

Practica 2 Sun RPC

Para la práctica he hecho dos calculadoras, las he dejado por separado para que no sea tan lioso.

- La primera es la calculadora básica que hace la suma, la resta, la multiplicación y la división. Además también le he añadido el resto ya que la división solo te daba el cociente, y la potencia, para que sea un poco diferente.
- La segunda calculadora es una calculadora de fracciones, le metemos dos fracciones y podemos sumarlas, restarlas, multiplicarlas o dividir las.

¿Cómo he creado las calculadoras?

- calculadora.x

Primero he hecho el .x de ambas calculadoras cada uno en una carpeta:

- Este es el de la calculadora básica:

```
1 struct inputs{
2     float num1;
3     float num2;
4     char operador;
5 };
6
7 program CALCULADORA_PROG{
8     version CALCULADORA_VER{
9         float ADD(inputs)=1;
10        float SUB(inputs)=2;
11        float MUL(inputs)=3;
12        float DIV(inputs)=4;
13        float RES(inputs)=5;
14        float POT(inputs)=6;
15
16    }=1;
17 }=0x2fffffff;
```

- Este es el de la calculadora de las fracciones:

```
1 struct inputs{
2     float num1;
3     float num2;
4     float num3;
5     float num4;
6     char operador;
7 };
8
9 program CALCULADORA_PROG{
10    version CALCULADORA_VER{
11        float ADD(inputs)=1;
12        float SUB(inputs)=2;
13        float MUL(inputs)=3;
14        float DIV(inputs)=4;
15
16    }=1;
17 }=0x2fffffff;
```

Después de crearlo pongo en la terminal el siguiente comando (tengo que tener descargado rpcgen):

```
claudiasalado@claudiasalado-VirtualBox:~/Desktop/DSD/P2_SUNRPC$ rpcgen -a -C calculadora.x
```

Y me crea varios archivos entre ellos calculadora_client.c y calculadora_server.c que son los que tengo modificar.

- calculadora_server.c

Aquí es donde declaro que hace cada método es decir como hace la suma la resta etc.

- En la calculadora básica el server me queda tal que así:

```
 7 #include "calculadora.h"
 8 #include <math.h>
 9 #include <stdio.h>
10
11 float *
12 add_1_svc(inputs *argp, struct svc_req *rqstp)
13 {
14     static float result;
15
16     //Ponemos aqui nuestro codigo para la suma
17     result = argp->num1+argp->num2;
18     printf("Got Request : Suma %f y %f \n",argp->num1,argp->num2);
19     printf("Sent Response : %f \n", result);
20
21     return &result;
22 }
23
24 float *
25 sub_1_svc(inputs *argp, struct svc_req *rqstp)
26 {
27     static float result;
28
29     //Ponemos aqui nuestro codigo para la resta
30     result = argp->num1-argp->num2;
31     printf("Got Request : Resta %f y %f \n",argp->num1,argp->num2);
32     printf("Sent Response : %f \n", result);
33
34
35     return &result;
36 }
37
38 float *
39 mul_1_svc(inputs *argp, struct svc_req *rqstp)
40 {
41     static float result;
42
43     //Ponemos aqui nuestro codigo para la multiplicacion
44     result = argp->num1*argp->num2;
45     printf("Got Request : Multiplca %f y %f \n",argp->num1,argp->num2);
46     printf("Sent Response : %f \n", result);
47
48     return &result;
49 }
..
```

```

51 float *
52 div_1_svc(inputs *argp, struct svc_req *rqstp)
53 {
54     static float result;
55
56     //Ponemos aqui nuestro codigo para la division
57     result = argp->num1/argp->num2;
58     printf("Got Request : Divide %f y %f \n",argp->num1,argp->num2);
59     printf("Sent Response : %f \n", result);
60
61     return &result;
62 }
63
64 float *
65 res_1_svc(inputs *argp, struct svc_req *rqstp)
66 {
67     static float result;
68     static int aux;
69     static int m;
70
71     //Ponemos aqui nuestro codigo para la division
72     aux=argp->num1/argp->num2;
73     printf(" %f \n", aux);
74
75     m=argp->num2*aux;
76     result=argp->num1-m;
77     printf("Got Request : La raiz %f y %f \n",argp->num1,argp->num2);
78     printf("Sent Response : %f \n", result);
79
80     return &result;
81 }
82
83 float *
84 pot_1_svc(inputs *argp, struct svc_req *rqstp)
85 {
86     static float result;
87
88     //Ponemos aqui nuestro codigo para la division
89     for(int i=0; i<argp->num2; i++)
90         result=argp->num1*argp->num1;
91     printf("Got Request : Eleva %f a %f \n",argp->num1,argp->num2);
92     printf("Sent Response : %f \n", result);
93
94     return &result;
95 }

