

Práctica 2 - Llamada remota a procedimiento (RPC)

Ejercicio 1: Calculadora

Jose Luis Gallego Peña - DSD2

1. Introducción

El ejercicio consiste en un desarrollar programa distribuido, una calculadora que realice varias operaciones. Todo ello utilizando Sun RPC.

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.

2. Compilación y ejecución

Para usar rpcgen he tenido que instalar los paquetes libtirpc-compatible y rpcsvc-proto puesto que no venían en mi sistema y sin ellos no podía usar las utilidades *rpcgen* y *rpcbind*.

Primero he generado el archivo **calculadora.x**, cuyo contenido se explicará en el próximo apartado. Con ese archivo, y estando en la carpeta que lo contiene, se ha ejecutado el siguiente comando:

```
rpcgen -Nc calculadora.x
```

Este comando genera los siguientes archivos:

- **calculadora_client.c**, 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.
- **calculadora_clnt.c**, stub del cliente que empaqueta los datos (agrupando argumentos y enviándolos al stub del servidor) y luego los desempaqueta para darselos al cliente.
- **calculadora_server.c**, 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.
- **Calculadora_svc.c**, stub del servidor que escucha al stub del cliente, desempaqueta los argumentos, hace las llamadas y devuelve al cliente los datos.
- **calculadora_xdr.c**, rutinas xdr para los tipos de datos definidos. Convierten datos al formato XDR y viceversa, lo cual hace que el programa pueda traducirse a varios lenguajes de programación ya que cada lenguaje tiene sus tipos de datos.
- **calculadora.h**, cabecera con definiciones comunes al cliente y servidor, en el que se pueden ver las variables y funciones definidas.
- **calculadora.x**, código en Sun RPC y parte más importante de todo el sistema. Define una interfaz, datos y rutinas accesibles remotamente.

La opción **N** sirve para generar el código pasando los argumentos por valor al estilo C, y poder pasar más de un parámetro en las funciones. Genera todo menos el cliente y el servidor. Es la opción a usar cuando se cambia algo en calculadora.x, como por ejemplo añadir una nueva función, y que no se sobrescriban los archivos de cliente y servidor que son los que hay que programar, el resto los genera RPC automáticamente y no hay que tocarlos. La opción **C** indica que genera el código en ANSI-C y **a** es para indicar que se generen plantillas de cliente y servidor (por tanto esto solo se hace la primera vez).

El funcionamiento consiste en abrir dos terminales, una ejecutando el cliente (con sus argumentos, entre ellos localhost, DNS de 127.0.0.1, porque hacemos referencia a nuestra máquina, ya que es un sistema distribuido en nuestra propia máquina y no en otra) 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, es necesario compilarlos. Rpcgen proporciona un makefile, sin embargo he preferido compilar ambos programas usando las siguientes ordenes de gcc:

```
gcc calculadora_client.c calculadora_clnt.c calculadora_xdr.c -o cliente -lnsl -ltirpc
gcc calculadora_server.c calculadora_svc.c calculadora_xdr.c -o servidor -lnsl -ltirpc
```

En mi caso he tenido que añadir la opción `-ltirpc` puesto que si no mi shell no me detectaba la biblioteca de rpc.

Una vez compilados ambos, se deben ejecutar en distintas terminales. Sin embargo, para que el servidor pueda ejecutarse y funcione hay que usar el comando `sudo rpcbind start` para iniciar el servicio `rpcbind` que gestiona las conexiones entre los programas, dando las direcciones.

