



Práctica 2

Llamada remota a procedimiento (RPC)

Miguel Muñoz Molina DSD2

Introducción

Para la realización de la calculadora, he comenzado creando el archivo calculator.x, que contiene nuestro programa principal, asignándole una versión y añadiendo los métodos que usaremos para las operaciones disponibles (+, -, x, /, ^).

```
program CALCULADORA {  
    version CALCULADORAVER {  
        double SUMA(double,double) = 1;  
        double RESTA(double,double) = 2;  
        double MULTIPLICACION(double,double) = 3;  
        double DIVISION(double,double) = 4;  
        double POTENCIA(double,double) = 5;  
    } = 1;  
} = 0x20000155;
```

A estas operaciones les pasaremos 2 números decimales, y nos devolverá el número decimal resultante del cálculo correspondiente.

Para compilarlo haremos:

```
> rpcgen -NCa calculator.x
```

Esto nos generará los distintos archivos para poder realizar la práctica.

Tras eso, debemos compilar el cliente y el servidor.

```
> cc calculator_client.c calculator_clnt.c calculator_xdr.c -o cliente -lnsl
```

```
> cc calculator_server.c calculator_svc.c calculator_xdr.c -o servidor -lnsl -lm
```

En el caso del servidor he añadido el “-lm” ya que utilizo la función pow, que necesita la librería math.

Una vez hecho esto, tendremos los dos ejecutables cliente y servidor.

En una terminal lanzaremos el servidor, asegurándonos que hemos hecho rpcbind start para iniciar el proceso.

En otra terminal haremos ./cliente localhost resto_de_argumentos

PASO 1

Lo primero que realicé fue una calculadora básica con suma, resta, multiplicación y división, en la que se pasaban dos números y se devolvía su resultado. Cree las 4 funciones básicas y un pequeño programa que recibía 3 argumentos (sumando el “localhost”), siendo dos de ellos números y otro un operador de los nombrados anteriormente.

El resultado dado era así:

```
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 3 + 2
3.000000 + 2.000000 = 5.000000
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 3 - 2
3.000000 - 2.000000 = 1.000000
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 3 x 2
3.000000 x 2.000000 = 6.000000
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 3 / 2
3.000000 / 2.000000 = 1.500000
```

Como vemos, la multiplicación he tenido que indicarla con ‘x’ y no con ‘*’ debido a que el símbolo ‘*’ no es reconocido como un argumento en la terminal de linux.

PASO 2

Decidí añadirle más operaciones a la calculadora, por ello, haciendo uso de la librería math he añadido las potencias, indicada con un ‘^’. Para ello, añadí una función extra al calculator.x y la incorporé al cliente y servidor.

Es por ello que en la compilación del servidor he tenido que añadir a la orden ‘-lm’ ya que si no, no reconocía la expresión ‘pow’ para hacer potencias.

El resultado dado era así:

```
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 3 ^ 2
3.000000 ^ 2.000000 = 9.000000
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 5 ^ -2
5.000000 ^ -2.000000 = 0.040000
```

PASO 3

En este momento decidí darle más complejidad a la calculadora, haciendo que en vez de recibir sólo 2 números y un operando pudiera recibir infinitos, permitiendo así introducir ecuaciones.

Para ello, tuve que cambiar la forma de recibir argumentos. Lo primero es comprobar que recibimos el número de argumentos correctos.

```
if (argc < 5 || argc%2 == 0) {  
    printf ("usage: %s server_host integer (operation integer)...\n", argv[0]);  
    exit (1);  
}
```

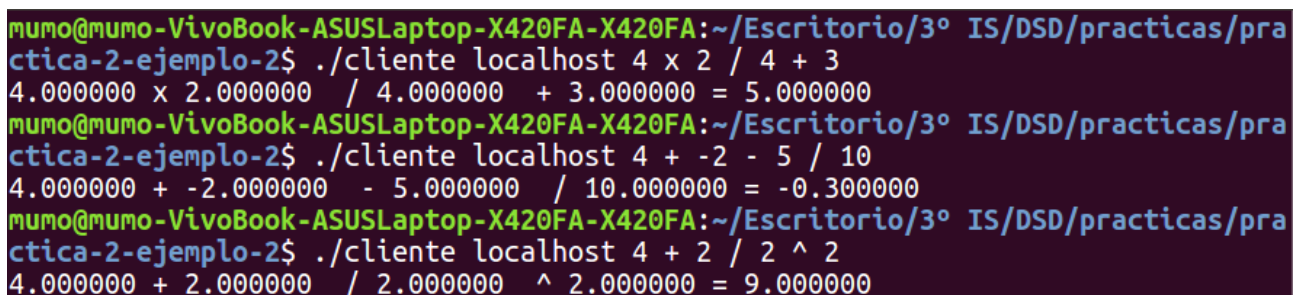
Por una parte `argc < 5` nos permite asegurar que al menos habrá dos números y un operando. A partir de ahí, el número de argumentos debe ser 5, 7, 9, 11, etc; es decir, un número impar.

Tras esto decidí guardar tanto los números como los operadores en 2 arrays distintos.

```
int utils=0;  
for (int ii=3; ii < argc; ii += 2){  
    operation[utils] = *(argv[ii]);  
    utils++;  
}  
utils = 0;  
for (int ii=2; ii < argc; ii += 2){  
    numeros[utils] = atoi(argv[ii]);  
    utils++;  
}
```

Una vez recibido ambos vectores, la único que nos queda es gestionar las distintas operaciones recorriéndolos y llamando a las funciones correspondientes del servidor.

El resultado dado era así:



```
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 4 x 2 / 4 + 3  
4.000000 x 2.000000 / 4.000000 + 3.000000 = 5.000000  
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 4 + -2 - 5 / 10  
4.000000 + -2.000000 - 5.000000 / 10.000000 = -0.300000  
mumo@mumo-VivoBook-ASUSLaptop-X420FA-X420FA:~/Escritorio/3º IS/DSD/practicas/practica-2-ejemplo-2$ ./cliente localhost 4 + 2 / 2 ^ 2  
4.000000 + 2.000000 / 2.000000 ^ 2.000000 = 9.000000
```

No se siguen las prioridades en los distintos operadores, van de izquierda a derecha