pasos para Procesar una Solicitud:

3. Cliente:
   - El cliente solicita una operación al servidor, como establecer, obtener, modificar o eliminar una tupla.
   - Prepara los datos necesarios para la operación y envía la solicitud al servidor a través del socket.

4. Servidor:
   - Cuando el servidor recibe la solicitud, adquiere el mutex para asegurar la exclusividad en el acceso a los datos compartidos.
   - Utiliza la función correspondiente (`set_value_tuplas_1_svc`, `get_value_tuplas_1_svc`, `modify_value_tuplas_1_svc`, `delete_key_tuplas_1_svc`, `exist_tuplas_1_svc`) para procesar la solicitud.
   - Ejecuta la operación utilizando los datos proporcionados por el cliente y actualiza los datos compartidos si es necesario.

5. Cliente:
   - Espera la respuesta del servidor.
   - Cuando recibe la respuesta, interpreta el resultado y procede en consecuencia.
   - Si la operación se realiza con éxito, el cliente continúa con su flujo de trabajo.

- En caso de un error o fallo en la operación, el cliente puede tomar medidas adecuadas, como volver a intentar la operación o notificar al usuario.

Finalización de la Solicitud:

6. Cliente:
   - Una vez que ha procesado la respuesta del servidor, cierra la conexión y libera los recursos utilizados.

7. Servidor:
   - Después de procesar la solicitud y enviar la respuesta al cliente, libera el mutex y queda a la espera de nuevas solicitudes.

Este proceso se repite cada vez que un cliente realiza una solicitud al servidor, asegurando que las operaciones se realicen de manera segura y eficiente en un entorno multihilo.

Compilación

Se utiliza un Makefile para compilar el cliente y el servidor, junto con una librería compartida llamada "libclaves.so". Las principales funciones del Makefile son establecer variables para el compilador, las banderas de compilación y enlazado, y las bibliotecas necesarias. Además, se crea un directorio llamado "lib" para generar la librería dinámica "libclaves.so" utilizando el comando `make` en la terminal. Ejecutar `make` en la terminal sería lo ideal para compilar el programa.

Se ha utilizado `.PHONY` para las palabras "all" y "clean", lo que significa que cuando se ejecute `make all`, se compilaron todos los programas, incluso si ya existen archivos llamados "all", y cuando se ejecute `make clean`, se borrarán los archivos con "clean" en su nombre, incluidos los archivos generados por la compilación.

Pruebas

Se han elaborado en total 14 tests. El archivo clientes.c ha sido ampliado con respecto a la anterior práctica, puesto que los errores encontrados de comunicación y concurrencia han sido resueltos. Los tests se han basado en los requisitos del enunciado y se han organizado en base a las funciones que involucran:

- **test_init**() -> comprueba la correcta ejecución de la operación init
- **test_set_value_nokey**() -> comprueba que el sistema inserta correctamente una tupla cuando no existe la tupla con la key especificada de antemano
- **test_set_value_exist_key**() -> en contraste con la anterior, en este caso ya tenemos una tupla con la key especificada, por lo que la operación no se puede dar

- **test_get_value**() -> en este caso, como la key existe, el sistema retorna los datos conseguidos de la tupla del servidor y los imprime por pantalla
- **test_get_value_nokey**() -> al contrario que el caso de *test_set_value_nokey*, con esta operación, cuando no existe una tupla con esa key en el servidor, el sistema debe retornar error al no encontrarla y no poder conseguir los datos
- **test_modify_value**() -> en este caso, como tenemos una key que corresponde con una tupla, podemos modificar los valores de esa tupla en el lado del servidor, por lo que este test no debe dar error
- **test_modify_value_nokey**() -> al igual que con *test_get_value_nokey*, cuando no tenemos una key que corresponde con una tupla del servidor, el sistema deberá mandar un error de vuelta al cliente
- **test_delete_key**() -> este test comprueba que cuando una tupla del servidor coincide con la key mandada por el cliente, el sistema debe borrar esa tupla, por lo que el test debe ejecutarse correctamente
- **test_delete_key_nokey**() -> en cambio, cuando le proporcionamos una key inexistente en las tuplas del servidor, el cliente debe recibir un error, ya que no se puede eliminar una tupla que no existe
- **test_exist_really**() -> el nombre de este test viene de la necesidad de distinguir entre una correcta ejecución del exists cuando realmente existe la tupla y una correcta ejecución del exist cuando la tupla no existe. Ambos casos son correctos, pero su significado es totalmente distinto. En este test comprobamos el caso 1, cuando realmente existe la tupla al proporcionarle una key correcta
- **test_exist_no**() **->** como se ha comentado anteriormente, en este segundo caso no existe la key en el lado del servidor, por lo que el sistema ejecuta la petición correctamente pero retorna negativo

- **test_test_modify_value_big_nvalue2()**-> este test es parecido al test anterior, solo que con otra operación distinta, aunque el resultado va a ser el mismo, sea cual sea la operación. No podemos tener un nvalue más grande de 32, por lo que el sistema va a terminar la ejecución cuando detecte un nvalue inválido
- **test_set_value_big_nvalue2**() -> cuando tenemos un valor de nvalue por encima de 32, el programa para su ejecución con un *exit*, por lo que esta prueba ha sido comentada en el cliente para que los demás tests puedan correr sin pararse
- **test_set_value_big_value1()->** otro caso de parámetros incorrectos de las funciones es este último test, en el que se comprueba qué es lo que ocurre si tenemos un value1 más grande de 256 caracteres. Al igual que en los tests de está índole, el sistema nos va a echar de la ejecución, resultando en un error
-