Sistemas Distribuidos

Implementación de RPC

Nathalia Moniz Cordova 100471979

Alberto García de la Torre Fernández del Moral 100472039

21 de abril del 2024

Diseño del programa

En este tercer ejercicio evaluable se hará uso de la implementación RPC (Remote Procedure Call). En este caso, parte de los archivos que hay en el trabajo han sido generados al ejecutar el archivo claves_inter.x

claves_inter.x

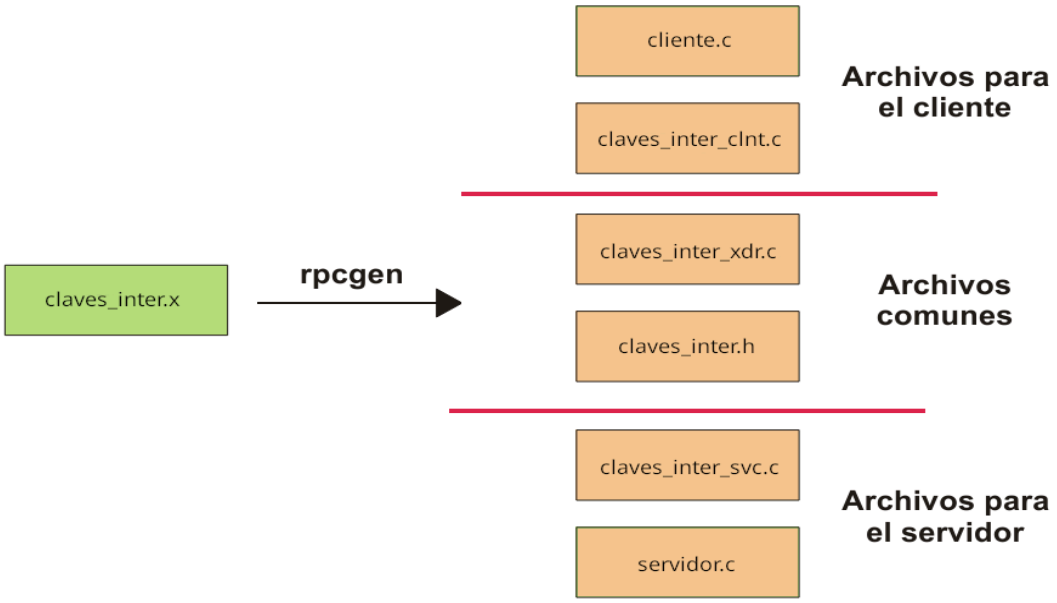
Es la interfaz encargada de generar la comunicación rpc entre cliente y servidor. El archivo es un ejecutable que está escrito en el lenguaje de especificación de procedimientos remotos (XDR). Al ejecutarlo mediante el comando rpcgen, genera cuatro archivos: **claves_inter_client.c**, **claves_inter_clnt.c**, **claves_inter_server.c**, **claves_inter_svc.c**, **claves_inter_xdr.c**, **claves_inter.h**.

Los programas generados por **claves_inter.x** se encargarán de conectar el lado del cliente con el del servidor.

A la hora de mandar los valores a través de las funciones hemos decidido hacerlo sin usar estructuras, debido a que no todas las funciones hacen uso de todos los argumentos y pasándole a cada una solo los argumentos que necesita hacemos un mejor uso del espacio de memoria.

Archivos generados por claves_inter.x

Consisten en seis archivos que se generan al ejecutar el comando rpcgen. Estos seis archivos se dividen en tres grupos: archivos para el cliente, archivos comunes, y archivos para el servidor.



claves.c

Al igual que en el archivo **servidor.c**, el archivo **claves.c** también contiene una estructura que ha sido generada por el comando rpcgen. En este caso, se han conservado también los nombres de las variables utilizadas, generando variables distintas para cada operación.

servidor.c

Esta función contiene una estructura de código que ha sido generada por rpcgen. Tiene una función por cada petición posible que puede realizar el cliente.

