Sistemas Distribuidos

Grado de Ingeniería en Informática
Curso 2023-2024

Ejercicio Evaluable 1

Integrantes del Grupo:
María Isabel Hernández Barrio (100472315)
Blanca Ruiz González(100472361)

En este proyecto hemos implementado un servicio distribuido utilizando colas de mensajes POSIX en C. El objetivo es el almacenamiento y gestión de elementos clave-valor1-valor2, donde la clave es un número entero y los valores asociados son una cadena de caracteres y un vector de elementos de tipo double. El sistema consta de un servidor concurrente encargado de gestionar las operaciones sobre los datos y una librería con todas las funciones para acceder a los servicios del servidor y cuyo uso se pone en práctica desde un cliente.

Para compilar el programa se hace uso de un Makefile, que genera los ejecutables "servidor" y "cliente" y la librería con las funciones desarrolladas. Hemos desarrollado los siguientes ficheros:

- ## Desarrollo del Servidor (servidor.c) :

El servidor se encarga de recibir las estructuras de datos que almacenan las tuplas clave-valor1-valor2, en forma de un struct petición, y procesarlas mediante threads para devolver un resultado en un struct respuesta. Nuestro servidor guarda los datos (cada estructura petición) en ficheros dentro del directorio llamado "tuplas", creado por el mismo servidor, cada nuevo fichero que guardamos ahí corresponde a un elemento y lleva como nombre la clave. Para gestionar estos datos, utilizamos varias funciones para crear el directorio, vaciarlo, guardar datos, leer esos datos, modificarlos, eliminarlos y comprobar la existencia de un fichero con una clave dada.

En el main() se inicializan las colas de mensajes, mutex y variables relacionadas con los threads y se crea un bucle infinito donde el servidor espera continuamente la recepción de mensajes de los clientes. Cuando se recibe un mensaje, se crea un nuevo hilo para tratarlo y se espera a que el hilo actual haya hecho una copia del mensaje para poder recibir el siguiente. Los hilos llevan el atributo "detached" porque la función main no va a necesitar ningún recurso o respuesta de ellos. Al final del programa, se cierran las colas de mensajes y se destruyen los mutex y variables de condición.

La función que ejecutan los hilos es tratar_mensaje(void *mess). Primero, se copia el mensaje a una estructura petición local, proceso que debemos proteger con un mutex para evitar condiciones de carrera, y se define una estructura respuesta. Después, llama a la función correspondiente al código de operación de la petición recibida (el mensaje). Estas funciones (las que realizan las tareas mencionadas en el primer párrafo) normalmente devuelven un 0 si han completado su tarea correctamente y un -1 en caso de error, y ese valor se guarda en la variable "todo_ok" de la estructura respuesta, que se envía a través de la cola del cliente, que se conoce gracias a la variable "q_name" del struct petición. En concreto, la función leerDatosFichero devuelve la estructura completa, ya que se deben devolver al cliente los datos correspondientes a una determinada clave (el correspondiente a get_value). Por otro lado, la función existeFichero (que corresponde al exist) devuelve un 1 si el fichero con "x" clave existe y un 0 si no. Finalmente se cierra la cola cliente y se sale del hilo.

- ## Implementación de los Servicios (claves.c) :

Los servicios ofrecidos a los clientes se implementan en el archivo claves.c. Consisten en las funciones init, set_value, get_value, delete_key, modify_value y exist, que incluiremos en la librería. Las operaciones se realizan contactando con el servidor a través de colas de mensajes POSIX.

Las funciones init, set_value, get_value, delete_key, modify_value y exist llaman a la función mandar_peticion. Esta función se encarga de enviar una petición al servidor a través de colas de mensajes POSIX. Toma como parámetros la operación a realizar (op), la clave asociada a la operación (clave), el valor 1 (value1), la longitud del vector de valores 2 (N_value2) y el vector de valores 2 (V_value2). La mayoría de los servicios que se piden no necesitan todos estos valores, nada más la operación y la clave, por lo que en el resto de parámetros les pasamos valores nulos o vacíos. Del servidor se recibe una estructura struct respuesta que mandar_peticion devuelve a la función que la llamó, que guardará los valores obtenidos en las direcciones recibidas por parámetros únicamente en el caso de get_value, y devuelve el valor de todo_ok. Además, en mandar_peticion comprobamos los rangos de value1 y N_value2.

