Ejercicio evaluable 2: Sockets

Sistemas distribuidos

Abril 2023

Grado en Ingeniería Informática

Pablo Hidalgo Delgado. NIA: 100451225
Pablo Brasero Martínez. NIA: 100451247

1. Introducción

El objetivo de este ejercicio es diseñar e implementar un servicio distribuido mediante el uso de sockets. Este servicio permitirá almacenar tuplas de datos que incluyen una clave y varios valores asociados a dicha clave.
Para ello, tomaremos como base la aplicación diseñada en el ejercicio anterior (con colas de mensajes).

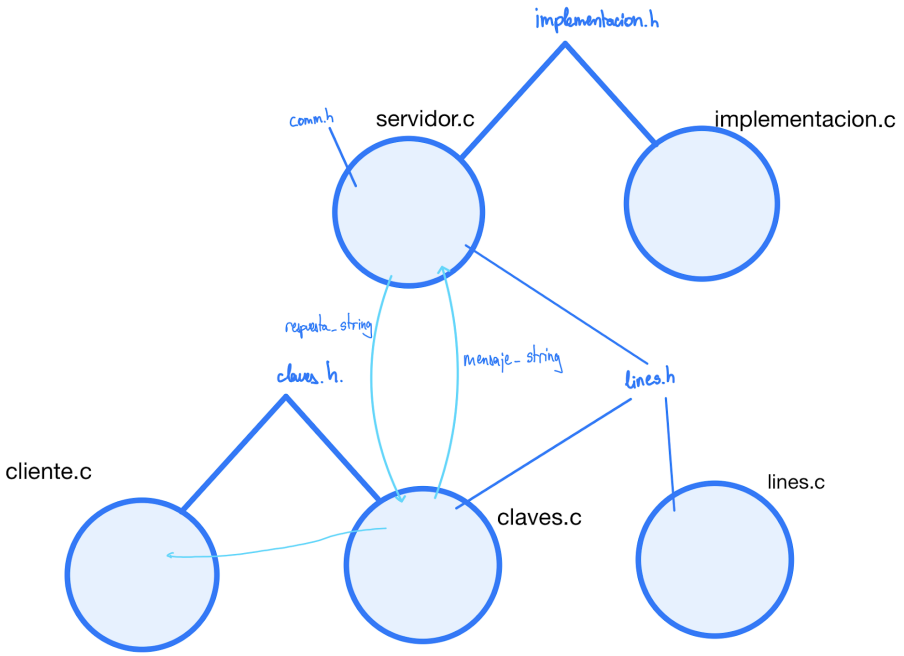
2. Diseño

Los pasos que hemos seguido para el diseño de nuestro sistema distribuido con sockets han sido:

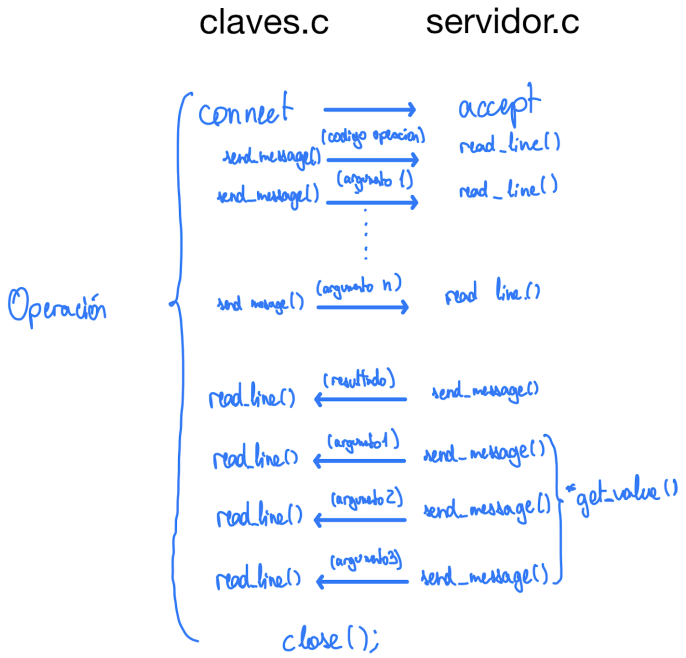
1. Identificar cliente servidor
2. Identificar las operaciones, argumentos y respuestas--> secuencia de intercambio de mensajes
3. Elegir protocolo UDP o TCP
   TCP: conexión por petición/operación ó conexión por sesión
4. Elegir/tener en cuenta el formato de los mensajes
5. Nombrado y concurrencia

