

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

6-4-2022

Práctica 2.1: Sun RPC

Calculadora con operaciones
básicas y complejas

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and curve upwards and to the right.

Alberto Llamas González
3º GII, IS

1. Introducción

La primera parte de la práctica, consiste en realizar una calculadora usando Sun RPC. En ella, el cliente envía una serie de datos al servidor quien realiza las operaciones correspondientes y devuelve el resultado al cliente, el cual lo imprime por pantalla.

En principio, se exigían unas operaciones mínimas básicas (suma, resta, multiplicación y división) y se han añadido algunas operaciones más complejas.

2. Calculadora básica

En primer lugar, se ha implementado una calculadora con las operaciones básicas, suma, resta, multiplicación y división. Para ello, se introducirán como argumentos de la llamada de cliente los números a operar y la operación a realizar. En el archivo de definiciones (.x) se definieron las siguientes funciones:

```
double SUMA(double, double) = 1;  
double RESTA(double, double) = 2;  
double MULTIPLICACION(double, double) = 3;  
double DIVISION(double, double) = 4;
```

Se ha usado double, por si el usuario decide utilizar decimales y para el caso de la división, donde podemos encontrarnos con un resultado no entero. Además, en el caso de la multiplicación, para poder utilizar el operando *, debemos introducirlo por teclado de la siguiente forma -> '*' (sin dejar espacios, se ha dejado espacios por claridad de lectura).

Cuando generamos las plantillas se crean las funciones en el servidor, pero sin funcionalidad alguna por lo que debemos añadírsela. En el caso de la suma tendríamos lo siguiente.

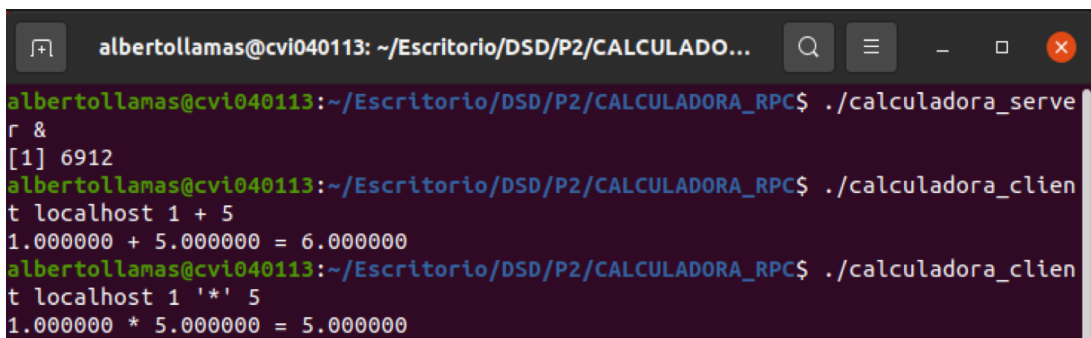
```
double *  
suma_1_svc(double arg1, double arg2, struct svc_req *rqstp)  
{  
    static double result;  
  
    result = arg1 + arg2;  
  
    return &result;  
}
```

En el cliente, con la función `calculadora_1(char *host, double numIzquierda, char operacion, double numDerecha)` invocamos a las diferentes funciones del servidor (en este caso base) dependiendo de la operación que el usuario quiera.

Ejemplo de uso:

`./calculadora_server &` (Para ejecutar el servidor en segundo plano)

`./calculadora_client localhost num1 operando num2` (Para ejecutar el cliente)



```
albertollamas@cvi040113: ~/Escritorio/DSD/P2/CALCULADO...  
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$ ./calculadora_server &  
[1] 6912  
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$ ./calculadora_client localhost 1 + 5  
1.000000 + 5.000000 = 6.000000  
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$ ./calculadora_client localhost 1 '*' 5  
1.000000 * 5.000000 = 5.000000
```

3. Aumentando la complejidad

Una vez realizadas las operaciones básicas, se decidió realizar operaciones algo más complejas como una ecuación de segundo grado u operaciones con vectores. Se añadió en el archivo de definiciones (.x) las siguientes operaciones:

```
ecuacionCuadrada CUADRATICA(double, double, double) = 5;  
vect SUMAVECTORES(vect,vect) = 6;  
vect RESTAVECTORES(vect,vect) = 7;  
vect PRODUCTOVECTORES(vect,vect) = 8;  
vect DIVISIONVECTORES(vect,vect) = 9;  
double PRODUCTOESCALAR(vect,vect) = 10;
```

Como podemos ver, tenemos dos tipos de datos nuevos; **vect** y **ecuacionCuadrada**. **vect** es un array de tamaño no definido mientras que **ecuacionCuadrada** es un struct que se define de la siguiente forma.

```
struct ecuacionCuadrada{  
    double a;  
    double b;  
    double c;  
    double res1;  
    double res2;  
};  
  
typedef double vect<>;
```

Para operar con vectores, se debe introducir como argumento lo siguiente

`./calculadora_client localhost vector`

lo cual inicia una secuencia de introducción de vectores y un operando para realizar la operación correspondiente entre dos vectores. Veamos la implementación de la suma entre vectores.