Un ejemplo 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.

```
Practicas/P2/Calculadora
➔ ./servidor

Practicas/P2/Calculadora
➔ ./cliente localhost 1 + 5.2
1.000000 + 5.200000 = 6.200000

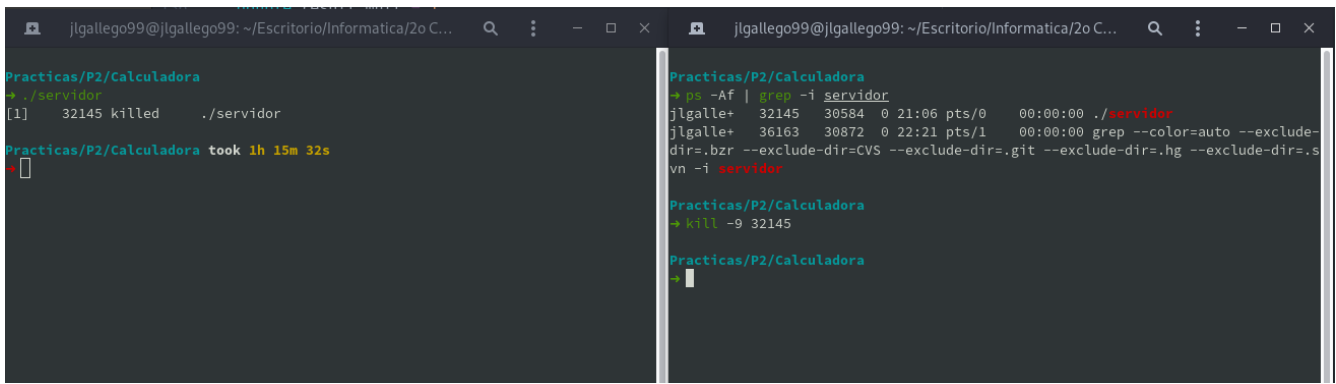
Practicas/P2/Calculadora
➔ ./cliente localhost 5 x vec
Introducir tamaño del vector: 3
Introducir valores en el vector 2: 2 5 1
( 2.000000 5.000000 1.000000 ) x 5.000000 = ( 10.000000 25.000000 5.000000 )

Practicas/P2/Calculadora took 5s
➔ ./cliente localhost vec x vec
Introducir tamaño de los vectores: 3
Introducir valores en el vector 1: 1 1 1
Introducir valores en el vector 2: 2 2 3
( 1.000000 1.000000 1.000000 ) x ( 2.000000 2.000000 3.000000 ) = ( 2.000000 2.000000 3.000000 )

Practicas/P2/Calculadora took 8s
➔ ./cliente localhost polinomio
MODO POLINOMIO. Introduzca primero un valor y luego un operador. Para terminar e
scriba "=" después de un valor.
1 + 2 x 2 - 1 x 2 =
3.000000

Practicas/P2/Calculadora took 6s
➔
```

Para cerrar el servidor es necesario matar al proceso puesto que no sirve usar la interrupción `Ctrl+C`. Para ello se ejecuta `bash 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 32145` para matar al proceso y así cerrar el servidor de forma correcta.



```
Practicas/P2/Calculadora
➔ ./servidor
[1] 32145 killed ./servidor
Practicas/P2/Calculadora took 1h 15m 32s
➔

Practicas/P2/Calculadora
➔ ps -Af | grep -i servidor
jlgalle+ 32145 30584 0 21:06 pts/0 00:00:00 ./servidor
jlgalle+ 36163 30872 0 22:21 pts/1 00:00:00 grep --color=auto --exclude-dir=.bzip --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn -i servidor
Practicas/P2/Calculadora
➔ kill -9 32145
Practicas/P2/Calculadora
➔
```

3. Explicación de la solución

3.1 Definición (calculadora.x)

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

Primero se han definido las variables y estructuras que se van a usar:

- **vec**, es un vector finito de números double, sirve para operar entre vectores y se pasa como parámetro en las funciones de suma, resta y multiplicación de vectores. Se ha usado `<>` en su definición para indicar que es un array sin un tamaño definido, que luego se definirá en el cliente reservando memoria en función del tamaño de los vectores que quiera el usuario.
- **operacion**, es una estructura con tres 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`, `/`.
- **polinomio**, es un vector de elementos del tipo *operacion* descrito anteriormente, ya que un polinomio comprende varios valores con sus operaciones asociadas. Este es el tipo de dato que se pasa en la función que calcula el valor de los polinomios. También se ha usado `<>` en su definición para indicar que es un array sin un tamaño definido.

