Sistemas Distribuidos

Ejercicio Evaluable 1: colas de mensajes.

Grado de Ingeniería en Informática. Curso 2018-2019.

Arquitectura de computadores

Departamento de Informática

Autores:
Juan José Garzón Luján. (100363861)
Carlos García Cañibano. (100363813)

Estructura simplificada de los ficheros y código del proyecto

En **claves.c** tenemos todas las acciones que pueden hacer los clientes en el servidor y todas ellas siguen una misma estructura:

- Creamos la cola del cliente, cada con el id que caracteriza al cliente, y abrimos la cola del servidor.
- Se rellena los datos necesarios de la petición que se enviará al servidor y se esperará su respuesta.
- En caso de cualquier error o de no haberse podido realizar la petición correctamente, se imprimirá un mensaje por pantalla con el problema.

Hemos decidido utilizar una lista enlazada para almacenar los elementos con clave-valo1-valor2, que nos permitirá almacenar todos los elementos que queramos sin tener problemas de almacenamiento. Todo ello esta definido en **linked_list.h**, la lista enlazada estará creada con tripletas que tendrán una clave única, dos valores y apuntarán a otra tripleta(enlace).

En **linked_list.c** tenemos todas las operaciones que hará el servidor, se apoyará en **linked_list.h** para usar las tripletas y definir las funciones que se harán en la lista enlazada.

Todas las operaciones devolverán un -1 si no se han podido hacer correctamente, sino, un 0. Excepto **elements()**, que devolverá el número de elementos.

Función **insert():** Recibe la tripleta a insertar en la lista enlazada que recibe. Su mecanismo es sencillo, si la lista esta vacía inserta la tripleta, sino, deberá verificar que la clave no existe ya en la lista y una se verifique que non está se inserta la nueva tripleta. En el caso de que la clave esté ya, no se podrá insertar la tripleta.

Función **show():** Recibe la lista y la recorrerá imprimiendo los valores de cada tripleta.

Función **erase():** Recibe la lista y elimina todas sus tripletas.

Función **elements():** Recibe la lista y retorna cuantas tripletas hay en ella.

Función **search():** Recibe la clave para buscar la tripleta en la lista que recibe. Devolverá una tripleta con sus valores, en caso de no encontrarlo, devolverá en la llave un “NOT_FOUND”.

Función **modify():** Recibe la clave de la tripleta y los valores nuevos de ella que se buscará en la lista recibida. Si no está la clave en la lista, no se podrá modificar la tripleta y en cambio si existe, le actualizará los nuevos valores.

Función **delete():** Recibe la clave para buscar la tripleta en la lista recibida. Si no existe, no se podrá eliminar la tripleta, pero si existe, habrá que comprobar si la primera cadena tiene la clave y si no comprobaremos las demás tripletas hasta encontrara la que hay que eliminar.

Función **verify():** Recibe la clave de la tripleta a buscar en la lista recibida. En caso de que exista retornará un 0, si no un -1.

En **servidor.c** tenemos todo lo referente a como gestionará las peticiones el servidor. Por cada petición, crearemos un thread y para evitar de la carrera de datos tenemos un lock para hasta que no hayamos hecho una copia de la petición y asi atender a la siguiente. En la función **tratar_mensaje()**

recibimos la petición y hacemos una copiar para trabajar con ella, dependiendo del código de operación haremos una acción en la lista del servidor. Para proteger la lista hacemos un lock hasta que no hallamos terminado operación con ella.

En **mensajes.h** tenemos los dos tipos de mensajes utilizados:

- Request: Es la petición que hace el cliente al servidor con un código de operación, el nombre del cliente y una tripleta.
- Answer: Esa la respuesta que envía el servidor al cliente, consta de un código con la respuesta de la acción y una tripleta, que se usará por ejemplo a la hora de una petición de **get_value()**.