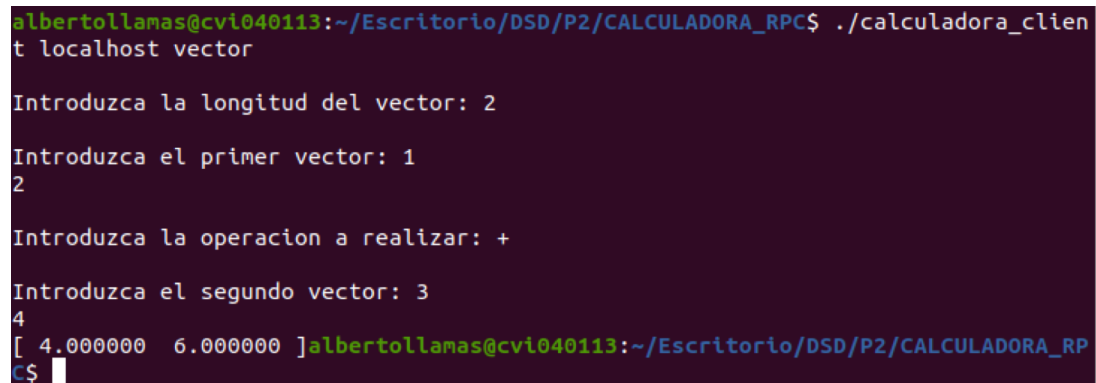
```
vect *
sumavectores_1_svc(vect arg1, vect arg2, struct svc_req *rqstp)
{
    static vect result;

    result.vect_len = arg1.vect_len;
    result.vect_val = malloc(result.vect_len * sizeof(double));

    for (int i = 0; i < result.vect_len; i++)
        result.vect_val[i] = arg1.vect_val[i] + arg2.vect_val[i];

    return &result;
}
```

Como vemos, es necesario, alojar en memoria el vector resultado, el cual se liberará cuando devuelva el resultado por pantalla, en el cliente.



```
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$ ./calculadora_client localhost vector

Introduzca la longitud del vector: 2

Introduzca el primer vector: 1
2

Introduzca la operacion a realizar: +

Introduzca el segundo vector: 3
4
[ 4.000000 6.000000 ]albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$
```

Si deseamos realizar una operación de segundo grado, simplemente debemos introducir lo siguiente

./calculadora_client localhost ecuacion a b c

donde a,b y c son respectivamente- $\rightarrow ax^2 + bx + c = 0$

Veamos la implementación del método.

```
ecuacionCuadrada *
cuadratica_1_svc(double arg1, double arg2, double arg3, struct svc_req
*rqstp)
{
    static ecuacionCuadrada result;
    result.a = arg1; result.b = arg2; result.c = arg3;

    if (result.a != 0){
        double discriminante = pow(result.b, 2.0) - 4 * result.a *
result.c;
        if (discriminante > 0) {
            result.res1 = (-result.b + sqrt(discriminante)) / (2 *
result.a);
            result.res2 = (-result.b - sqrt(discriminante)) / (2 *
result.a);
        } else if (discriminante == 0){
            result.res1 = result.res2 = (-result.b) /
(2*result.a);
        } else {
            printf("\nNo tiene raices reales");
            result.res1 = result.res2 = INT_MIN;
        }
    } else {
        printf("\nInfinito");
        result.res1 = result.res2 = INT_MIN;
    }
}
```

En este caso, si no tenemos resultado real, no vamos a imprimir nada en el cliente.

```
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$ ./calculadora_client
localhost ecuacion 1 -5 6
ax2 + bx + c = 0
Los resultados de la ecuacion de segundo grado son: 3.000000 , 2.000000
albertollamas@cvi040113:~/Escritorio/DSD/P2/CALCULADORA_RPC$
```

4. Conclusiones

Concluyendo, vemos que es bastante tedioso el uso de C sobre todo cuando utilizamos vectores, ya que es necesario alojarlos en memoria lo que hace que tengamos más problemas en la implementación. Las operaciones se realizan de forma correcta, aunque como veremos en la siguiente parte, utilizando Apache Thrift, todo es más sencillo.

5. Bibliografía

- Apuntes proporcionados por los profesores de la asignatura
- <https://stackoverflow.com/questions/5240789/scanf-leaves-the-newline-character-in-the-buffer>
- <https://stackoverflow.com/questions/10409032/why-am-i-getting-undefined-reference-to-sqrt-error-even-though-i-include-math>
- https://docs.oracle.com/cd/E18752_01/html/816-1435/rpcproto-24229.html