```

- En la calculadora de fracciones el server me queda tal que así:

He tenido que hacer dos funciones a parte que son el mcd y el mcm, el máximo común divisor lo necesito para hacer el mínimo común múltiplo, este último se utiliza para calcular la suma y resta de fracciones.

```

7 #include "calculadora.h"
8
9 //Creo un metodo que haga el MCD que nos sirve para sacar el MCM
10 float maximo_comun_divisor(float a, float b) {
11     float temporal; //Para no perder b
12     float aux;
13     while (b != 0) {
14         temporal = b;
15         aux = a/b;
16         b = a-(aux*b);
17         a = temporal;
18     }
19     return a;
20 }
21
22 //Hago este metodo del MCM para que sea mas sencillo hacer la suma y la resta
23 float minimo_comun_multiplo(float a, float b) {
24     return (a * b) / maximo_comun_divisor(a, b);
25 }
26
27
28 //SUMA
29 float *add_1_svc(inputs *argp, struct svc_req *rqstp){
30     static float result;
31
32     //Todo lo necesario para sumar fracciones
33     float mcm=minimo_comun_multiplo(argp->num2,argp->num4); //Calcula el mcm de los denominadores
34     float m1=mcm/argp->num2*argp->num1; //Calcula el numerador de la primera fraccion
35     float m2=mcm/argp->num4*argp->num3; //Calcula el numerador de la segunda fraccion
36     float m=m1+m2; //Calcula el numerador total
37
38     //Ponemos aqui nuestro codigo para la suma
39     result = m/mcm;
40     printf("Got Request : Suma %f/%f y %f/%f \n",argp->num1,argp->num2,argp->num3,argp->num4);
41     printf("Sent Response : %f/%f = %f \n",m,mcm,result);
42
43     return &result;
44 }
45

```

```

1 //RESTA
2 float *sub_1_svc(inputs *argp, struct svc_req *rqstp){
3     static float result;
4
5     //Todo lo necesario para la resta de Fracciones
6     float mcm=minimo_comun_multiplo(argp->num2,argp->num4); //Calcula el mcm de los denominadores
7     float n1=mcm/argp->num2*argp->num1; //Calcula el numerador de la primera fraccion
8     float n2=mcm/argp->num4*argp->num3; //Calcula el numerador de la segunda fraccion
9     float m=n1-n2; //Calcula el numerador total
10
11
12     //Ponemos aquí nuestro código para la resta
13     result = m/mcm;
14     printf("Got Request : Resta %f/%f y %f/%f \n",argp->num1,argp->num2,argp->num3,argp->num4);
15     printf("Sent Response : %f/%f = %f \n",m,mcm,result);
16
17     return &result;
18 }
19
20 //MULTIPLICACION
21 float *mul_1_svc(inputs *argp, struct svc_req *rqstp){
22     static float result;
23
24     //Todo lo necesario para la multiplicación de fracciones
25     float m=argp->num1*argp->num3; //Calcula el numerador final
26     float d=argp->num2*argp->num4; //Calcula el denominador final
27
28     //Ponemos aquí nuestro código para la multiplicación
29     result = m/d;
30     printf("Got Request : Multiplica %f/%f y %f/%f \n",argp->num1,argp->num2,argp->num3,argp->num4);
31     printf("Sent Response : %f/%f = %f \n",m,d,result);
32     return &result;
33 }
34
35 //DIVISION
36 float *div_1_svc(inputs *argp, struct svc_req *rqstp){
37     static float result;
38
39     //Todo lo necesario para la división de fracciones
40     float m=argp->num1*argp->num4; //Calcula el numerador final
41     float d=argp->num2*argp->num3; //Calcula el denominador final
42
43     //Ponemos aquí nuestro código para la división
44     result = m/d;
45     printf("Got Request : Divide %f/%f y %f/%f \n",argp->num1,argp->num2,argp->num3,argp->num4);
46     printf("Sent Response : %f/%f = %f \n",m,d,result);
47
48     return &result;
49 }
50

