Sistemas Distribuidos

Implementación de sockets

Nathalia Moniz Cordova 100471979

Alberto García de la Torre Fernández del Moral 100472039

7 de abril del 2024

Diseño del programa

Este segundo ejercicio evaluable ha consistido en transformar el proyecto del primer ejercicio evaluable para que ahora la comunicación entre cliente y servidor funcionase con sockets. Por tanto, los cambios realizados en el código no han sido tan extensos como los realizados en la primera práctica. Las partes que han sufrido cambios significativos con respecto a la práctica anterior han sido el servidor.c y la parte de claves.c, sin embargo, pasaremos igualmente a comentar cada parte del programa solo que de manera más breve.

claves.c

En este archivo se encuentran las implementaciones de cada una de las funciones que se encargan de comunicarse con el servidor a través de los sockets y que a su vez permitirán la ejecución de las operaciones de inicializar el servicio (**init**), establecer valores para claves específicas (**set_value**), obtener valores asociados con claves (**get_value**), modificar valores (**modify_value**), eliminar claves (**delete_key**) y verificar la existencia de una clave específica (**exists**).

En cada función se asigna el operador y la llave sobre la que se va a realizar la acción. Dependiendo del tipo de acción, también se tomarán más argumentos para el valor 1 y el vector de valores 2, incluyendo su tamaño. Se crea un socket que se conecta con la ip del servidor, y posteriormente se envía cada valor necesario para realizar la acción de manera individual. Una vez se han enviado todos los valores necesarios, el programa permanecerá en espera hasta recibir de vuelta todos los datos que el servidor debe enviar. Por último, transformará los datos provenientes de la red e informará al cliente del resultado obtenido.

servidor.c

Esta parte del programa se encarga de procesar las peticiones del cliente y enviarle las respuestas. El servidor utiliza sockets para manejar los valores de entrada y salida. El servidor permanecerá en un bucle constante esperando a recibir algún mensaje del cliente.

Una vez le llega el mensaje, crea un hilo para tratar la petición. Según el tipo de operador obtenido en la petición, el servidor decidirá si debe recibir más valores del cliente antes de realizar la petición. Una vez tiene todos los valores necesarios,

realiza la acción solicitada sobre la lista enlazada, y devuelve los resultados obtenidos al cliente mediante sockets.

list.c

Para el almacenamiento de los datos, hemos implementado una lista enlazada, puesto que nos facilita la inserción, eliminación y búsqueda de elementos y por su fácil manipulación e implementación. El archivo **list.c** contiene todas las funciones que puede realizar la lista enlazada, y hay que acceder a este archivo para realizar cualquier tipo de modificación o consulta sobre los datos almacenados. Existe una función distinta por cada acción que puede realizar el cliente. Además, se ha implementado la función de imprimir la lista, para poder tener un mayor control sobre los cambios realizados.

Todas las secciones críticas (donde se accede la lista y se modifica la misma) deben ser protegidas mediantes exclusiones mutuas (mutex). De esta manera, evitamos que dos hilos intenten modificar la lista al mismo tiempo y se den condiciones de carrera. Luego, una vez modificada la lista se puede desbloquear el mutex para dar paso a que otro hilo pueda llevar a cabo la función.

cliente.c

Por último, contamos con la parte del cliente que simplemente se encargará de llamar a las funciones, ya que todo el manejo de sockets se hace en **claves.c** y en **servidor.c**. El código del cliente es prácticamente idéntico al que había en el primer ejercicio evaluable, ya que no ha requerido de cambios adicionales. Hemos añadido la impresión de los valores obtenidos mediante la función **get_value**, ya que anteriormente no lo realizábamos y es una parte importante, ya que así el cliente puede ver que los valores se han obtenido y guardado correctamente.

