Universidad Carlos III de Madrid

Universidad Carlos III

Sistemas Distribuidos

Curso 2024-25

Sistemas Distribuidos

Grado Ing Informática

Fecha: **2/3/2025** - ENTREGA: **I**

GRUPO: **80**

Alumnos:

Rodrigo García Salado 100478705

Iván Flores López 100495990

INDICE

Diseño realizado

En un primer lugar el servidor creará un socket que inicializará con el bind, asignando gracias a esta función su dirección y el puerto. El siguiente paso será iniciar un proceso de escucha por parte del servidor con el listen(), esperando a la llegada de un mensaje de un cliente, para seguidamente llamar a la función accept() para aceptar la próxima solicitud de connect() de un cliente.

El cliente, por su parte, creará un socket y llamará a la función connect() que pedirá una solicitud de conexión al servidor y que paralelamente debe estar en estado de accept(). A partir de éste punto se inicia una conexión entre el cliente y el servidor los cuales se comunicarán libremente hasta que ambos llamen a la función de close(), que terminará con ésta conexión.

Diseño entre el proxy y el servidor

El protocolo de aplicación entre el proxy y el servidor se basa en la comunicación binaria secuencial, donde cada mensaje comienza con un campo op que indica la operación solicitada. Dependiendo de la operación, se envían otros campos como key, value1, N_value2, V_value2, y value3.x/y. La respuesta del servidor incluye un status y los mismos campos en el orden correspondiente. La comunicación es síncrona, con el proxy esperando la respuesta del servidor después de enviar cada mensaje. Los datos se transmiten de manera directa utilizando las funciones write() y read(), sin encabezados adicionales ni delimitadores.

Estructura del sistema

Nuestro sistema está dividido en 4 archivos.c que contienen los datos de la práctica y el código para que la misma funcione. Por motivos de la propia práctica se han tenido que crear 2 archivos ejecutables, uno de ellos (servidorEJ) hará la función que tendría cualquier servidor, recibiendo solicitudes y enviando los resultados procesados por sus respectivas funciones, éste ejecutable será el resultado de compilar “servidor-sock.c” y “claves.c”.

Por otro lado tenemos a nuestro ejecutable llamado “clienteEJ” en el cual se situará nuestro menú y que hará llamadas al servidor pidiéndole ejecutar una función u otra según su solicitud. Este ejecutable surge de compilar “proxy-sock.c” y nuestro código cliente situado dentro de la carpeta “cliente” llamada “app-cliente.c”.

Para compilar el programa y generar los archivos ejecutables simplemente habrá que poner el comando “make” en la consola situándose en la raíz del proyecto. Para lanzar el servidor, habrá que poner el comando “./servidorEJ <PuertodeEscucha>”. Para ejecutar el cliente se deberá primero declarar las variables de entorno IP_TUPLAS y PORT_TUPLAS con el comando “export IP_TUPLAS=127.0.0.1” ya que es la ip del servidor y “export PORT_TUPLAS=<puerto que se ponga en el puerto del servidor>”. Una vez hecho esto sólo habrá que lanzar en la consola el comando “./clienteEJ”.

Plan de pruebas:

Nuestras pruebas están situadas dentro del código principal, como una de las opciones del menú de la aplicación, no son un ejecutable aparte.
Primera prueba: Prueba de set value con key inexistente.

-Se inserta en la key 999 (vacia) con set_value una tupla y se comprueba su existencia con get_value. Se envia el mensaje correctamente y devuelve el servidor un status = 0 mostrandose en la cola del cliente que la tupla ha sido insertada correctamente

-Se recoge la tupla con key 999, el servidor la encuentra y devuelve los valores insertados dentro de la tupla, mostrándose en la cola del cliente

- Prueba modify_value con la tupla 999 cambiando sus valores a unos y comprobandolo con un get_value. El servidor responde con status = 0 y la consola del cliente muestra que la tupla se ha modificado correctamente. Finalmente se comprueba que los datos que han guardado correctamente con un get y se confirma el cambio de datos del modify_value.

- Prueba Exist_key Se busca la existencia de la key 999 la cual existe y el servidor devuelve un status = 0 y el cliente muestra por su consola que la tupla existe.

- Prueba de set_value a key 999 (ya existente), la función devuelve -1 (error) ya que ya existe un valor para esa key. En la consola del cliente se muestra “Error al insertar, la tupla ya existe”

- Prueba delete key a 999 la función devuelve 0 y la key 999 se borra, mostrándose en la terminal del cliente que la tupla se ha borrado correctamente.

- Prueba modify value a key 999 (ahora vacía), la función devuelve -1 (error) ya que no existe un valor con esa key. En la terminal del cliente se muestra el mensaje “Error al modificar la tupla, la tupla no existe”

- Prueba Destroy, al comienzo insertamos la tupla 1000 con valores por defecto, y se verifica la existencia de las tuplas 999 y 1000 comprobando que existen. Eliminamos toda la lista enlazada llamando al destroy, el servidor devuelve status = 0 y en la terminal del cliente se muestra “Tuplas eliminadas correctamente”. Además en ésta prueba se intenta hacer un get value de las keys 999 y 1000 para demostrar que efectivamente se han borrado todas correctamente.

-Se han hecho pruebas de inserción borrado y consulta al mismo tiempo con diferentes clientes al mismo tiempo confirmando el funcionamiento de la concurrencia.