Sistemas Operativos II - Laboratorio II IPC

Ingeniería en Compitación - FCEFyN - UNC

Interprocess Communication

Requisitos Previos

Para usar este programa es necesario instalar en su sistema la herramienta SQLite y sus bibliotecas de funciones, para esto abra una terminal en la carpeta del proyecto y ejecute el comando **make reqs**, el cual le solicita su contraseña e instala los requisitos.

Descripción General

En este laboratorio se presenta una aplicación servidor junto a 3 tipos diferentes de clientes, conectados a través de **sockets de internet**, que se detallan a continuación:

- **Servidor**: Da servicio a los clientes y gestiona 5 conexiones a la base de datos. Tanto los clientes como el servidor se deben desconectar con la señal SIGINT (Ctrl+C), cuyo handler notifica al servidor que se ha desconectado un cliente.
- **Cliente A**: Ejecuta una query SQL de manera repetida, en este caso la query es "SELECT * FROM Cars WHERE Price > 50000", que se envia cada 1 segundo a la base de datos y se muestra en pantalla el resultado. *Usa un socket de tipo UDP*.
- **Cliente B**: Ofrece una consola para ingresar una sentencia SQL para ejecutar en la base de datos, y se muestra en pantalla el resultado. Si se ingresa el comando "EXIT" (sin commillas) se desconecta el cliente del servidor. *Usa un socket de tipo TCP en IPv6*.
- **Cliente C**: Permite al cliente descargar una copia de la base de datos en el estado actual, con el nombre que se indique por parámetro. *Usa un socket de tipo TCP en IPv4*.

Comandos de ejecución

Servidor: ./bin/server , ejemplo: ./bin/server 8080 8081 8082 Cliente A: ./bin/client_a <IP_Servidor> , ejemplo: ./bin/server localhost 8080 Cliente B: ./bin/client_b <IP_Servidor> , ejemplo: ./bin/server localhost 8081 Cliente C: ./bin/client_c <IP_Servidor> <Nombre.db>, ejemplo: ./bin/server localhost 8082 test.db

Automatización

Se agregaron 3 scripts que permiten automatizar la instanciación de los programas clientes, para facilitar la prueba con multiples instancias del mismo.

Base de Datos

La base de datos se gestiona gracias a la librería SQLite, que permite trabajar con bases de datos en el sistema de archivos, en este caso la base de datos que usamos es una muy simple llamada **Cars.db** que se encuentra en la carpeta raiz del repositorio. Contiene una tabla con nombres de autos (sin modelo) y sus respectivos precios, y otra tabla **Logs** donde se almacenan las querys ejecutadas en la base de datos desde

el inicio del programa servidor. Nótese que esta tabla se vacía cada vez que se inicia el servidor, se puede evitar esto eliminando la linea que dropea esta tabla de ser necesario.

Gestión de las Conexiones a la Base de Datos

En un principio se implementó un **pool de conexiones** para gestionar las consultas a la base de datos, mediante la librería **Libzdb**, al final de este documento se encuentran las referencias. Esta librería permite indicar un numero de conexiones maximo, mientras tendrá en espera a todos los clientes que deseen comunicarse con la base de datos para atenderlos a su debido tiempo. Sin embargo, aunque funcionaba bien con los clientes A y C, esta implementación complicaba al cliente tipo B porque los metodos para acceder al resultado de una query dependen del tipo de dato, lo que implica conocer de antemano la base de datos, y el código queda bastante hardcodeado. Por este motivo se decidió retirar el código de Connection Pooling y gestionar las conexiones con un arreglo de 5 apuntadores sqlite, y con la **politica** de darle una **conexion aleatoria** de estas 5 al cliente que la solicita, dado que es una forma muy sencilla y estadisticamente probable de asignar conexiones disponibles a los clientes.

Links / Referencias

- Ejemplos de SQLite3: https://zetcode.com/db/sqlitec/
- SQLite3 Multithread: https://www.sqlite.org/threadsafe.html
- Libzdb: https://www.tildeslash.com/libzdb/#api
- Transferencia de Archivos por TCP Sockets: https://www.youtube.com/watch?v=12F3GBw28Lg
- SQLite
- Git Workflow