El parámetro devuelto por las funciones viene dado por la variable retval, la cual se encontraba ya dentro de la estructura generada por rpcgen.

El cliente recibe el resultado de la operación mediante el puntero result, que guarda el resultado de la operación realizada en la lista.

list.c

Las funciones contenidas en este archivo son iguales a excepción de la función **get()**. Para esta práctica, hemos visto conveniente devolver una estructura de respuesta con esta función. Dentro de la estructura se encuentran los valores encontrados con la llave seleccionada.

También se han conservado los mutex realizados en cada una de las operaciones donde se manipula algún valor de la lista.

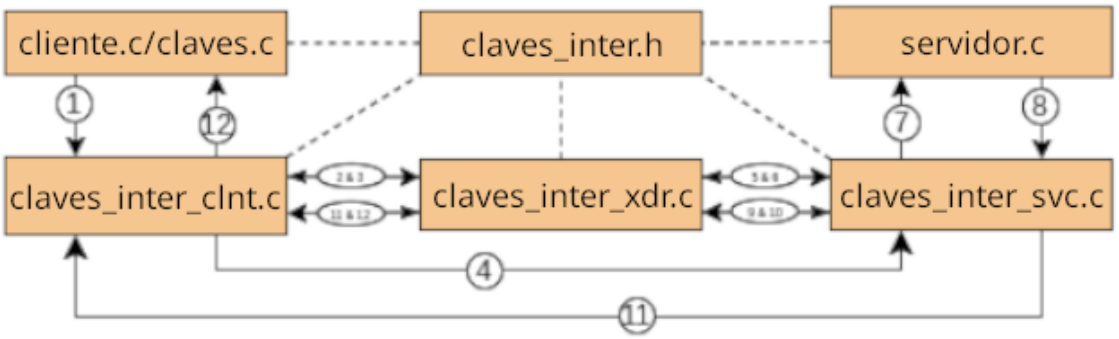
cliente.c

Por último, contamos con la parte del cliente que simplemente se encargará de llamar a las funciones, ya que todo el manejo de sockets se hace en **claves.c** y en **servidor.c**. El código del cliente es prácticamente idéntico al que había en los anteriores ejercicios evaluables, ya que no ha requerido de cambios adicionales.

Comunicación y protocolo

El protocolo utilizado es TCP. Al tratarse de peticiones con una cantidad estricta de información, se requiere el uso de un protocolo que asegure la fiabilidad de la información transmitida.

Como ya se ha mencionado en la explicación de **claves_inter.x**, el ejecutable genera seis nuevos archivos. Estos archivos permiten la comunicación entre el cliente y el servidor, ya que comparten un mismo header que contiene las funciones necesarias para realizar la comunicación. Cuando se usa una de estas funciones desde el cliente o el servidor, la información es transmitida a través de los archivos claves_inter hacia el otro extremo.



Compilación y ejecución del programa

Estando en la carpeta **Ejercicio-evaluable-3-SSDD** hay que introducir en un terminal el siguiente comando:

```
Unset
make
```

