Práctica 2 - Llamada remota a procedimiento (RPC)

Ejercicio 2: Calculadora en Apache Thrift

Jose Luis Gallego Peña - DSD2

1. Introducción

El ejercicio consiste en un desarrollar programa distribuido, una calculadora que realice varias operaciones, al igual que en el ejercicio 1, sin embargo se usará una tecnología distinta llamada Apache Thrift, que está también basada en IDL para generar clientes y servidores, pero con buen soporte multilenguaje. Es por esto que el objetivo de esta aplicación es, además, tener cliente y servidor en lenguajes de programación distintos.

Se ha decidido que:

- El cliente se haga en el lenguaje **C++**, puesto que permite fácilmente controlar parámetros por terminal y es el lenguaje que más domino, además de que estaba ya definido en C en el anterior ejercicio.
- El servidor se haga en el lenguaje **Python**, puesto que es un lenguaje muy simple, es el que tiene el código de servidor más corto y además tiene bibliotecas para gestionar las operaciones matemáticas de la calculadora más fácilmente. En este caso, se usa la biblioteca *numpy* para ello.

Las operaciones que realiza la calculadora son sumas, restas, multiplicaciones y divisiones de números reales, polinomios y vectores. En el caso de los vectores, se pueden operar entre sí dos vectores (realizando la operación que corresponda posición a posición) o un vector multiplicado por un escalar (el valor escalar se multiplica por todos los valores del vector, sólo está disponible la multiplicación en este caso).

He utilizado el SO Arch Linux x86_64, con kernel Linux 5.5.9-arch1-2. y con Python 3.8.2.

2. Compilación y ejecución

Primero he generado el IDL, el archivo llamado **calculadora.thrift**, con las definiciones comunes al cliente y al servidor. Todo ello en el propio lenguaje de Apache Thrift, que es independiente de los lenguajes de programación para facilitar la comunicación entre ellos y la traducción posterior. El contenido de este archivo se explicará más adelante.

Una vez generado el fichero IDL hay que compilarlo, para esto hay que usar el compilador de thrift. Como no lo tenía, he tenido que instalar lo siguiente usando yay, el gestor de paquetes de mi distribución:

```
yay -S thrift
```

Una vez instalado, ya sí se puede usar el compilador. Puesto que tenemos un cliente y un servidor en distintos lenguajes, los archivos que generará el compilador de thrift serán diferentes según el lenguaje, por lo que hay que ejecutarlo dos veces, uno con la opción para python y otro con la opción para c++:

```
thrift -gen py calculadora.thrift
thrift -gen cpp calculadora.thrift
```

Esto nos creará dos carpetas, *gen-py* y *gen-cpp* en las que incluiremos los archivos *servidor.py* y *cliente.cpp* respectivamente, es decir, cada una en la carpeta que corresponda con su lenguaje.

Resumidamente, tenemos la siguiente estructura de carpetas y ficheros:

- **calculadora.thrift**, código en el lenguaje de Apache Thrift y parte más importante de todo el sistema. Define las estructuras, funciones y datos accesibles desde el cliente y servidor, y desde este archivo se genera todo lo demás.
- **gen-cpp**, carpeta con todos los ficheros generados por el compilador de thrift que usará el cliente.
  - **cliente.cpp**, el programa que ejecutará el cliente y cuyo contenido se explicará en el próximo apartado. Se ejecuta en un terminal distinto del terminal del servidor.
- **gen-py**, carpeta con todos los ficheros generados por el compilador de thrift que usará el servidor.

- **servidor.py**, el programa que ejecutará el servidor y cuyo contenido se explicará en el próximo apartado. Se ejecuta en un terminal distinto del terminal del cliente.

Sin embargo, con Apache Thrift esto no basta. Este servicio necesita, además del compilador, bibliotecas específicas para poder ejecutar el código de cada lenguaje. Es decir, sólo podríamos programar servidores y clientes de Apache Thrift en lenguajes que soporte, por suerte tanto python como c++ los soporta.