Comunicación y protocolo

La comunicación entre los procesos cliente y servidor se lleva a cabo a través del protocolo de red TCP . Tanto el cliente como el servidor crean un socket TCP utilizando la función **socket** con el parámetro **SOCK_STREAM**, lo que indica que se utilizará TCP para la comunicación . Luego, en el lado del cliente, se establece una conexión utilizando la función **connect**, donde se especifica la dirección IP y el puerto del servidor. Luego, se envían mensajes al servidor utilizando la función **sendMessage**, que utiliza en su implementación la función **write** que se encarga de escribir los datos desde un búfer de memoria al descriptor de archivo del socket, para luego enviarlos a través del socket TCP.

Por otro lado, en el lado del servidor, se aceptan conexiones entrantes utilizando la función **accept**, creando un nuevo socket para cada cliente conectado. Posteriormente, se reciben los mensajes del cliente utilizando la función **recvMessage**, que en su implementación utiliza la función **read** para leer los datos del descriptor de archivo. El servidor procesa las solicitudes del cliente y envía las respuestas usando el mismo sistema de envío de mensajes. Este intercambio de mensajes a través de conexiones TCP garantiza una comunicación confiable orientada a conexión y bidireccional entre el cliente y el servidor.

Podemos comprender mejor cómo se lleva a cabo la comunicación entre los procesos a través de la siguiente figura.

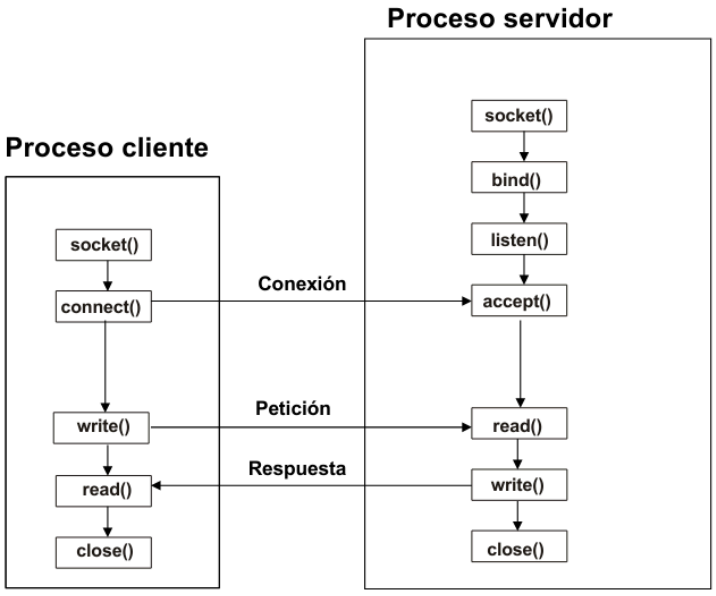

imagen 1. Esquema de comunicación de los procesos

En cuanto al protocolo de comunicación establecido entre el cliente y el servidor, este se trata de un protocolo basado en mensajes, donde cada mensaje contiene una operación específica a realizar junto con los datos necesarios para dicha operación.

En cada función se manda el número correspondiente a dicha operación y dependiendo de ésta se mandan también los parámetros que sean necesarios para que el servidor lleve a cabo la función, por ejemplo, para **delete_key** solo enviamos la operación y la clave, mientras que para **set_value** la operación, la clave, el valor 1, el número de elementos del vector y el vector. Posteriormente, cada función sin excepción recibe una respuesta del servidor que en la mayoría de los casos es simplemente un 0 o un -1 indicando que se ha realizado la operación con éxito, pero que en el caso de la función **get_value** recibimos los valores obtenidos a partir de la clave, además de ese 0 o -1. Finalmente, el servidor es responsable de recibir las solicitudes de los clientes (los datos se reciben en el mismo orden en el que han sido enviados en el cliente (claves.c)), procesarlas y responder de acuerdo con el protocolo.

