Sistemas Distribuidos.

Ejercicio Evaluable 2: Sockets

Servidor.c

El archivo servidor.c contiene la estructura de datos que se utiliza para el array distribuido. Debido a problemas con la entrega anterior, la estructura de datos se cambió a un simple array dinámico que contiene structs de la tupla clave-valor descrita en el enunciado. La struct se llama KeyValue y contiene la clave, valor1, N y valor2. Luego definimos las funciones individuales que se solicitan para añadir elementos al array, eliminar del array, modificar elementos existentes, verificar si un elemento existe y obtener la tupla KeyValue de un elemento en el array. Se incluyó la comprobación de errores adecuada para asegurar que permanecemos dentro de los límites, que los elementos existen y que el array se asigna, desasigna y redimensiona correctamente al insertar elementos y al eliminarlos. En este proceso, adaptamos la comunicación para no enviar directamente las structs a través de la red. En lugar de ello, los datos se serializan en un formato que puede ser enviado y recibido, y luego se deserializan de nuevo en la estructura original en el otro extremo. Esta serialización y deserialización se maneja mediante funciones específicas que preparan los datos para la transmisión y los reconstruyen después. También inicializamos un mutex aquí. En todos los puntos durante la creación del socket (crear, vincular, escuchar, aceptar, ...), nos aseguramos de que cualquier error cerrará cualquier descriptor abierto que podamos tener, así como liberar el array y destruir el mutex. Tras crear con éxito el socket, aceptamos conexiones entrantes y asignamos un socket a cada una, pasando el socket de la conexión entrante como argumento a la función del hilo. La función del hilo procesa entonces los datos serializados, realizando la operación solicitada y devolviendo el resultado de manera

serializada. Se realiza la operación y el resultado, ya sea ERROR (-1) o ÉXITO (0), se serializa y se envía de vuelta al cliente que solicitó la operación. Luego cerramos el socket del cliente. Eso explica todo lo que hay en servidor.c excepto la concurrencia. Desafortunadamente, debido a la naturaleza del problema y quizás al proceso de diseño por el que opté, la concurrencia fue muy limitada. Originalmente, mi plan era permitir operaciones concurrentes pero después de terminar todo el diseño, me di cuenta de que eso podría no ser posible quizás debido a la estructura de datos utilizada. Por ejemplo, si un cliente inserta en el array y otro cliente elimina algo del array y un tercer cliente verifica si un elemento existe, podría suceder lo siguiente: el Cliente A inserta en el array, lo que significa que redimensiona el array e incrementa el tamaño, mientras que el Cliente B desplaza el array hacia la izquierda debido a la eliminación de un elemento y el Cliente C verifica si un elemento que puede haber sido eliminado existe dependiendo de cuál Cliente ejecuta primero. Esto podría causar un comportamiento indefinido y hay muchos casos de este tipo que podrían surgir quizás debido a la alta cohesión. Por esa razón, simplemente decidí permitir que un usuario realice una operación a la vez, lo que significa que la ejecución de solicitudes se volverá serial. Aunque esto es contraintuitivo a la idea de concurrencia y múltiples clientes ejecutando operaciones al mismo tiempo, fue todo lo que se me ocurrió y la opción más segura para evitar carreras. Desafortunadamente, no entiendo completamente el uso de variables condicionales, quizás hubieran sido útiles aquí.

Cliente.c & Clientes.c

Cliente.c y Clientes.c son dos archivos de prueba diferentes.

Cliente.c establece un valor, obtiene el valor, lo modifica, verifica si la clave existe, la elimina, verifica la existencia de la clave nuevamente y luego llama a init. Este archivo de prueba simplemente sirve para comprobar si la función se comporta correctamente y devuelve los valores correctos, modifica el array como se solicita y cómo funcionan en casos de errores u operaciones inválidas, como verificar una clave eliminada. Cuando se ejecuta la función, está claro que funcionan como se espera.

Clientes.c crea 5 clientes para probar la concurrencia del servidor. Aunque anteriormente discutimos las limitaciones de concurrencia y cómo estos clientes ejecutarán su operación de manera serial, sigue siendo importante verlo suceder. Cada cliente tiene su turno para ejecutar sus operaciones. Los clientes se crean utilizando hilos.

Claves.c

Claves.c implementa las funciones definidas en claves.h, ajustándose a un nuevo enfoque de comunicación que evita enviar estructuras directamente a través de la red. Cada función prepara un mensaje, lo serializa en un flujo de bytes, lo envía al servidor, espera una respuesta, la deserializa y procesa el resultado. Este cambio responde a la necesidad de cumplir con restricciones específicas del proyecto, adaptándose para garantizar la compatibilidad y eficiencia en la transmisión de datos entre cliente y servidor. Las operaciones cubiertas incluyen inicialización, establecimiento, obtención, modificación y eliminación de valores, así como la verificación de la existencia de claves, utilizando el protocolo de comunicación serializado.

Message.c

Message.c surge como una pieza central en este esquema modificado, alojando las funciones de serialización y deserialización que transforman la estructura de mensajes entre su representación en memoria y una forma apta para la transmisión de red. Estas operaciones permiten manejar los datos estructurados de manera eficaz al enviar y recibir a través de la red, facilitando el intercambio de información compleja entre el cliente y el servidor sin comprometer la integridad de los datos. Este archivo encapsula la lógica necesaria para interpretar y reconstruir los mensajes, asegurando que ambos extremos de la comunicación interpreten los datos de manera coherente.

Compilation

Para compilar, simplemente ejecute 'make'. Se generarán dos ejecutables, uno para el cliente y otro para el servidor, y también uno para la prueba concurrente, clientes. Para ejecutar, realice los siguientes pasos:

- ./servidor <PORT>

Para cliente.c

- env IP_TUPLAS=localhost PORT_TUPLAS=<PORT> ./cliente

Para clientes.c

- env IP_TUPLAS=localhost PORT_TUPLAS=<PORT> ./clientes

Reemplace <PORT> con el puerto de su elección, por ejemplo, 2000.