Para añadir las bibliotecas de python simplemente tenemos que instalarlas con *pip*, el gestor de paquetes de python. Yo no lo tenía así que tuve que instalarlo antes con mi gestor de paquetes:

```
yay -S python-pip
pip install thrift
```

Con esto ya se podría poner en funcionamiento el servidor python, sin embargo en C++ es algo más complicado. Para instalar las bibliotecas de Thrift para C++ en el sistema Linux, seguimos las indicaciones que vemos en el github de Apache Thrift: github.com/apache/thrift, que son las siguientes:

```
git clone https://github.com/apache/thrift.git
cd thrift
sudo ./bootstrap.sh
sudo ./configure --with-boost=/usr/local
sudo make
sudo make install
```

Esto nos instala todas las bibliotecas de todos los lenguajes, ya que no hay una opción simple para instalar solo la de C++. Si no hubiesemos hecho nada de esto, al compilar nos daría errores y no podríamos generar el cliente.

El funcionamiento consiste en abrir dos terminales, una ejecutando el cliente y otra ejecutando el servidor. El servidor una vez se ejecuta, se mantiene activo indefinidamente, escuchando peticiones del cliente. El cliente por su parte, cuando hace su llamada al servidor y esta es completada, se cierra.

Para ejecutar el servidor no es necesario compilarlo, puesto que Python es un lenguaje interpretado. Simplemente hay que ejecutar lo siguiente:

```
python servidor.py
```

Pero el cliente es en C++, y este es un lenguaje compilado, por lo que hay que compilar primero el cliente contra todos los demas archivos y luego ejecutarlo, de la siguiente manera:

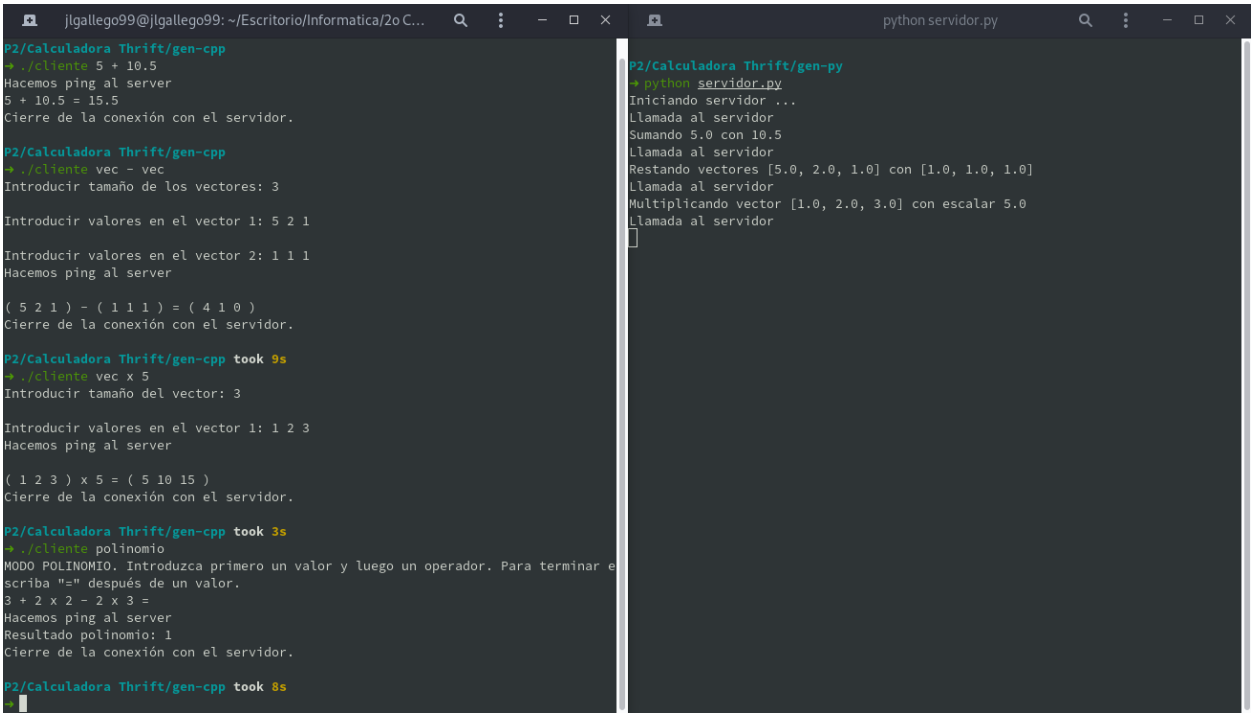
```
g++ cliente.cpp calculadora_constants.cpp Calculadora.cpp calculadora_types.cpp -I. -o cliente -lthrift
```

Si no se añadía la opción *-lthrift* el compilador no detectaba la ruta de la biblioteca de Thrift y no compilaba, por tanto es obligatorio añadirlo.

