Sistemas Distribuidos

Grado de Ingeniería en Informática
Curso 2023-2024

Ejercicio Evaluable 2

Integrantes del Grupo:
María Isabel Hernández Barrio (100472315)
Blanca Ruiz González(100472361)

El objetivo de este ejercicio es describir el diseño y la implementación de un servicio de elementos clave-valor1-valor2, como en el ejercicio 1 pero ahora utilizando sockets TCP. Este servicio consta de un servidor concurrente que gestiona las operaciones sobre las tuplas clave-valor almacenadas, y un conjunto de funciones en el lado del cliente que permiten interactuar con dicho servidor.

El sistema se compone de las siguientes partes:

- Servidor (servidor.c):

Servidor TCP concurrente que gestiona operaciones sobre elementos clave-valor almacenados en archivos. El servidor crea un socket, lo vincula a una dirección y puerto específicos, y luego escucha conexiones entrantes dentro de un bucle. Cuando se establece una conexión con un cliente, se crea un hilo para manejar la petición del cliente, permitiendo así la concurrencia en la gestión de múltiples clientes simultáneamente. Las operaciones admitidas incluyen vaciar el directorio de almacenamiento, guardar datos en archivos, leer datos de archivos, modificar valores existentes, eliminar archivos y verificar la existencia de archivos. Cada operación es manejada por una función específica que interactúa con los archivos en el directorio "tuplas". Una vez completada la operación, se envía una respuesta al cliente a través del socket. El servidor continúa escuchando nuevas conexiones indefinidamente mientras los hilos atienden las peticiones.

- Biblioteca del Cliente (claves.c):

El archivo "claves.c" implementa las funciones definidas en el archivo de encabezado "claves.h", las cuales interactúan con el servidor mediante la función ‘mandar_peticion()’. Esta función establece una conexión TCP/IP con el servidor utilizando los parámetros de dirección IP y puerto obtenidos de las variables de entorno "IP_TUPLAS" y "PORT_TUPLAS". Luego, envía una solicitud al servidor con la operación especificada (‘op’). Finalmente, se recibe la respuesta del servidor y se cierra la conexión. Una cosa que hemos cambiado respecto a la anterior entrega es enviar el operador como un carácter en vez de como un entero, para enviar menos bytes a través del socket y optimizar la conexión.

- Biblioteca Dinámica (libclaves.so):

Es una biblioteca dinámica generada a partir de claves.c que contiene todas las funciones necesarias para interactuar con el servidor. Los clientes enlazan con esta biblioteca para utilizar los servicios que ofrece el servidor.

- Cliente (cliente.c):

Realiza una serie de pruebas para verificar el correcto funcionamiento de las funciones definidas en el archivo "claves.c", las cuales interactúan con el servidor. Comenzamos iniciando el servicio mediante la función ‘init()’ y vamos implementando el resto de las funciones para realizar un programa de prueba. Se realizan también pruebas adicionales para verificar el comportamiento esperado en situaciones específicas, como intentar borrar una clave que no existe, verificar si se eliminan correctamente todos los archivos, y probar la restricción de rango para el parámetro ‘N_value2’.

- Mensaje.h:

Define dos estructuras de datos: ‘struct peticion’ y ‘struct respuesta’.

‘struct peticion’ consta de:

- op: Un array de caracteres de tamaño 1 que representa el tipo de operación a realizar
- clave: Un entero que representa la clave del fichero.
- value1: Un array de caracteres de tamaño MAXSIZECHAR (256) que almacena un valor asociado a la petición.
- N_value2: Un entero que indica la cantidad de elementos en el arreglo V_value2.
- V_value2: Un array de dobles de tamaño MAXSIZEVECTOR (32) que almacena valores asociados a la petición.

‘struct respuesta’ consta de:

- value1: Un array de caracteres de tamaño MAXSIZECHAR (256) que almacena un valor asociado a la respuesta.
- N_value2: Un entero que indica la cantidad de elementos en el arreglo V_value2.
- V_value2: Un array de dobles de tamaño MAXSIZEVECTOR (32) que almacena valores asociados a la respuesta.
- todo_ok: Un entero que se devuelve para confirmar si la operación ha sido realizada con éxito (0) o si ha habido algún problema (-1); o si el archivo con determinada clave existe (1) o no (0), en el caso de la función exist.

Uso y Plan de pruebas:

Para poner el sistema en marcha, hay que hacer un make, ya que se hace mediante un MakeFile, como en la anterior entrega. El servidor tiene el siguiente uso: `servidor <PUERTO>.` ; y el cliente no toma ningún parámetro, pero se le debe especificar mediante variables de entorno la dirección IP y el puerto desde el que debe conectarse al servidor, el mismo que hemos especificado para inicializar el servidor, en este caso usamos el 8083 y el localhost como dirección.

El cliente imprime el plan de pruebas, hemos utilizado el mismo que en la entrega anterior. Primero, ejecutamos el servidor con `./servidor 8083`

Al ejecutar el cliente (`env IP_TUPLAS=localhost PORT_TUPLAS=8083 ./cliente`) obtenemos un plan de pruebas exitoso:

```
Init OK.
Set_value OK.
Valor1 debería ser Hola: Hola
N_valor2 debería ser 3: 3
V_valor2 debería ser 1.0, 2.0, 3.0
  V_valor2[0]: 1.000000
  V_valor2[1]: 2.000000
  V_valor2[2]: 3.000000
Get_value OK.
Valor1 debería ser Adios: Adios
N_valor2 debería ser 3: 3
V_valor2 debería ser 4.0, 5.0, 6.0
  V_valor2[0]: 4.000000
  V_valor2[1]: 5.000000
  V_valor2[2]: 6.000000
Resultado de Exist debería ser 1: 1
Modify Value y Exist OK.
Resultado de Exist debería ser 0: 0
Delete Key OK.
Error en el delete_key 2 ( OK. ERROR ESPERADO)
Set_value OK.
Resultado de Exist debería ser 0: 0
Init 2 TEST OK.
Numero de N_value2 no puede ser < 1: Success
Error en el set_value ( OK. ERROR ESPERADO)
Fin de los tests, todo OK
```

Hemos dejado los prints del servidor que indican su estado: esperando a nuevas peticiones y avisando cada vez que recibe una conexión; por tanto, el servidor se verá así:

Sin embargo, estos prints podrían omitirse.