Práctica 2 DSD

Nombre: Ana López Mohedano
Curso: 3º / Grupo: 2

Esta práctica consiste en realizar una calculadora que recibe un tipo de operación, envía los datos al servidor, que procesará los datos y hará los cálculos necesarios, y devolverá el resultado al cliente.
En mi caso, he hecho una calculadora interactiva con un menú por consola, con las operaciones básicas de suma, resta, multiplicación y división, y como funcionalidad adicional, he añadido como estructura vectores, y la calculadora sumará los elementos del vector, devolviendo un vector resultado del mismo tamaño que los otros.

Fichero .x

Con calculadora.x, hemos definido las 5 funciones que realizará la calculadora.

```
typedef double vect<5>;

struct result {
    double valor_resultado;
    int code;
};


/* la siguiente union se utiliza para discriminar entre llamadas con e
union readdir_res switch (int errno) {
    case 0:
        double result; /* sin error */
    default:
        void; /* con error: nada */
};

program CALCULADORA {
    version CALCULADORA_1 {
        result suma(double n1, double n2) = 1;
        result multiplica(double n1, double n2) = 2;
        result divide(double n1, double n2) = 3;
        result resta(double n1, double n2) = 4;
        vect suma_vectores(vect n1, vect n2) = 5;
    } =1;
} = 0x20000156;
```

También hemos definido lo que devolverán las funciones. Primero tenemos un struct result, que devolverá el valor del resultado y un código de error, si lo hubiese. El valor del resultado será un double, ya que la suma es de elementos double. Segundo, hemos definido un

vector con 5 elementos con “typedef double vect<5>”, que será los valores que recibe la función y que devuelve.

Cliente

En el cliente habrá una función calculadora_1, que recibirá el host, los dos valores y el tipo de operación (suma, resta, multiplicación o división). También tendrá una calculadora_2 para la operación con vectores, recibirá el host, los dos vectores, y devolverá un vector.

```
double
calculadora_1(char *host, double n1, double n2, int tipo)
{
    CLIENT *clnt;
    result  *result_1;
    // vect  *result_5;
    // vect  suma_vectores_1_n1;
    // vect  suma_vectores_1_n2;

#ifndef DEBUG
    clnt = clnt_create (host, CALCULADORA, CALCULADORA_1, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif  /* DEBUG */

    if(tipo == suma_tipo)
        result_1 = suma_1(n1, n2, clnt);
        if (result_1 == (result *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    else if(tipo == multiplica_tipo){
        result_1 = multiplica_1(n1, n2, clnt);
        if (result_1 == (result *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    }
    else if(tipo == divide_tipo){
        result_1 = divide_1(n1, n2, clnt);
        if (result_1 == (result *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    }
    else if(tipo == resta_tipo){
        result_1 = resta_1(n1, n2, clnt);
        if (result_1 == (result *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    }
```

```c
vect*
calculadora_2(char *host, vect n1, vect n2)
{
    CLIENT *clnt;
    // result  *result_1;
    vect  *result;
    // vect  suma_vectores_1_n1;
    // vect  suma_vectores_1_n2;

#ifndef DEBUG
    clnt = clnt_create (host, CALCULADORA, CALCULADORA_1, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif  /* DEBUG */

    // if(tipo == vector_tipo){
        result = suma_vectores_1(n1, n2, clnt);
        if (result == (vect *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    // }

#ifndef DEBUG
    clnt_destroy (clnt);
#endif   /* DEBUG */

    return result;
}
```

Servidor

El servidor se encargará de realizar las operaciones, con una función por cada operación posible.

```c
#include "calculadora.h"

result *
suma_1_svc(double n1, double n2,  struct svc_req *rqstp)
{
    static result  result;

    result.valor_resultado = n1+n2;
    result.code = 0;

    return &result;
}
```