El protocolo garantiza que tanto el cliente como el servidor estén sincronizados en sus operaciones sobre la lista y que puedan comunicarse de manera efectiva para realizar las operaciones solicitadas. Además, se utilizan conversiones de formato de red para garantizar la compatibilidad entre diferentes arquitecturas de sistemas.

Compilación y ejecución del programa

Estando en la carpeta **Ejercicio-evaluable-2-SSDD** hay que introducir en un terminal el siguiente comando:

```
Unset
make
```

Cuya salida debe ser similar a la siguiente:

```
Unset
gcc -Wall   -c cliente.c
gcc -Wall   -fPIC -c claves.c -o claves.o
gcc -Wall   -fPIC -c send-recv.c -o send-recv.o
gcc -Wall   -shared -o libclaves.so claves.o send-recv.o
gcc -Wall   -o cliente cliente.o -L. -lclaves -pthread
gcc -Wall   -c servidor.c
gcc -Wall   -o servidor servidor.o -L. -lclaves -pthread
```

Para la ejecución del programa se tiene que iniciar en primer lugar el servidor seguido del puerto con el siguiente comando:

```
Unset
./servidor 8080
```

Y para el cliente primero debemos decirle al sistema operativo dónde buscar las bibliotecas compartidas (archivos de bibliotecas dinámicas) necesarias para ejecutar el programa con el siguiente comando:

```
Unset
export
LD_LIBRARY_PATH=/home/username/Ejercicio-evaluable-2-SSDD:$LD_LIBRARY_PATH
```

Y finalmente ejecutar el cliente indicando la IP y el puerto y los argumentos que se desee:

```
Unset
env IP_TUPLAS=localhost PORT_TUPLAS=8080 ./cliente <op> <key> <value1>
<N_value2>
```

Los valores del vector no se introducen por terminal, sino que son generados aleatoriamente por el archivo **clientes.c**. El número de valores generados viene dado por <N_value2>.

Pruebas

Las pruebas realizadas consisten en 13 tests que se encuentran dentro de la carpeta “tests”. Cada una de las pruebas comprueba una especificación distinta del enunciado. Hay tres tipos de test: correctos (test 1-5), errores (test 6-11), y de concurrencia (test 12 y 13). A continuación describiré el objetivo de cada test:

- **test1:** comprueba el funcionamiento de la función set_value.
- **test2:** comprueba el funcionamiento de la función get_value.
- **test3:** comprueba el funcionamiento de la función modify_value.
- **test4:** comprueba el funcionamiento de la función delete_key.
- **test5:** comprueba el funcionamiento de la función exist.
- **test6:** muestra un error si no se realiza init al inicio.
- **test7:** muestra un error si se intenta introducir un valor con una llave idéntica a otro existente, o si el tamaño del vector no es válido.
- **test8:** muestra un error si el valor de la llave buscada no se corresponde a ninguno de la lista.
- **test9:** muestra un error si el valor de la llave a modificar no se corresponde a ninguno de la lista.
- **test10:** muestra un error si el valor de la llave a eliminar no se corresponde a ninguno de la lista.
- **test11:** la función exist devuelve el valor 0 en caso de no encontrar un valor en la lista que se corresponde con la llave introducida.
- **test12:** comprueba que es posible ejecutar dos procesos de cliente simultáneamente.
- **test13:** simula el comportamiento de 4 clientes que realizan una gran cantidad de cambios simultáneos en la lista.
- **aux:** inserta 100 elementos en la lista con valores de llave del 0 al 100.
- **aux2:** inserta 100 elementos en la lista con valores de llave del 100 al 200.
- **aux3:** modifica 100 elementos pares de la lista con valores de llave del 0 al 200.
- **aux4:** elimina 200 elementos en la lista con valores de llave del 200 al 0. Empieza la eliminación de los elementos por el valor número 200.

Ejecución de las pruebas

Al ejecutar las pruebas, se debe introducir el nombre del test que deseas realizar. A continuación hay un ejemplo del comando para ejecutar el test1.sh:

```
Unset
./test1.sh
```