```

- calculadora_client.c

Aquí modificó la forma en la que acepta las entradas del cliente desde teclado y género los procedimientos remotos(add, sub, mul, div, etc) invocando resultados para el cliente.

- En la calculadora básica el cliente me queda tal que así:

```
7 #include "calculadora.h"
8 #include <math.h>
9 #include <stdio.h>
10
11 float
12 calculadora_prog_1(char *host, float a, float b, char op, CLIENT *clnt)
13 {
14     float *result_1;
15     inputs add_1_arg;
16     float *result_2;
17     inputs sub_1_arg;
18     float *result_3;
19     inputs mul_1_arg;
20     float *result_4;
21     inputs div_1_arg;
22     float *result_5;
23     inputs res_1_arg;
24     float *result_6;
25     inputs pot_1_arg;
26
27
28 //SUMA
29     if (op=='+'){
30         add_1_arg.num1=a;           //El primer sumando va a ser el primer float que pasamos.
31         add_1_arg.num2=b;           //El segundo sumando sera el segundo float.
32         add_1_arg.operador=op;      //El char que pasemos sera el operador de la suma.
33
34         result_1 = add_1(&add_1_arg, clnt);           //Calcula la suma
35         if(result_1 == (float *) NULL){               //Comprueba que se hace bien la suma, si esta vacio el resultado es que no se ha hecho
36             clnt_perror(clnt, "SYNTAX ERROR");        //Nos da un mensaje de error
37         }
38
39         return *result_1;           //Nos devuelve el resultado de la suma;
40     }
41
42 //RESTA
43     if (op=='-'){
44         sub_1_arg.num1=a;           //El minuendo va a ser el primer float que pasamos.
45         sub_1_arg.num2=b;           //El sustraendo sera el segundo float.
46         sub_1_arg.operador=op;      //El char que pasemos sera el operador de la resta.
47
48         //Viene por defecto al crearse con rpcgen, lo dejo asi.
49         result_2= sub_1(&sub_1_arg, clnt);           //Calcula la resta
50         if(result_2 == (float *) NULL){               //Comprueba que se hace bien la resta, si esta vacio el resultado es que no se ha hecho
51             clnt_perror(clnt, "SYNTAX ERROR");        //Nos da un mensaje de error
52         }
53
54         return *result_2;           //Nos devuelve el resultado de la resta;
55     }
56 }
```

```

//MULTIPLICACION
if (op == '*'){
    mul_1_arg.num1=a;           //El primer factor va a ser el primer float que pasamos.
    mul_1_arg.num2=b;           //El segundo factor sera el segundo float.
    mul_1_arg.operador=op;       //El char que pasemos sera el operador de la multiplicación.

    //Viene por defecto al crearse con rpcgen, lo dejo asi.
    result_3= mul_1(&mul_1_arg, clnt); //Calcula la multiplicacion
    if(result_3 == (float *) NULL){    //Comprueba que se hace bien la multiplicacion, si esta vacio el resultado es que no se ha hecho
        clnt_perror(clnt, "SYNTAX ERROR"); //Nos da un mensaje de error
    }

    return *result_3;           //Nos devuelve el producto de la multiplicacion;
}

//DIVISION
if (op == '/'){
    div_1_arg.num1=a;           //El dividendo va a ser el primer float que pasamos.
    div_1_arg.num2=b;           //El divisor sera el segundo float.
    div_1_arg.operador=op;       //El char que pasemos sera el operador de la division.

    //Tenemos que tener en cuenta que si el divisor es 0 nos tiene que dar error por tanto
    if (b == 0){
        printf("SYNTAX ERROR \n");
        exit(0);
    }else{

        //Viene por defecto al crearse con rpcgen, lo dejo asi.
        result_4= div_1(&div_1_arg, clnt); //Calcula la division
        if(result_4 == (float *) NULL){    //Comprueba que se hace bien la division, si esta vacio el resultado es que no se ha hecho
            clnt_perror(clnt, "SYNTAX ERROR"); //Nos da un mensaje de error
        }

        return *result_4;           //Nos devuelve el cociente de la division;
    }
}

//RESTO
if (op == '%'){
    res_1_arg.num1=a;           //El dividendo va a ser el primer float que pasamos.
    res_1_arg.num2=b;           //El divisor sera el segundo float.
    res_1_arg.operador=op;       //El char que pasemos sera el operador del resto.

    //Tenemos que tener en cuenta que si el divisor es 0 nos tiene que dar error por tanto
    if (b == 0){
        printf("SYNTAX ERROR \n");
        exit(0);
    }else{

        //Viene por defecto al crearse con rpcgen, lo dejo asi.
        result_5= res_1(&res_1_arg, clnt); //Calcula la division
        if(result_5 == (float *) NULL){    //Comprueba que se hace bien la division, si esta vacio el resultado es que no se ha hecho
            clnt_perror(clnt, "SYNTAX ERROR"); //Nos da un mensaje de error
        }

        return *result_5;           //Nos devuelve el resto de la division;
    }
}

//POTENCIA
if (op == '^'){
    pot_1_arg.num1=a;           //La base va a ser el primer float que pasamos.
    pot_1_arg.num2=b;           //El exponente sera el segundo float.
    pot_1_arg.operador=op;       //El char que pasemos sera el operador de la potencia.

    //Viene por defecto al crearse con rpcgen, lo dejo asi.
    result_6 = pot_1(&pot_1_arg, clnt);
    if (result_6 == (float *) NULL) {
        clnt_perror (clnt, "call failed");
    }

    return *result_6;           //Nos devuelve el resultado de la potencia;
}
}