Funcionamiento

Para generar los archivos a partir del fichero .x, usaremos el comando rpcgen -NCa calculadora.x, y se generarán los archivos correspondientes. Para la compilación, usaremos make -f Makefile.calculadora. Este makefile se habrá creado automáticamente con el comando rpcgen usado anteriormente.

```
ana@ana:~/UNI/2022_2023/DSD/Practicas_DSD/Practica2_DSD$ make -f Makefile.calcul
adora
cc -g    -c -o calculadora_clnt.o calculadora_clnt.c
cc -g    -c -o calculadora_client.o calculadora_client.c
cc -g    -c -o calculadora_xdr.o calculadora_xdr.c
cc -g     -o calculadora_client  calculadora_clnt.o calculadora_client.o calcula
dora_xdr.o -lnsl
cc -g    -c -o calculadora_svc.o calculadora_svc.c
cc -g    -c -o calculadora_server.o calculadora_server.c
cc -g     -o calculadora_server  calculadora_svc.o calculadora_server.o calculad
ora_xdr.o -lnsl
ana@ana:~/UNI/2022_2023/DSD/Practicas_DSD/Practica2_DSD$
```

Una vez compilado, abriremos dos terminales, una con el servidor que se quedará ejecutando, y otra con el cliente, que es el que mostrará el menú de la calculadora.

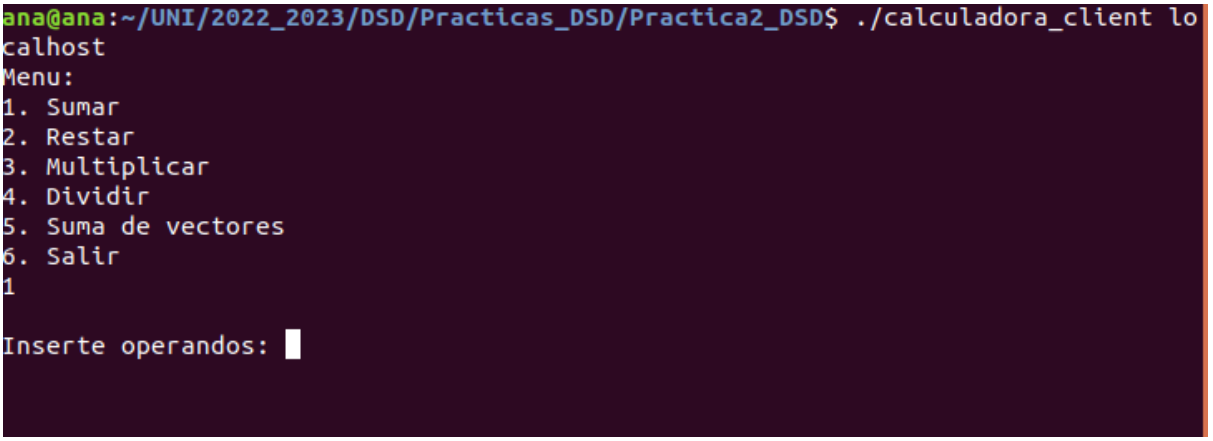
Para el cliente: $ ./calculadora_client localhost

```
ana@ana:~/UNI/2022_2023/DSD/Practicas_DSD/Practica2_DSD$ ./calculadora_client lo
calhost
Menu:
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Suma de vectores
6. Salir
```

