Universidad
Carlos III de Madrid

Universidad Carlos III de Madrid
Curso 2023/2024. Sistemas Distribuidos
Practica 2.
Sockets TCP

Fecha: 7/04/2024 Entrega: Final

Grupo: 84

Alumno 1:

Nombre y Apellidos: Fernando Consiglieri Alcántara

NIA: 100472111

Alumno 2:

Nombre y Apellidos: David Andrés Yáñez Martínez

NIA: 100451958

Diseño de los archivos

Nuestra aplicación se divide en tres componentes distintos:

- Servidor: Consta de los archivos servidor.c, claves.c, list.c y lines.c. El archivo servidor.c se encarga de recibir a través del socket las peticiones de los clientes, llamar a las funciones implementadas y devolver las respuestas correspondientes a los clientes. Claves.c contiene la implementación de las funciones requeridas en la práctica, mientras que list.c es el desarrollo de la lista enlazada, adaptada para este ejercicio y utilizada para almacenar los mensajes proporcionados por los clientes. Por último lines.c incluye tres métodos proporcionados en clase que facilitan la lectura y escritura en el socket.
- Cliente: Solo incluye el archivo cliente.c, que simula la interacción de un cliente y llama a la API mediante la librería dinámica para llevar a cabo las acciones deseadas.
- Libclaves.so: Consiste en el archivo proxy.c, que funciona como la API y se encarga de crear las colas de mensajes del cliente y estructurar los mensajes que luego se pasan al cliente.

Protocolo de comunicación

El protocolo de comunicación de la aplicación sigue un enfoque simple y directo para transmitir los datos necesarios en cada solicitud. Aquí está la descripción del protocolo:

Código de operación (op): El código de operación se envía como un entero directamente utilizando la función htonl para convertirlo al formato de red antes de enviarlo. Esto garantiza que el orden de bytes sea consistente independientemente de la arquitectura del sistema.

Clave (key) y N: Al igual que el código de operación, la clave y el valor de N se convierten a formato de red utilizando htonl antes de enviarlos. Esto asegura que se envíen en el formato adecuado para su interpretación en el servidor.

Valor de cadena (v1): Dado que v1 es una cadena, se utiliza la función readLine para recibir la cadena caracter a caracter hasta encontrar el carácter nulo \0 que indica el final de la cadena.

Array de doubles (v2): Para el array de doubles v2, primero se envía el valor de N para indicar cuántos elementos se enviarán. Luego, se itera sobre el array y cada elemento se convierte a una cadena antes de ser enviado.

Valor de error: El valor de error devuelto por el servidor también se envía como un entero, y se convierte a formato de red utilizando htonl antes de ser enviado.

En resumen, el protocolo de comunicación garantiza la consistencia en el formato de los datos enviados y recibidos, lo que facilita su interpretación tanto en el cliente como en

el servidor. El uso de las funciones htonl, readLine, sendMessage y recvMessage simplifica la implementación del protocolo y garantiza la interoperabilidad en diferentes plataformas.

Forma de compilación

La compilación de nuestro proyecto es sencilla y no varía demasiado en comparación a la original:

- Primero generamos nuestra librería dinámica a partir de proxy.c. Compilamos su código objeto con la flag -fPIC para el enlazado y luego la unimos a la librería dinámica con -shared.
- Después, enlazamos esta librería dinámica con nuestro cliente.c, especificando la ubicación de la librería con las flags: -L. -lclaves -Wl,-rpath,. -lpthread.
- Finalmente, compilamos nuestro servidor, que consiste en unir los tres archivos .c que conforman nuestro servidor.

Todos los archivos .c tienen flags comunes como -Wall -Werror para garantizar un código limpio. Además, según el uso de las librerías pthread o funciones mqueue, se añaden las flags -lpthread o -lrt respectivamente.
Para ejecutar nuestros programas, basta con utilizar las reglas “runc”, que ejecuta el cliente, y “runs”, que ejecuta el servidor. Dentro de runc, tenemos runc1, runc2 y runc3 que explicaremos más adelante.

Batería de pruebas

La batería de pruebas no se ha visto afectada con respecto al ejercicio anterior, el archivo cliente.c se ha mantenido en estructura tal y como estaba en el original.

Antes de explicar la batería de pruebas que hemos diseñado, es importante resaltar que la implementación de nuestra librería no solo devuelve un entero indicando si la acción se realizó con éxito o no, sino que también ofrece detallados resultados por la línea de comando mediante printf. Estos resultados indican si la acción se completó o falló, por lo que no es necesario incluir printf's en el código del cliente. Sin embargo, en mi implementación del cliente, también he incluido printf's, lo que resulta en la impresión redundante de información en ocasiones. Es redundante, pero dado que la práctica no especificaba claramente este aspecto, lo he mantenido, ya que no afecta la funcionalidad del sistema.

Nuestra batería de pruebas consiste en un archivo llamado cliente.c que contiene tres funciones principales. Estas funciones pueden ser seleccionadas al ejecutar el programa mediante la línea de argumentos:

Si ejecutamos ./cliente 1 o make runc1, generará un cliente que probará todas las posibles acciones. Además, también se realizarán pruebas preguntando por 'get' y 'exist' cuando la clave no existe, y usando 'set' cuando la clave existe.

Si ejecutamos ./cliente 2 o make runc2, se realizarán pruebas con un cliente que desconoce cómo funciona la API y probará con mensajes no válidos, como arrays muy largos o valores de 'N' no permitidos.

Si ejecutamos ./cliente 3 o make runc3, se aplicarán pruebas con hilos. Se generarán 50 clientes que probarán acciones previsibles simplemente para evaluar la concurrencia de nuestro servidor.