Universidad Carlos III

Sistemas Distribuídos

Curso 2023-24

Práctica 2

Sockets

Ingeniería Informática, Tercer curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)
César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)

Prof . Félix García Caballeira y Alejandro Calderón Mateos
Grupo: 81

Índice

1. Introducción

El desarrollo de este proyecto consiste en implementar una aplicación cliente-servidor, donde los diferentes clientes podrán guardar información en tuplas a través del servidor, de forma transparente. La comunicación entre el cliente y el servidor se dará a través de una interaz de Sockets. Los mensajes transmitidos deben ser independientes del lenguaje de programación.

Cabe destacar que el tratamiento de peticiones en el lado del servidor será de forma concurrente, mediante el uso de procesos ligeros.

2. Diseño

La aplicación constará de dos partes diferenciadas: los clientes y el servidor.

2.1. Cliente y Biblioteca dinámica

Siguiendo la misma aproximación que en el ejercicio anterior, el cliente no ha necesitado de cambios ya que la única parte afectada por los requisitos de este ejercicio es la comunicación. La comunicación entre el cliente y el servidor está completamente encapsulada en la biblioteca dinámica *libclaves.so*. Por esto, el único archivo modificado será *src/claves.c*, que sirve como código fuente de la bibliteca dinámica.

2.2. Servidor

En el servidor se encuentran las funcionalidades que se encargan de la comunicación con el cliente, en el fichero *src/servidor.c*, y la implementación de los servicios para las tuplas, en el fichero *src/imp_clave.c*. De nuevo, el único archivo que necesitó de modificaciones es *servidor.c* para adaptar el código a la comunicación mediante Sockets.

2.2.1. Estructura del Servidor

El servidor debe realizar dos acciones principales: aceptar la comunicación con un cliente y lanzar un hilo para gestionar dicha petición. Para la primera acción, el servidor debe estar alojado en un puerto que recibirá como argumento de programa. En dicho puerto esperará hasta recibir una petición de conexión con algún cliente. Para la segunda acción, usamos la interfaz ofrecida por *pthread.h* para lanzar un hilo independientemente que tratará la petición y comunicará al cliente la respuesta.

2.2.2. Implementación en el servidor

La implementación de los servicios requeridos está contenida en el fichero *src/imp_claves.c*. Dado que los servicios coinciden exáctamente con los del ejercicio anterior, este fichero no ha sufrido ningún cambio. Se compilará junto con el servidor.

Siguiendo la misma idea que en la práctica anterior, guardaremos las tuplas en ficheros dado su caracter permanente entre sesiones y las facilidades que presentan para la gestión de la concurrencia.

2.2.3. Concurrencia del Servidor

Seguiremos un esquema de hilos bajo demanda. Es decir, crearemos un hilo por cada petición recibida para su gestión.

El servidor contará con un hilo principal (más adelante referido como *main*) encargado de escuchar y recibir peticiones y crear nuevos hilos *detached* para la gestión de dichas peticiones. Cada uno de los hilos creados por *main* ejecutará la función `tratar_peticion()`. En esta función, copiará el socket del cliente (protegiendo la copia por un *mutex* para resolver la condición de carrera) y se llamará al servicio adecuado según el código de operación de la petición. Después, se enviará la respuesta al cliente y se cerrará la conexión.

2.3. Comunicación

La comunicación, tal y como especificamos anteriormente, se realiza a través de Sockets. Este mecanismo permite comunicar dos procesos cualesquiera en la misma red, independientemente de la arquitectura de la máquina que los ejecuta o el lenguaje de programación en el que esté escrita la aplicación. Es por esto que es importante que la información transmitida por las partes cliente y servidor de nuestro sistema sea también independiente de estos menesteres.

2.3.1. Uso de cadenas de texto

Para hacer los mensajes independientes del lenguaje de programación o arquitectura de la máquina, transmitiremos la información como cadenas de texto. Además, los mensajes serán transmitidos byte a byte para evitar la problemática presentada por la diferencia entre arquitecturas *little-endian* y *big-endian*.