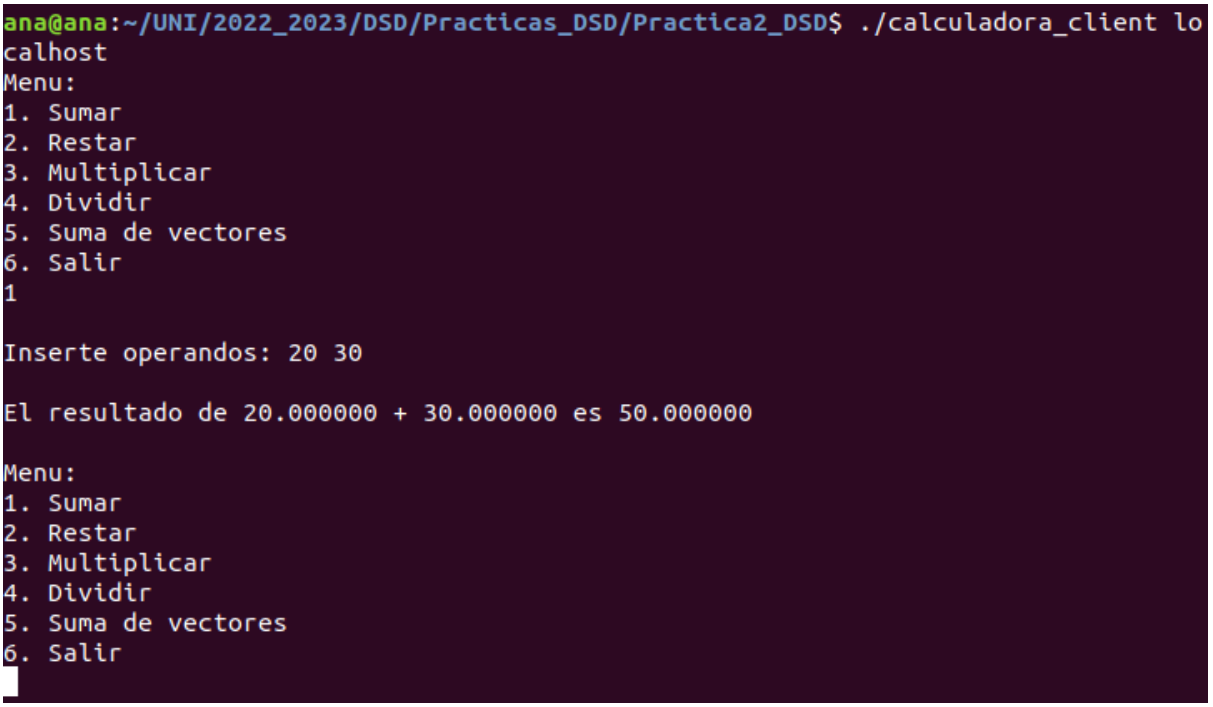
Para el servidor: $ ./calculadora_server

```
cc -g    -c -o calculadora_client.o calculadora_client.c
cc -g    -c -o calculadora_xdr.o calculadora_xdr.c
cc -g     -o calculadora_client  calculadora_clnt.o calculadora_client.o calcula
dora_xdr.o -lnsl
cc -g    -c -o calculadora_svc.o calculadora_svc.c
cc -g    -c -o calculadora_server.o calculadora_server.c
cc -g     -o calculadora_server  calculadora_svc.o calculadora_server.o calculad
ora_xdr.o -lnsl
ana@ana:~/UNI/2022_2023/DSD/Practicas_DSD/Practica2_DSD$ ./calculadora_server
```

Mostrará un menú con las operaciones, por ejemplo, para la suma, introducimos 1:



Pedirá que introduzcamos los operandos.



Una vez introducidos, mostrará el resultado y volverá a aparecer el menú por si quiere seguir realizando operaciones. Probaremos la suma de vectores.

```
Inserte operandos: 20 30

El resultado de 20.000000 + 30.000000 es 50.000000

Menu:
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Suma de vectores
6. Salir
5

Inserte 5 elementos del primer vector:
```

Pedirá los elementos de los vectores a sumar.

```
Menu:
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Suma de vectores
6. Salir
5

Inserte 5 elementos del primer vector: 1 2 3 4 5
Vector 1: 1.000000 2.000000 3.000000 4.000000 5.000000
Inserte 5 elementos del segundo vector: 5 4 3 2 1
Vector 2: 5.000000 4.000000 3.000000 2.000000 1.000000

El vector resultado es 6.000000 6.000000 6.000000 6.000000 6.000000

Menu:
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Suma de vectores
6. Salir
```

Una vez introducidos, mostrará el vector resultado, con los elementos sumados.