- ## Desarrollo del Cliente (cliente.c) :

Primero, en el main() se declaran las variables necesarias para almacenar los valores 1 y 2, así como una variable para almacenar el número de elementos del vector de valores 2.

Después, se llama a la función init() para inicializar el servicio de elementos clave-valor1-valor2. La función set_value() inserta un elemento clave-valor1-valor2 en el servicio. Con get_value() obtenemos los valores que acabamos de guardan para comprobar que realmente se guardó bien y se imprimen por pantalla. Seguidamente, llamamos a modify_value() para modificar los valores de la clave anterior y tras utilizar otro get_value() de la misma clave se imprime el resultado. Además, comprobamos el funcionamiento de exist() cuando la clave dada sí existe. Después llamamos a delete_key() para borrar nuestro elemento y volvemos a hacer un exist() para comprobar que también funciona para una clave ahora inexistente. Después hacemos un test más para set_value() y inicializamos de nuevo el sistema (init) para ver que el directorio “tuplas” se reinicia, es decir, se vacía. Esto debería haber borrado el fichero de la clave recién guardada, así que lo comprobamos con otro exist(). Finalmente hacemos un set_value con valores fuera de rango para chequear que salta el error devolviendo un -1.

Todos los test han dado un resultado positivo.

```
isaplus@gram0:~/Documentos/Distribuidos/P1$ ./cliente
## Test INIT ##

Init OK.

 ## Test SET_VALUE ##

 Set_value OK.

 ## GET_VALUE TEST ##

Valor1: Hola
N_valor2: 3
V_valor2[0]: 1.000000
V_valor2[1]: 2.000000
V_valor2[2]: 3.000000
Get_value OK.

 ## MODIFY_VALUE TEST + EXIST ##

Valor1: Adios
N_valor2: 3
V_valor2[0]: 4.000000
V_valor2[1]: 5.000000
V_valor2[2]: 6.000000
Result of Exist should be 1: 1
Modify Value OK.

 ## DELETE_KEY TEST + EXIST ##

Result of Exist should be 0: 0
Delete_key OK.

 ## DELETE_KEY TEST 2 - Clave inexistente ##

Error en el delete_key 2 ( OK. ERROR ESPERADO)

 ## SET_VALUE TEST 2 ##

Set_value OK.

 # INIT TEST 2 + EXIST ##

Result of Exist should be 0: 0
Init 2 TEST OK.

 ## SET_VALUE TEST 3 - Valor N_value2 fuera de rango

N_value2 fuera de rango
: Success
 Error en el set_value ( OK. ERROR ESPERADO)

Fin de pruebas!
```

- **Mensaje.h :**

En mensaje.h definimos las estructuras para las peticiones y las respuestas.

En struct peticion, cabe destacar la variable op para guardar el tipo de operación que la petición requiere. 0 corresponde a init (vaciarDirectorio), 1 corresponde a set_value (guardarDatosFichero), 2 es para get_value (leerDatosFichero), 3 para modify_value (modificarValores), 4 es para delete_key (eliminarFichero) y 5 es para exist (existeFichero). Además, es necesario incluir el nombre de la cola del cliente que hace la petición, para que después el servidor sepa a qué cliente devolverle el resultado.

En struct respuesta la variable todo_ok se usa para confirmar que la operación ha sido realizada con éxito (0) o si ha habido algún problema (-1), o si existe o no el elemento con la clave dada en el caso de exist(), que devuelve 1 si existe y 0 en caso contrario.

- **Makefile :**

El archivo Makefile contiene reglas para compilar y enlazar el código. Se enfoca en la creación de la biblioteca dinámica libclaves.so, que contiene la implementación de los servicios clave-valor definidos en claves.c. Esta biblioteca es enlazada por el cliente para acceder a los servicios proporcionados por el servidor. Las reglas también incluyen la compilación de servidor.c y cliente.c para crear los ejecutables servidor y cliente respectivamente, usando la biblioteca libclaves.so. La regla clean se utiliza para eliminar los archivos generados durante la compilación y el enlazado.

Así, para compilar utilizamos “make” y para ejecutar hacemos un “./servidor” y un “./cliente”.