2.3.2. Funciones auxiliares para la comunicación

Para facilitar la implementación de la comunicación mediante Sockets se han implementado una serie de funciones auxiliares recogidas en el fichero *src/common.c*. Estas implementan acciones repetidas tanto en el lado cliente y servidor, así como gestión de errores y envío y recepción de mensajes adaptados a las necesidades del sistema.

- `serverSocket()`: crea y devuelve un *socket* para el servidor en un número de puerto dado. Además, se ejecuta la llamada `listen()` para permitir al servidor aceptar conexiones más adelante.
- `serverAccept()`: esta función permite al servidor aceptar una conexión con un cliente.
- `clientSocket()`: crea y devuelve un socket para un cliente en el sistema. Además, este *socket* estará conectado al servidor.
- `sendMessage()`: envía un mensaje contenido en un *buffer* a través de un *socket*.
- `recvMessage()`: recibe un mensaje y lo almacena en n *buffer* dado.
- `writeline()`: se apoya de la función `sendMessage()` para enviar un mensaje hasta el final.
- `readLine()`: se apoya de `recvMessage()` para recibir un menesaje hasta el final de la cadena de texto.

3. Compilación

En esta sección se describe la forma de compilar el proyecto para generar dos ejecutables cliente y servidor. Todos los comandos que se exponen a continuación quedan recogidos en un archivo *Makefile* encargado de generar ambos ejecutables.

3.1. Biblioteca Dinámica

Tal y cómo hemos descrito anteriormente, el código fuente de la biblioteca dinámica está contenido en el archivo *claves.c*. Para compilarlo como una biblioteca dinámica se han empleado los siguientes comandos.

```
\# c digo objeto
gcc -c -fPIC claves.c
\# biblioteca din mica
gcc -shared -o libclaves.so claves.o
```

Con estos comandos generamos un archivo de código para enlazar con otro ejecutable. Este código será independiente de la posición.

3.2. Cliente

El código fuente del cliente está contenido en el archivo *cliente.c*. Este se compila enlazándolo con la biblioteca dinámica a través de los siguientes comandos:

```
\# generacion del objeto
gcc -c cliente.c

\# generar ejecutable con biblioteca din mica
gcc -L. -Wl,-rpath=. -lclaves -o cliente cliente.o
```

3.3. Servidor

4. Descripción de pruebas

Para validar el correcto funcionamiento del sistema, hemos realizado dos bloques de pruebas.

El primero, encargado de comprobar la implementación y concurrencia del sistema, reutiliza las pruebas desarrolladas para la primera parte. El resultado de estas pruebas es exitoso, tal y como esperábamos ya que no hemos modificado la implementación de los servicios.

Para el segundo bloque, encargado de comprobar la comunicación y correcta serialización de los datos, hemos conectado dos máquinas distintas (una lanzando el servidor y otra varios clientes) en una misma red LAN. El resultado de estas pruebas también ha sido exitoso.

5. Conclusiones

Este ejercicio nos ha permitido explorar una nueva aproximación para el desarrollo de aplicaciones distribuídas, la llamada a procedimientos remotos. Esta forma de desarrollo nos permite una mayor abstracción, ayudándonos de herramientas de generación de código automático.

La herramienta utilizada, aunque algo tosca, nos ofrece las mismas ventajas que el desarrollo con sockets pero con una sintaxis más sencilla y reduciendo el tiempo de codificación. Sin embargo, esto no quita la necesidad de revisar el código generado y adaptarlo a las necesidades concretas de la aplicación.

El desarrollo de esta práctica, junto con las dos anteriores, nos ha permitido explorar diversos mecanismos de comunicación, cada uno con sus particularidades. Esto nos ha permitido afianzar las bases en la programación de aplicaciones distribuídas, dándonos las herramientas suficientes para afrontar el desarrollo de la práctica final.