Sistemas distribuidos G81: Tercera Práctica

József Iván Gafo - 100456709@alumnos.uc3m.es
Pablo Moreno González - 100451061@alumnos.uc3m.es

1. Introducción

El objetivo de la tercera parte de esta práctica, es de reutilizar los clientes de la parte 1 e implementar un nuevo proxy y servidor usando los rpc.

Para esto, el sistema se va a componer de tres elementos fundamentales.

Cliente: va a ser el que envía las peticiones al servidor mediante una API (app-cliente).

Proxy: va a actuar como intermediario entre el cliente y el servidor.

Servidor: va a ser el encargado de procesar las peticiones de los clientes y de almacenar las duplas en una lista enlazada que previamente hemos implementado. También, el servidor será multihilo permitiendo que varios clientes sean atendido en una misma instancia de tiempo.

2. Diseño

2.1 Como se ha generado el código

Primero hemos creado un archivo .x llamado claves_rpc y le hemos definido qué funciones tiene el servidor que dará servicio asignando un número de operación empezando por el 1. A continuación hemos modificado el input y output de las funciones como nos lo indica el manual de rpc. También hemos puesto el número de versión de nuestro programa y hemos puesto nuestro nia en este caso “1003456709” para el identificador del rpc y evitar conflictos con otros programas.

Después de haber terminado claves_rpc.x, usamos el comando “rpcgen -M -N -a claves_rpc.x” para generar los archivos cliente y servidor. Siendo -M multihilos , -N desactiva antiguas versiones de rpc y por último el -a para incluir archivos de ejemplo el servidor y cliente, en el cual cogeremos la estructura del servidor de ejemplo y adaptarnos al problema y el cliente se ha hecho desde cero, pero el ejemplo nos sirvió para entender cómo funciona el rpc.
Un problema que tuvimos fue que nos decía que había conflictos de versiones al ejecutarse el servidor y cliente, pero se soluciono al cambiar el nombre del programa y la versión.

2.1 Arquitectura del Sistema

1. Se arranca el servidor (e.g./servidor-rpc) y este se registrará en el sistema rpc (portmapper) para que los clientes lo puedan encontrar.
2. El cliente debe primero establecer las variables del entorno en el cual define la IP (e.g env IP TUPLA=localhost)
3. El flujo del sistema va a ser que primero el cliente realiza una petición mediante la Api (*claves.h)*.
4. El proxy-rpc va a convertir la petición de tal manera que el servidor lo pueda recibir correctamente sin errores.
5. El servidor recibe la petición, le asigna un hilo y los servicios generados por el rpc se encargará de gestionar la petición del cliente.
6. Una vez que el servidor haya gestionado el cliente, le envía la respuesta.
7. El proxy recibe la respuesta y se lo envía al cliente.

2.2 Estructura de los Archivos

```
P3_sistemas_distribuidos
├── app-cliente/
│   ├── app-cliente-1.c #Pruebas (insertar, modificar, eliminar, existir, destroy)
│   ├── app-cliente-2.c #Pruebas de errores, límites y validaciones
│   └── app-cliente-3.c #Pruebas de concurrencia en múltiples procesos
├── claves/
│   ├── claves.c #Implementación del almacenamiento en memoria (listas enlazadas)
│   └── claves.h #API del sistema clave-valor
├── rpc/
│   ├── proxy/
│   │   └── proxy-rpc.c #Implementación del proxy rpc
│   ├── servidor/
│   │   └── servidor-rpc.c #Implementación del servidor rpc
│   ├── claves_rpc.h #Encabezado generado por rpcgen
│   ├── claves_rpc.x #Archivo para generar el rpc
│   ├── claves_rpc_clnt.c #código generado para el cliente, generado por rpcgen
│   ├── claves_rpc_svc.c #código generado para el servidor, generado por rpcgen
│   ├── claves_rpc_xdr.c #código para la (des)serialización de datos,generado por rpcgen
│   └── Makefile.claves_rpc #makefile generado por rpcgen, (no se usa)
├── Makefile #Makefile principal de nuestro proyecto
└── README.md #Documentación básica del proyecto
```

2.3 Sección de Pruebas

Para verificar el correcto funcionamiento de nuestro nuevo proxy y nuevo servidor usando programación de sockets, hemos reutilizado los mismos app clientes de la práctica 1. En el cual el primer app cliente está encargado de verificar el correcto funcionamiento de los distintos componentes, el 2 encargado de mirar los límites y el último encargado de verificar que nuestro sistema puede funcionar correctamente de manera distribuida.

3. Implementación

3.1 Implementación del servidor

Hemos cogido la template que nos ha generado rpc_gen y verificamos primero que el input este correcto, luego si se trata de un vector lo guardamos en una variable local, y por último si había que poner las coordenadas creamos una estructura Coord y guardabamos su valor equivalente de la estructura rpc_Coord y por ultimo ejecutamos la función y el resultado se guardaba en * resultado.
Y para terminar la función, retornamos verdadero para decir que se ha ejecutado correctamente y falso que ocurrió un error grave al procesar la solicitud, en nuestro caso este último no lo hemos retornado, ya que debemos indicar el resultado de la operación -1 si ha fallado.

3.1 Implementación del proxy

Para el proxy hemos implementado una función init_rpc_client que está encargada de recoger la ip declarada en la variable de entorno “IP_TUPLAS” y crear una conexión tcp con el servidor. A continuación implementamos todas las funciones de claves.h, y lo primero es validar la entrada, luego hacer el init_rpc_client, a continuación copiamos los valores de entrada en la estructura adecuada definida en “claves_rpc.x” (si hace falta), para enviarlo al servicio rpc. A continuación ejecutamos la función definida por claves_rpc.h que ya estará encargado de hacer la comunicación con el servidor. Por último retornamos el resultado obtenido por el rpc a excepción de get_value_rpc, en el cual devuelve una estructura y en caso exitoso debemos traducir la estructura y guardarlo en los parámetros dados al proxy para dárselo al app-cliente.

4. Compilación y Ejecución

Usamos un Makefile para compilar y ejecutar todo el sistema:

```
# Limpiar archivos antiguos
make clean

# Compilar todo
make


# Ejecutar el servidor
./servidor-rpc
```

```
# Variables de entorno para proxy
export IP_TUPLAS=localhost


# En otra terminal, ejecutar el cliente
./app-cliente-1
./app-cliente-2
./app-cliente-3
```

Primero, compilamos todo el proyecto con el Make, esto compilará todos los archivos que necesita el servidor, el próximo, las librerías comunes y los tres clientes de prueba que hemos implementado.

Segundo, una vez compilado todo el proyecto pasamos a la ejecución del sistema donde iniciaremos el servidor. (si activamos el DEBUG, eso nos mostrará mensajes informativos sobre las conexiones y las operaciones).

Tercero, pasamos a ejecutar los clientes. Antes de ejecutar cada cliente, es necesario configurar las variables de entorno para que nuestro PROXY sepa la dirección IP al que ha de conectarse. Antes de ejecutar cada cliente es necesario configurar las variables de entorno para que nuestro próximo sepa la IP al que ha de conectarse. (el cliente 1 va a ser para funcionalidades básicas, el cliente 2 para los errores y límites, y el cliente 3 para la concurrencia.)