Los tipos de dato **vec** y **polinomio** al ser vectores, Sun RPC los traduce a C como una estructura que comprende dos partes, el propio vector y su tamaño, lo cual hace más fácil el operar con ellos. En el cliente cuando se almacenen datos en vector, finalmente también se almacenará su tamaño en el campo correspondiente.

Además de estos tipos de datos que se usarán en la calculadora para encapsular y tratar de forma más cómoda los datos, se han definido dos uniones como tipo de datos que devolverán las funciones del servidor, ambas definen el resultado de las operaciones. Ambas, además, tienen un argumento **error** que se trata en el cliente para filtrar el que haya fallos, y en ese caso terminar el programa y no seguir.

- **calculadora_res**, en caso de que el error sea 0 (no hay error) define un tipo de dato double para el resultado;
- **calculadora_res_vec**, en caso de que el error sea 0 (no hay error) define un tipo de dato vec para el resultado.

Luego del `rpcgen`, estas uniones se transforman en C en un struct, con el propio campo *resultado* metido en un tipo union con nombre igual al tipo calculadora que se devuelve pero terminado en `u`. Por ejemplo si se quiere acceder al resultado como tal en esta estructura `calculadora_res`, se deberá hacer:

```
<tipo> resultado = result.calculadora_res_u.resultado;
```

Se ha creado un programa llamado *CALCPROG*, cuyo número de programa es el primero disponible (0x2000001) y que contiene una única versión con identificador 1, *CALCVER*. Esta versión define todas las funciones que puede realizar el servidor calculadora, y están numeradas del 1 al 9:

1. `calculadora_res SUMA(double, double)`
2. `calculadora_res RESTA(double, double)`
3. `calculadora_res MULTIPLICACION(double, double)`
4. `calculadora_res DIVISION(double, double)`
5. `calculadora_res_vec SUMAVEC(vec, vec)`

6. `calculadora_res_vec RESTAVEC(vec, vec)`
7. `calculadora_res_vec MULTIPLICACIONVEC(vec, vec)`
8. `calculadora_res_vec MULTIPLICACIONVEC_ESC(vec, double)`
9. `calculadora_res POLINOMIO(polinomio)`

3.2 Cliente (`calculadora_client.c`)

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 *scanf* y se almacenan en el vector con la memoria ya inicializada.

Además, define varios comandos para mostrar información al usuario: *info*, *op_basicas*, *op_vectores* y *op_polinomio*.

Finalmente se crea el cliente con *clnt_create*, se llama a las distintas funciones del servidor en función del tipo de los datos que ha introducido el usuario y del operador elegido y se muestran por terminal los resultados obtenidos del servidor mediante la variable *calculadora_res*.

3.3 Servidor (`calculadora_server.c`)

Define el comportamiento de las distintas operaciones que puede realizar la calculadora, en función de las cabeceras del archivo *calculadora.h*.

Para las operaciones de números reales recibe un valor *double* de C, para las de vectores recibe la variable *vec* definida en *calculadora.x* y para los polinomios recibe la variable *polinomio* también definida en el archivo *calculadora.x*.

En el caso de los vectores, se recorrerán los vectores de principio a fin, operando posición a posición y almacenando en el resultado.

En el caso de los polinomios, se recorren los valores del polinomio, analizando las operaciones que tiene este valor detrás y delante, para así poder preservar la jerarquía de operaciones. Va sumando los distintos resultados parciales de cada iteración en una variable, y está el caso especial de las multiplicaciones y divisiones, que almacenan sus resultados parciales en unas variables propias para así poder encadenar multiplicaciones o divisiones. Una vez que se ha terminado una cadena de varias multiplicaciones o divisiones, se añadirá al resultado parcial sumando. Además, se tienen en cuenta si los números tienen un - antes para indicar que son negativos.

Las operaciones se realizan y van almacenando finalmente en una variable resultado, de tipo *calculadora_res* para devolver números reales, o de tipo *calculadora_res_vec* para devolver un vector. Esto es lo que obtiene el cliente.