- Los pasos 1 y 2 ya los realizamos en la práctica anterior. El código de operación de las funciones será un string con el nombre de la operación.
- En el paso 3, elegimos el protocolo TCP ya que es el indicado por el enunciado. Tomamos la decisión de realizar una conexión por petición/operación.
- En cuanto al paso 4, decidimos establecer el formato de los mensajes a un string para evitar convertir los datos a formato red y formato de máquina. Por tanto, los argumentos necesarios para cada operación se enviarán y recibirán como una cadena de caracteres.
- Por último, decidimos crear un hilo que maneje cada petición recibida por el servidor. Nos aseguramos por medio de mutex y variables condicionales que el copiado del descriptor de socket a la variable local dentro del hilo se haga antes de que se acepte otra conexión.

El diseño de nuestro sistema distribuido sería el siguiente:



En cuanto al envío de mensajes por medio de sockets, esta sería su representación:



Vemos cómo se han añadido 2 ficheros lines.c y su cabecera lines.h con respecto al diseño de la práctica anterior. Estos ficheros contienen las funciones destinadas a recibir y enviar mensajes de manera continua hasta que se haya enviado o recibido el mensaje completo. También contiene una función readLine() que lee un mensaje hasta que se encuentra el carácter '\0' indicando el fin de la cadena o el carácter '\n' indicando una nueva línea. Esta es la función que utilizaremos para leer mensajes de un socket.

3. Implementación

Aparte de incluir el archivo lines.c, modificamos únicamente los archivos servidor.c y claves.c con respecto a la práctica anterior.

- servidor.c

En primer lugar, creamos el socket correspondiente al servidor y le asignamos el puerto introducido como parámetro. Llamamos a la función listen para que “escuche” a posibles conexiones. Una vez se detecta una conexión, la aceptamos y creamos un hilo para manejarla. Controlamos, como hemos dicho anteriormente, que el copiado del descriptor de socket nuevo a la variable local dentro del hilo se haga antes de que se acepte otra conexión.
Dentro de cada hilo, para manejar los mensajes que se envían y se reciben, como los tamaños de memoria que debemos reservar son relativamente pequeños y disponemos de memoria suficiente, reservamos memoria de manera estática.
- La máxima longitud de las cadenas en el servidor serán:
Int: 12→ el número con más carácteres que puede representar un int en C es -2,147,483,648. El menos y el ‘\0 suman 2 caracteres más
Double: 30 → tomamos como decisión que el número máximo de dígitos que puede tener un double que se desee almacenar en el servidor es de 30
String: 255 → el enunciado nos especifica la longitud máxima
Resultado: 3 → 0 ó 1 ó -1 + ‘\0’.
De esta manera, leemos el código de operación utilizando readLine() y, dependiendo de la operación que se detecte, realizamos la operación correspondiente. Si la operación debe obtener valores como argumentos, llamamos de nuevo a la función readline tantas veces como número de argumentos.
Con el objetivo de que las operaciones se realicen de manera atómica en el servidor, debemos proteger la llamada a estas operaciones por medio de mutex. Así, antes de que el servidor realice la operación, realizamos un lock, luego llamamos a la función y cuando obtengamos el resultado hacemos un unlock del mutex
Por último, devolvemos el resultado o resultados obtenido por el socket del cliente, cerramos el descriptor del socket y salimos del hilo.

