

Practica 2 RPC Sun



**UNIVERSIDAD
DE GRANADA**

José Luis Molina Aguilar

21 de marzo de 2023

Curso 2022-2023
Correo : joselu201@correo.ugr.es

Índice

1	Descripción	3
2	Compilación	4
3	Ejecucion	4
3.1	Ejecucion del Servidor	4
3.2	Ejecución del Cliente	4
4	Ejemplos de ejecución	6

1. Descripción

Desarrollo de una calculadora siguiendo un esquema de sistema distribuido del tipo cliente-servidor donde el cliente sera el que realice el input de datos y operaciones y se los envíe al servidor para que procese dichos datos, una vez completados, el servidor responderá con el resultado de la operación al cliente para mostrar por pantalla el resultado.

Se ha realizado la implementación de la calculadora de números tipo `double` de la siguiente forma.

```
const MAX_SIZE = 100;
struct operandos{
    double values<MAX_SIZE>;
};
```

Esto nos permite representar simples números doubles ademas de arrays dinámicos del mismo tipo ya que RPC realiza la traducción de esta estructura a esta en C.

```
#define MAX_SIZE 100

struct operandos {
    struct {
        u_int values_len;
        double *values_val;
    } values;
};
typedef struct operandos operandos;
```

De esta forma podemos realizar de una forma sencilla todas las operaciones básicas (suma, resta, multiplicación y division) tanto para números como para arrays.

Vamos a definir la calculadora, para ello simplemente necesitaremos los dos operandos, ya sean solo doubles o vectores y en mi caso cada uno de las operaciones que realiza.

```
program CALCULADORA {
    version CALCULADORA_1 {
        operandos suma(operandos op1, operandos op2) = 1;
        operandos restar(operandos op1, operandos op2) = 2;
        operandos multiplica(operandos op1, operandos op2) = 3;
        operandos divide(operandos op1, operandos op2) = 4;
    } = 1;
} = 0x20000001;
```

Ademas para realizar una ampliación he realizado otro program un poco mas complejo en el que definimos la operación de producto escalar y producto vectorial

```
program CALC_VEC_COMPLEJO {  
    version CALC_VEC_COMPLEJO.1 {  
        operandos producto_escalar(operandos op1, operandos op2) = 1;  
        operandos producto_vectorial(operandos op1, operandos op2) = 2;  
    } = 1;  
} = 0x20000002;
```

La idea es que la calculadora normal llame a este tipo de calculadora de operaciones complejas sobre vectores cuando le indique el tipo de operación.

2. Compilación

Necesitaremos tener activo el servicio de rpcbin, para ello:

```
$ sudo service rpcbind start
```

Ahora podemos mediante rpcgen y el archivo con extension .x compilar el archivo y generar automaticamente una estructura tipo C.

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

Una vez que hacemos esto simplemente haremos make, con un makefile que se genera automáticamente para obtener los ejecutables.

```
$ make -f Makefile.calculadora
```

3. Ejecucion

3.1. Ejecucion del Servidor

Para la ejecucion del servidor simplemente en un terminal escribir.

```
$ ./calculadora_server
```

Se nos quedara esperando la recepción de mensajes de parte del cliente.

3.2. Ejecución del Cliente

```
$ ./calculadora_cliente localhost <operando_1> <operador> <operando_2>
```

Donde operando pueden ser:

- entero
- double
- vectores

Por la parte de del operador podemos diferenciar los siguiente:

- `+` : Para realizar la suma tanto de valores simples como de vectores.
- `-` : Para realizar la resta tanto de valores simples como de vectores.
- `*` : Para realizar la multiplicacion tanto de valores simples como de vectores.
- `/` : Para realizar la division tanto de valores simples como de vectores.

Para la parte de operaciones complejas con vectores tenemos

- `x` : Para realizar el producto escalar
- `c` : Para realizar el producto vectorial entre 2 vectores, recordar que el producto vectorias es solo definido para vectores en un espacio 3D, por lo que solo se permite vecotores de 3 elementos

Sabiendo esto podemos ver algunos ejemplos de como ejecutar el cliente de forma correcta.

```
$ ./calculadora_cliente localhost 10 + 10 # Suma normal
$ ./calculadora_cliente localhost 10 11 12 + 13 14 15 # Suma de vectores
$ ./calculadora_cliente localhost 10 x 2 3 4 5 #Producto escalar
$ ./calculadora_cliente localhost 2 3 4 c 5 4 3 #Producto vectorial
```

Como podemos ver si ponemos varios elementos delante del operador, esto lo interpretaremos como un vector, de esta forma hacemos mucho mas flexible el paso de valores por parte del cliente.