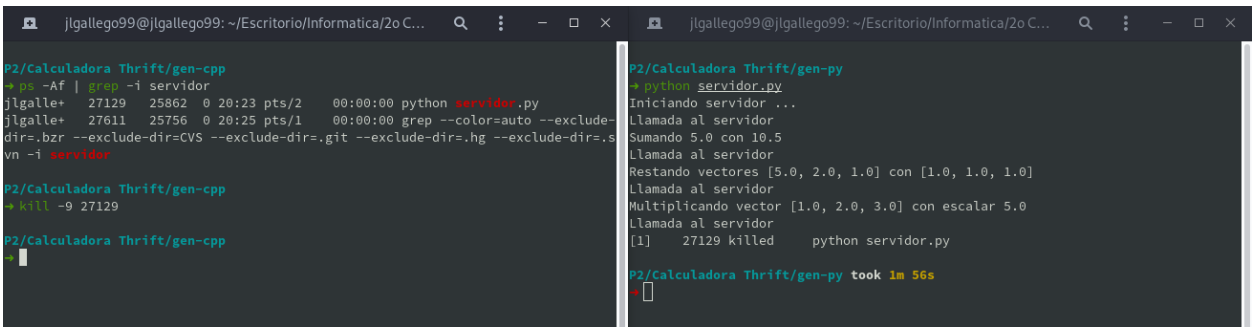
Un ejemplo de ejecución sería el siguiente, en el que se ven todas las distintas funcionalidades que tiene la calculadora:

- `./cliente localhost <valor1> <operador> <valor2>` Suma, resta, multiplica o divide dos números pasados por parámetros de terminal.
- `./cliente localhost <valor> x vec` o también `./cliente localhost vec x <valor>` Multiplica un vector por un valor pasado por terminal. Al poner vec por terminal se indica que se quiere introducir varios valores para formar un vector.
- `./cliente localhost vec <operador> vec` Suma, resta o multiplica un vector por otro. Al poner dos vec por terminal se indica que se quiere introducir varios valores para formar dos vectores.
- `./cliente localhost polinomio` Al pasar la opción polinomio por terminal, se indica que se quieren introducir varios números y operaciones para formar un polinomio, que se operará en el orden en el que se escribe (y teniendo en cuenta que las multiplicaciones y divisiones se operan antes que las sumas y las restas).

En todas ellas al final se da el resultado de la operación.



Para cerrar el servidor en este caso, a diferencia de con Sun RPC, sí se puede hacer con una interrupción de Ctrl+C. Pero también podemos buscar el proceso y matarlo, para ello se ejecuta `ps -Af | grep -i servidor`, que nos dará una lista de los procesos llamados "servidor". Buscamos el servidor y vemos que su PID es 32145, así que usamos `kill -9 27129` para matar al proceso y así cerrar el servidor de forma correcta.



3. Explicación de la solución

3.1 Definición (calculadora.thrift)

Define una interfaz, datos y rutinas a las que se puede acceder remotamente. Está escrito en el lenguaje de Apache Thrift.

Se ha definido una estructura para usar en el cliente y servidor:

```
struct operacion{
1: required double val;
2: required string op_izq;
3: required string op_dcha;
}
```

Define un tipo operación, para usarlo en un vector y así poer definir polinomios. Tiene los campos **val**, el propio número real de esa operación; **op_izq**, el operador que tiene a la izquierda ese valor (si es el primero

del polinomio es un + por defecto, que en el caso de ser val negativo no tendría efecto); y **op_dcha**, la operación que tiene a la derecha ese valor. Estas operaciones podrán luego ser: +, -, x, /. Todos los campos son *requiered* puesto que tienen que estar rellenos sí o sí, no pueden tener valor null.