- claves.c

Para cada operación, creamos un socket y establecemos una conexión con el servidor de dirección IP el valor que se almacena en la variable de entorno IP_TUPLAS y puerto el valor que se almacena en la variable de entorno PORT_TUPLAS. Enviamos el código de operación correspondiente haciendo uso de la función sendMessage() y, si se deben pasar argumentos, los pasamos de uno en uno haciendo tantas llamadas a la misma función como número de argumentos se deban de pasar. Por último, recibimos el resultado con readLine(). Realizamos todo esto convirtiendo los formatos de los valores a cadenas cada vez que se quiera enviar un dato por el socket, y convirtiéndolos, si es necesario, a su correspondiente formato cada vez que leamos del descriptor de socket.

Cabe mencionar que, en cuanto a los posibles valores de los parámetros introducidos por el cliente, controlamos en las funciones correspondientes que la longitud de la cadena de value1 no sea mayor que 255. Los valores de int no los controlamos debido a que un int

siempre tendrá menos de 12 dígitos. Es decir, reservamos suficiente espacio para poder guardar cualquier valor representable por este tipo de datos en C.

A diferencia de la práctica anterior, esta vez hemos considerado los posibles errores dentro de las funciones (error al crear el socket, error al leer un mensaje…). En ese caso devolvemos -1.

4. Compilación

Para la compilación del programa utilizamos un archivo makefile. Simplemente ejecutando el comando make en la terminal, se generan dos ejecutables: el ejecutable del servidor que implementa el servicio y el ejecutable de cada clienteX, obtenido a partir del archivo clienteX.c y la biblioteca dinámica creada libclaves.so.
Para ejecutar el cliente, es necesario establecer la variable de entorno IP_TUPLAS con la dirección IP del servidor y la variable de entorno PORT_TUPLAS con el puerto del servidor.
Para ejecutar el servidor, debemos pasarle como argumento el puerto al que debe de ser asociado.

5. Pruebas

Para validar el funcionamiento de nuestro sistema distribuido, hemos realizado distintas pruebas.

- cliente0

En primer lugar, confirmamos que el envío y recibimiento de mensajes por medio de sockets funciona correctamente creando un cliente0 que realizará todas las operaciones posibles. Cuando ejecutamos este cliente podemos observar que se obtienen los resultados esperados.

- cliente1 y cliente2

En segundo lugar, verificamos que la paralelización y sincronización entre los clientes se realiza correctamente. Esto lo hacemos comprobando que 2 clientes pueden realizar operaciones a la vez y que los resultados se sincronizan. Para ello, dormimos a un cliente1 durante 15 segundos. Mientras este cliente1 está dormido, ejecutamos otro cliente2 que realizará una serie de operaciones y peticiones al servidor. Como cada petición es ejecutada por un hilo, las operaciones realizadas por el segundo cliente deben resultar exitosas y no debe afectar que otro cliente esté ejecutándose (dormido) y haya realizado peticiones al servidor previas. Además, cuando el cliente 1 despierte, debe poder ver los resultados realizados por el cliente 2.
Más concretamente, el cliente1 realizará un set_value de 2 claves con keys 10 y 11. Posteriormente, se dormirá durante 15 segundos. Durante estos 15 segundos, se ejecutará el cliente 2, que comprobará si existen las claves 10 y 11 y las eliminará. Cuando despierte el cliente1, verificará si las claves 10 y 11 existen y, si todo ha ido bien, el resultado obtenido deberá corresponder a que las claves no existen.
El resultado de esta prueba ha sido exitoso.

- cliente3

También hemos realizado pruebas para comprobar el caso en que se pasa como argumento a la función set_value un string de mayor longitud que la permitida (255). En este caso,se muestra un mensaje de error por pantalla y la función devuelve -1. Además, la clave no debe ser insertada en el servidor y el cliente sigue pudiendo realizar operaciones. Esta prueba está realizada con éxito en el cliente3.c.

- cliente4

Por último, hemos realizado distintos casos de prueba en los que el cliente4 realiza operaciones erróneas. Este cliente realiza operaciones ante un servidor no inicializado, es decir, sin haber llamado a la función init. Copiamos el código del cliente0 pero eliminando la operación init. Podemos observar que todos los resultados de las funciones son -1 menos exit que es 0 ya que no existe ninguna clave con la key introducida. Hemos de realizar esta prueba en primer lugar ya que los otros clientes ya inicializan el servidor.