Para la recepción de los datos utilizaremos los parámetros que le pasamos al cliente, el primer argumento que le pasamos sera el host, determinaremos el primer operando de la siguiente forma, todos los argumentos que encontremos entre el host y el operador pertenecerán al primer operando, esto puede ser un solo elemento o un conjunto de estos, el cual determinaría un vector. Justo después estaría el operador como hemos dicho antes y desde ese operador hasta el final de argc el segundo operando.

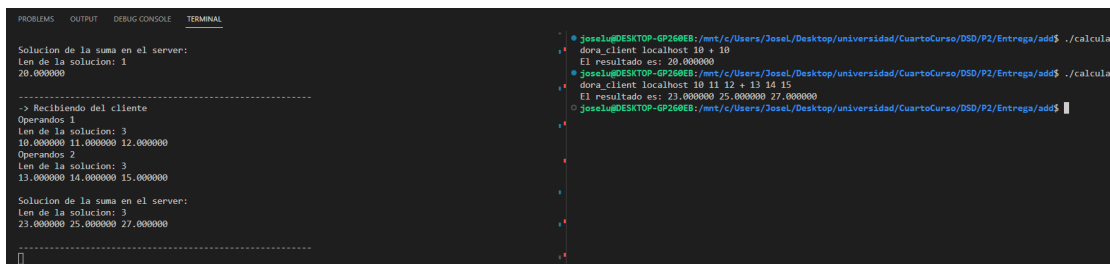
4. Ejemplos de ejecución

Vamos a ver un par de ejemplos, el terminal derecho sera el server, mientras que el izqdo sera el cliente.



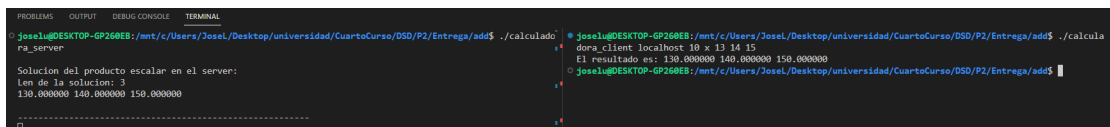
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calculado
ra_server
-> Recibiendo del cliente
Operandos 1
Len de la solucion: 1
10.000000
Operandos 2
Len de la solucion: 1
10.000000
Solucion de la suma en el server:
Len de la solucion: 1
20.000000
-----
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calcula
dora_client localhost 10 + 10
El resultado es: 20.000000
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$
```

Figura 4.1: Prueba suma




```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Solucion de la suma en el server:
Len de la solucion: 1
20.000000
-----
-> Recibiendo del cliente
Operandos 1
Len de la solucion: 3
10.000000 11.000000 12.000000
Operandos 2
Len de la solucion: 3
13.000000 14.000000 15.000000
Solucion de la suma en el server:
Len de la solucion: 3
23.000000 25.000000 27.000000
-----
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calcula
dora_client localhost 10 * 10
El resultado es: 100.000000
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calcula
dora_client localhost 10 * 12 + 13 * 14
El resultado es: 120.000000 182.000000
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$
```

Figura 4.2: Prueba multiplicación



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calculado
ra_server
Solucion del producto escalar en el server:
Len de la solucion: 3
130.000000 140.000000 150.000000
-----
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calcula
dora_client localhost 10 11 12 13 14 15
El resultado es: 130.000000 140.000000 150.000000
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$
```

Figura 4.3: Prueba escalar



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calculado
ra_server
-> Recibiendo del cliente
Operandos 1
Len de la solucion: 3
10.000000 2.000000 3.000000
Operandos 2
Len de la solucion: 3
13.000000 14.000000 15.000000
Result len: 3
Solucion del cross product en el server:
Len de la solucion: 3
-12.000000 -111.000000 114.000000
-----
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$ ./calcula
dora_client localhost 10 2 3 13 14 15
El resultado es: -12.000000 -111.000000 114.000000
joselu@DESKTOP-GP268EB:/mnt/c/Users/Josel/Desktop/universidad/CuartoCurso/DSD/P2/Entrega/adt$
```

Figura 4.4: Prueba cross product