```

```

35 int main (int argc, char *argv[]){
36     //Añadimos las variables que faltan
37     char *host;
38     float a, b;
39     char op;
40     CLIENT *clnt;
41
42     if (argc < 2) {
43         printf ("usage: %s server_host\n", argv[0]);
44         exit (1);
45     }
46
47     //Creamos un pequeño menu
48     printf("Bienvenido estas usando la calculadora de Claudia \n");
49     printf("Recuerda: \n + para la SUMA \n - para la RESTA \n * para la multiplicacion \n / para la division \n porcentaje para el resto \n ^ para la potencia ");
50     printf("Indique el primer numero \n");
51     scanf("%f", &a);
52     printf("Indique el segundo numero \n");
53     scanf("%f", &b);
54     printf("Indique el operador \n");
55     scanf("%s", &op);
56
57     //Viene por defecto al crearse con rpcgen, lo dejo asi.
58     host = argv[1];
59     clnt = clnt_create (host, CALCULADORA_PROG, CALCULADORA_VER, "udp");
60     if (clnt == NULL) {
61         clnt_pcreateerror (host);
62         exit (1);
63     }
64     //Imprimimos el resultado
65     printf("El resultado es: %f \n", calculadora_prog_1(host,a,b,op,clnt));
66
67     //Viene por defecto al crearse con rpcgen, lo dejo asi.
68     clnt_destroy (clnt);
69     exit (0);
70 }