Cuya salida debe ser similar a la siguiente. Hay que tener en cuenta que hay un par de warnings de los archivos generados por rpcgen, pero estos no interrumpen el correcto funcionamiento del proyecto. Hemos decidido no borrar ni modificar estos archivos:

```
Unset
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o servidor.o
servidor.c
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o claves.o claves.c
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o claves_inter_xdr.o
claves_inter_xdr.c
claves_inter_xdr.c: In function ‘xdr_double_array’:
claves_inter_xdr.c:11:27: warning: unused variable ‘buf’ [-Wunused-variable]
   11 |      register int32_t *buf;
      |                           ^~~
claves_inter_xdr.c: In function ‘xdr_KeyValue’:
claves_inter_xdr.c:24:13: warning: unused variable ‘i’ [-Wunused-variable]
   24 |      int i;
      |            ^
claves_inter_xdr.c:22:27: warning: unused variable ‘buf’ [-Wunused-variable]
   22 |      register int32_t *buf;
      |                           ^~~
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o
claves_inter_clnt.o claves_inter_clnt.c
gcc -shared -o libclaves.so claves.o claves_inter_xdr.o claves_inter_clnt.o
-Wall -fPIC -g -I/usr/include/tirpc
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o claves_inter_svc.o
claves_inter_svc.c
gcc -o servidor servidor.o libclaves.so claves_inter_svc.o
claves_inter_xdr.o -Wall -fPIC -g -I/usr/include/tirpc -lnsl -lpthread -ldl
-ltirpc -lm
gcc -Wall -fPIC -g -I/usr/include/tirpc -I/include  -c -o cliente.o
cliente.c
gcc -o cliente cliente.o libclaves.so -Wall -fPIC -g -I/usr/include/tirpc
-lnsl -lpthread -ldl -ltirpc -lm
```

Para la ejecución del programa se tiene que iniciar en primer lugar el servidor:

```
Unset
./servidor
```

Y para el cliente primero debemos decirle al sistema operativo dónde buscar las bibliotecas compartidas (archivos de bibliotecas dinámicas) necesarias para ejecutar el programa con el siguiente comando:

```
Unset
export
LD_LIBRARY_PATH=/home/username/Ejercicio-evaluable-3-SSDD:$LD_LIBRARY_PATH
```

Y finalmente ejecutar el cliente indicando la IP y el puerto y los argumentos que se desee:

```
Unset
env IP_TUPLAS=localhost ./cliente <op> <key> <value1> <N_value2>
```

Los valores del vector no se introducen por terminal, sino que son generados aleatoriamente por el archivo **clientes.c**. El número de valores generados viene dado por <N_value2>.

Pruebas

Las pruebas realizadas consisten en 13 tests que se encuentran dentro de la carpeta “tests”. Cada una de las pruebas comprueba una especificación distinta del enunciado. Hay tres tipos de test: correctos (test 1-5), errores (test 6-11), y de concurrencia (test 12 y 13). A continuación describiré el objetivo de cada test:

- **test1:** comprueba el funcionamiento de la función set_value.
- **test2:** comprueba el funcionamiento de la función get_value.
- **test3:** comprueba el funcionamiento de la función modify_value.
- **test4:** comprueba el funcionamiento de la función delete_key.
- **test5:** comprueba el funcionamiento de la función exist.
- **test6:** muestra un error si no se realiza init al inicio.
- **test7:** muestra un error si se intenta introducir un valor con una llave idéntica a otro existente, o si el tamaño del vector no es válido.
- **test8:** muestra un error si el valor de la llave buscada no se corresponde a ninguno de la lista.
- **test9:** muestra un error si el valor de la llave a modificar no se corresponde a ninguno de la lista.
- **test10:** muestra un error si el valor de la llave a eliminar no se corresponde a ninguno de la lista.
- **test11:** la función exist devuelve el valor 0 en caso de no encontrar un valor en la lista que se corresponde con la llave introducida.
- **test12:** comprueba que es posible ejecutar dos procesos de cliente simultáneamente.
- **test13:** simula el comportamiento de 4 clientes que realizan una gran cantidad de cambios simultáneos en la lista.
- **aux:** inserta 100 elementos en la lista con valores de llave del 0 al 100.
- **aux2:** inserta 100 elementos en la lista con valores de llave del 100 al 200.
- **aux3:** modifica 100 elementos pares de la lista con valores de llave del 0 al 200.
- **aux4:** elimina 200 elementos en la lista con valores de llave del 200 al 0. Empieza la eliminación de los elementos por el valor número 200.

Ejecución de las pruebas

Al ejecutar las pruebas, se debe introducir el nombre del test que deseas realizar. A continuación hay un ejemplo del comando para ejecutar el test1.sh:

```
Unset
./test1.sh
```