Esto será utilizado tanto por **cliente.c** como por **servidor.c.**

Acerca de la compilación del proyecto y ficheros adicionales

Para la compilación del proyecto se ha proporcionado un fichero makefile que puede ser consultado. Básicamente establece las directivas y dependencias de compilación. Adicionalmente también se le ha añadido la funcionalidad “clean” que permitira con el mandato “make clean” limpiar los ficheros generados durante la compilación. Con una única invocación de “make” todo debería estar listo para ser ejecutado y probado.

También se añade una carpeta con logs de ejecución de algunas pruebas realizadas, por si resultase de utilidad a la hora de revisar la funcionalidad del proyecto desarrollado. Su nomenclatura y código de versión serán definidos en el apartado de pruebas.

Apartado de pruebas

Para probar el proyecto se han creado hasta seis clientes distintos con funcionalidades entremezcladas simulando un posible contexto de ejecución realizado por múltiples usuarios simultáneos.

Por ejemplo, cliente1 crea 100 tripletas y elimina 25 del tramo 25-50. En cambio, cliente2 modifica 25 tripletas del tramo 1-25 y más tarde consulta sus valores actualizados. Finalmente, cliente3 comprueba la existencia de todas las tripletas creadas por cliente1 para comprobar el estado de los datos del servidor en un momento dado.

Se han ejecutado en diferentes secuencias. Por ejemplo: cliente1-cliente2-cliente3, cliente2-cliente1-cliente3 y se ha experimentando retrasando o adelantando la ejecución de cliente3.

Los resultados obtenidos indican que las operaciones se realizan de manera correcta concurrentemente. Si una tripleta no ha sido creada o se ha borrado, las consultas de existencia indican que no existe. Si una tripleta es modificada, sus nuevos valores son perfectamente obtenidos por las funciones pertienentes y, cliente3, indica que las tripletas existen o no dependiendo del instante de ejecución del mismo. Hemos registrado los resultados de dos ejecuciones en ficheros de texto para no hacer excesivamente grande este documento y pueden ser consultados para obtener mayores detalles. No hemos observado comportamientos anómalos. También se ha utilizado la función sleep(1) para poder trazar de mejor manera la ejecución de todos los procesos. El orden concreto que registran los logs es cliente1-cliente2-cliente3 y cliente2-cliente1-cliente3.

Para el resto de pruebas proponemos las siguientes secuencias de ejecución con los clientes que nos faltan:

V1: cliente 4 5 6
V2: cliente 5 4 6

El cliente 4 inserta tripletas, intenta conseguir valores asociados a tripletas e intenta saber si una tripleta existe o no.
El cliente 5 solo inserta 100 tripletas.
El cliente 6 modifica tripletas y elimina tripletas.

En la primera versión hemos ejecutado los clientes en el orden 4, 5 y 6.
Los resultados son coherentes, en el cliente 4 puede insertar todos excepto la primera tripleta ya que está insertada por el cliente 5, las tripletas que no puede conseguir es porque el cliente 5 todavía no ha llegado a insertar todas las tripletas restantes del 1 al 100. Lo mismo sucede con las tripletas que existe y que no. En el cliente 5 puede insertar todas menos las 6, 11, y así de 5 en 5, ya que lo insertó el cliente 4, pero hay excepciones en la 36, 45 y 56, ya que son borradas por el cliente 6. El cliente 6 puede editar todo porque ya están las tripletas en el servidor, en cambio al borrar solo están las del cliente 4. El cliente 5 imprime 100 elementos ya que es la cantidad creada, en cambio el cliente 6 imprime 45 porque no se ha completado el proceso del cliente 5.

En la segunda versión hemos ejecutado los clientes el orden 5, 4 y 6.
Los resultados también son coherentes, en el cliente 4 puede esta vez insertar esta vez la primera ya que el init esta hecho después del cliente 5 haber insertado. Pasa lo mismo que la anterior vez a la hora de decir si existe o no una tripleta, o pueda conseguir tripletas o no. En el cliente 5 no hay diferencias. En el cliente 6 la diferencia es que no puede modificar la tripleta 1 ya que se hace el init en el cliente 4 que va antes que él.