Sistemas distribuidos: ejercicio 1

Colas de mensajes

Belén Gómez Arnaldo 100472037

Ana Maria Ortega Mateo 100472023

17 de marzo de 2023

Descripción de la arquitectura.

La arquitectura de nuestro código se divide en dos partes fundamentales; el lado del servidor y el lado del cliente.

- Servidor: el servidor es el encargado de gestionar las peticiones de los clientes y se encarga de almacenar la información que se va añadiendo según las acciones que los clientes realizan. En “servidor.c”, se encuentra la implementación de todas las funciones. En la cabecera de “servidor.h” definimos la estructura de una lista enlazada, que es la que utiliza el servidor para ir guardando las tuplas de los clientes según las acciones realizadas.
- Cliente: el cliente está compuesto por una biblioteca (“libclaves.so”), por dos archivos de código(“clienteX.c” y “claves.c”) y una cabecera de “claves.h” donde se definen todas las funciones que el cliente puede realizar. En “clienteX.c” no se implementa nada de código de las funciones, sirve solamente para que el cliente pueda escribir las funciones que quiere realizar, junto con los parámetros necesarios para realizar la función. En claves.c es donde se realiza la lógica de las funciones por parte del cliente y donde se crea la petición y se envía a la cola del servidor. La biblioteca sirve para generar el ejecutable y así poder enlazar claves.c con el cliente.

Para la comunicación entre cliente y servidor hay una cabecera compartida “mensajes.h”, en la que se define las estructura tanto de petición como de respuesta que las colas de mensajes tanto del servidor como del cliente utilizan.

Descripción de los componentes.

Para la implementación de las funciones y el desarrollo de la práctica se han desarrollado tres componentes principales.

1. ClienteX.c : El cliente podrá introducir cualquier acción de las que están definidas y se ejecutará las funciones correspondientes a dicha acción.
2. Claves.c : Se encarga de manejar las acciones que realiza el cliente y mandarle la petición al servidor. Está compuesta por seis funciones diferentes; init, set_value, get_value, modify, delete y exist. Todas comparten estos pasos:

   -Se crean las colas del servidor y cliente y las estructuras tanto de petición y respuesta.

   -Se abren las colas con los flags y atributos predefinidos.

-Se asigna a petición la operación que se va a realizar y en caso de que se necesite los demás datos de petición como key, valor1, N y valor2, también se almacenan en la estructura, para así poderlo mandar a la cola del servidor (realizando un send con los valores necesarios). Además, se almacena el nombre de la cola del cliente (queuename) en la estructura de petición (pet.q_name).

-Después de mandarlo espera a que le llegue a la cola del cliente una respuesta (con recieve) por parte del servidor. Según la acción realizada, como por ejemplo en get value, se guardarán los valores de la estructura de respuesta para que el cliente pueda conocer los datos que ha pedido.

-Por último, se cierran tanto la cola del servidor como la del cliente y se destruye la cola del cliente.

Tanto los errores que se pueden producir como ciertas limitaciones a la hora de realizar las funciones, como por ejemplo que N sea mayor de 0 en set_value, se tienen en cuenta en este componente en las funciones correspondientes. En caso de error se notifica el error producido y se cierran las colas y se destruyen si es necesario.

3. Servidor.c : En servidor.c se realiza la implementación de funciones para gestionar las peticiones y almacenar las tuplas. Parte de una función main en la que se inicializan los mutex y las variables de condición para mantener concurrente el sistema, y se crean los atributos del hilo. Los hilos son independientes(detached), no dependen el uno del otro por lo que no tendrán que realizar un join para esperar a que los demás hilos se ejecuten. Después se crea la cola del servidor, y se realiza un bucle infinito, en el que el servidor espera a recibir las peticiones de los clientes. Cada solicitud recibida se procesa en un hilo, que ejecuta la función atender petición. Hemos considerado que es mejor un sistema de hilos bajo demanda a crear un pool de hilos porque de esta forma solo se crean hilos cuando el servidor recibe una petición y no hay hilos inactivos consumiendo recursos. Además, de esta forma el sistema se puede ajustar mejor a la carga de trabajo y no llegará al límite de hilos cuando haya muchas peticiones.

   En atender_peticion , se utiliza un mutex y una variable de condición para proteger la copia de la estructura petición antes de procesarla. Después de realizar la copia se procede a, según la operación asignada, realizar la función correspondiente mediante el manejo de listas enlazada:

   -Init: Inicializa la lista y libera la memoria en caso de que hubiera una lista existente.