Una vez definidos los tipos que se van a usar, en este caso *operacion*, se define el programa con sus funciones, llamado Calculadora y con las siguientes funciones:

1. ping(), una simple prueba para probar que el servidor está funcionando, simplemente muestra por pantalla un mensaje.
2. double suma(1: double num1, 2: double num2),
3. double resta(1: double num1, 2: double num2),
4. double multiplicacion(1: double num1, 2: double num2),
5. double division(1: double num1, 2: double num2),
6. list sumavec(1: list vec1, 2: list vec2),
7. list restavec(1: list vec1, 2: list vec2),
8. list multiplicacionvec(1: list vec1, 2: list vec2),
9. list multiplicacionvec_esc(1: list vec, 2: double num),
10. double polinomio(1: list polinomio),

Las operaciones básicas devuelven un double y reciben como parámetro los dos operandos de la operación, también double. Las operaciones con array devuelven un array de doubles y reciben lo mismo como operandos de la operación. Por último, la función polinomio recibe una lista de operaciones, que eso luego se traducirá en C++ a un std::vector y en Python a una lista, es por esto que había que definir antes el tipo de dato operacion.

3.2 Cliente (cliente.cpp)

El funcionamiento del cliente es muy parecido al del ejercicio 1, pero con las particularidades de Apache Thrift.

Antes de realizar cualquier conexión al servidor, define varias variables para almacenar las distintas estructuras de datos que se pasan por terminal. El cliente gestiona todo lo relacionado con lo que introduce el usuario, dependiendo de si introduce una cosa u otra, se filtra para almacenar los datos en las variables correspondientes y llamar a las funciones del servidor que toque. Usa la variable flag_calc para saber qué tipo de operaciones hacer y qué tipo de variables almacenar, siendo 0 para números reales, 1 para vectores, 2 para vectores y escalares y 3 para polinomios.

En el caso de números reales los coge directamente de los argumentos del programa, pero en el caso de los vectores se usa la palabra clave vec para indicar que se van a introducir los valores, y por tanto en este caso se leen usando cin y se almacenan en el vector. Además, define varios comandos para mostrar información al usuario: op_polinomio. info, op_basicas, op_vectores y op_polinomio.

Una vez hecho esto, ya crea el socket y el protocolo de transporte, y con ellos el objeto cliente, a partir del cual se harán las llamadas a los métodos que son parte de este objeto, es decir, las funciones de la calculadora que ejecuta el servidor. Se abre la conexión, se hacen las llamadas a las funciones que toquen según se haya indicado por línea de comandos, y cuando se muestra el resultado se cierra la conexión y termina el programa.

Existe una particularidad, y es que en C++ los resultados se pasan por referencia para que sea más eficiente, por tanto las funciones definidas en el archivo thrift que devuelven vectores, al compilarlas con el compilador de Thrift se convierten en funciones void y el primer parámetro de esta función es el resultado pasado por referencia.

3.3 Servidor (servidor.py)

Define el comportamiento de las distintas operaciones que puede realizar la calculadora. Para resolver algunas de las operaciones se ha usado la biblioteca *numpy* de python.

Para las operaciones de números reales recibe un número, para las de vectores recibe un array definida y para los polinomios recibe la variable pol, un array de operaciones, que fueron definidos en el archivo

calculadora.thrift.

El servidor tiene dos partes, handler y main:

- En el **handler** se definen las distintas funciones de la calculadora, que son las que llamará el cliente.
- En el **main** se crea un objeto handler, se asocia al procesador y se crea el método de transporte. En el método de transporte hay que especificar el puerto y el cliente ya que Thrift no tiene binding automático como sí tenía Sun RPC. Se ha especificado *127.0.0.1* en lugar del DNS localhost puesto que si no daba warnings y buscando en la documentación se recomienda usar la IP en este caso. Finalmente se lanza el servidor con los parámetros indicados, que se queda operando infinitamente hasta que se apague.

Como se ha dicho, en C++ los valores se reciben por referencia como un parámetro de las distintas funciones, sin embargo esto en Python se abstrae por dos razones: en Python el paso es por valor, y además es el middelware de Apache Thrift el que gestiona los valores que se devuelven, por tanto en las funciones del servidor se devuelven los resultados usando *return* pese a que luego en la cabecera del cliente C++ veamos que estas funciones son void y no devuelven nada.