   -Set value: Añade una tupla a la lista con los valores correspondientes.

-Get value: En caso de que la clave que se quiera acceder existe, devuelve en la estructura de respuesta los valores de la clave almacenada.

-Modify_value: En caso de que la clave coincida con alguna de las tuplas almacenada en la lista, modifica sus valores. Libera los valores anteriormente guardados y asigna el espacio correspondiente de los nuevos valores.

-Delete: Elimina de la lista el valor con la clave correspondiente, siempre y cuando exista y no esté vacía. En caso de que se elimine atribuye los valores de siguiente del nodo anterior adecuadamente y libera la memoria de la tupla eliminada.

-Exist: Comprueba si la clave se encuentra almacenada en la lista.

Todas las funciones guardan en la estructura de respuesta su estado y devolverá 0 si todo ha ido bien y -1 en caso de error, exceptuando exist que devuelve 1 si existe y 0 si no existe. Están protegidas por un mutex (funciones) para que se realice las funciones de forma atómica y que no sea posible que dos hilos quieran acceder a una misma función a la vez. Finalmente, se envía al cliente la respuesta, previamente habiendo abierto su cola y el hilo abandonará la función. Se tiene en cuenta todos los errores posibles que se realizan durante todo este proceso y si es necesario se cierran y destruyen las colas correspondientes.

Ejecución.

Para ejecutar el programa hemos creado varios archivos de clientes donde se llaman a las funciones y se crean sus argumentos. Para ejecutar cada archivo solo es necesario poner en terminal el comando "*./clienteX*", siendo X el número del cliente que se quiere ejecutar. Los códigos de los clientes se pueden cambiar para hacer diferentes funciones, nosotras hemos creado el código que consideramos adecuado para hacer una buena batería de pruebas. Si se quiere cambiar el código de los clientes hay que ejecutar después el comando "make". Para ejecutar varios clientes de forma concurrente basta con usar el comando "*clienteX & clienteX*".

Al ejecutar el comando make se crea también el ejecutable del servidor, por lo que solo hace falta poner en terminal el comando "*./servidor*". El servidor se ejecuta continuamente hasta que se pare manualmente en la terminal. También se crea el archivo de *libclaves.so*.

Pruebas.

Comportamiento secuencial

Para comprobar el funcionamiento secuencial del programa usaremos el archivo de *cliente.c*. Cuando se llama a cualquiera de las funciones se comprueba el valor que devuelven. En caso de que sea -1 se imprimirá un mensaje de error. Para depurar el código del servidor y comprobar el funcionamiento de la lista enlazada se ha incluído una función de *printLista* que imprime la lista en el momento en el que se llama. En el código actual no se llama a esta función, pero si se quiere comprobar se puede hacer desde el servidor. Se han hecho las siguientes pruebas, en cliente1 y cliente2:

- En cliente1 se comprueba todas las funciones(init, set_value, modify, get_value, delete y exist), comprobando que se va realizando las funciones correctamente y se muestra por la terminal los valores esperados. Realizamos en primer lugar un init, después dos set values, comprobamos con get value la clave añadida anteriormente y comprobamos los valores devueltos. Hacemos un modify para comprobar que se cambian los valores en la lista enlazada y un exist para comprobar que la clave existe. Finalmente eliminamos una de las tuplas y comprobamos con otro exist que ya no se encuentra almacenada.
- En cliente2 realizamos pruebas para la depuración de errores y poder comprobar que son detectados correctamente.

Comportamiento concurrente

Para comprobar el comportamiento concurrente, lanzamos el primer cliente(cliente1) y hemos creado otro en este caso cliente3, que se ejecutará de manera concurrente. Ambos realizan distintas acciones, si un cliente realiza el set de una clave, el otro puede acceder a ella mediante un get. De la misma manera, si un cliente elimina una clave, el otro cliente no podrá acceder a ella. En el enunciado no se especifica el orden exacto en el que se tienen que hacer las funciones de los clientes. No se especifica que se tenga que intercalar o que se tenga que hacer primero todo un cliente y después el siguiente. Por ello, no hemos definido un orden concreto, pero sí hemos utilizado mutex para realizar las funciones de forma atómica y mantener la consistencia del sistema.