```

- En la calculadora de fracciones el cliente me queda tal que así:

```

7 #include "calculadora.h"
8
9
10 float
11 calculadora_prog_1(char *host, float a, float b, float c, float d, char op, CLIENT *clnt)
12 {
13     float *result_1;
14     inputs add_1_arg;
15     float *result_2;
16     inputs sub_1_arg;
17     float *result_3;
18     inputs mul_1_arg;
19     float *result_4;
20     inputs div_1_arg;
21     float *result_5;
22     inputs pot_1_arg;
23
24 //SUMA
25 if(op=='+'){
26
27     add_1_arg.num1=a;
28     add_1_arg.num2=b;
29     add_1_arg.num3=c;
30     add_1_arg.num4=d;
31     add_1_arg.operador=op;
32
33     result_1 = add_1(&add_1_arg, clnt);
34     if (result_1 == (float *) NULL) {
35         clnt_perror (clnt, "call failed");
36     }
37     return *result_1;
38 }
39
40 //RESTA
41 if(op=='-'){
42
43     sub_1_arg.num1=a;
44     sub_1_arg.num2=b;
45     sub_1_arg.num3=c;
46     sub_1_arg.num4=d;
47     sub_1_arg.operador=op;
48
49     result_2 = sub_1(&sub_1_arg, clnt);
50     if (result_2 == (float *) NULL) {
51         clnt_perror (clnt, "call failed");
52     }
53     return *result_2;
54 }
55

```

```

56 //MULTIPLICACION
57 if(op=='*'){
58
59     mul_1_arg.num1=a;
60     mul_1_arg.num2=b;
61     mul_1_arg.num3=c;
62     mul_1_arg.num4=d;
63     mul_1_arg.operador=op;
64
65     result_3 = mul_1(&mul_1_arg, clnt);
66     if (result_3 == (float *) NULL) {
67         clnt_perror (clnt, "call failed");
68     }
69     return *result_3;
70 }
71
72 //DIVISION
73 if(op=='/'){
74
75     div_1_arg.num1=a;
76     div_1_arg.num2=b;
77     div_1_arg.num3=c;
78     div_1_arg.num4=d;
79     div_1_arg.operador=op;
80
81     result_4 = div_1(&div_1_arg, clnt);
82     if (result_4 == (float *) NULL) {
83         clnt_perror (clnt, "call failed");
84     }
85     return *result_4;
86 }
87
88 }
89
90 int
91 main (int argc, char *argv[])
92 {
93     //Añadimos las variables que faltan
94     char *host;
95     float a, b, c, d;
96     char op;
97     CLIENT *clnt;
98
99     if (argc < 2) {
100         printf ("usage: %s server_host\n", argv[0]);
101         exit (1);
102     }
103
104     //Creamos un pequeño menu
105     printf("Bienvenido estas usando la calculadora de Claudia \n");
106     printf("Recuerda: \n + para la SUMA \n - para la RESTA \n * para la multiplicacion \n / para la division \n");
107     printf("Indique el numerador de la primera fraccion \n");
108     scanf("%f", &a);
109     printf("Indique el denominador de la primera fraccion \n");
110     scanf("%f", &b);
111     printf("Indique el numerador de la segunda fraccion \n");
112     scanf("%f", &c);
113     printf("Indique el denominador de la segunda fraccion \n");
114     scanf("%f", &d);
115     printf("Indique el operador \n");
116     scanf("%s", &op);
117
118     //Viene por defecto al crearse con rpcgen, lo dejo asi.
119     host = argv[1];
120     clnt = clnt_create (host, CALCULADORA_PROG, CALCULADORA_VER, "udp");
121     if (clnt == NULL) {
122         clnt_pcreateerror (host);
123         exit (1);
124     }
125
126     //Imprimimos el resultado
127     printf("El resultado es: %f \n", calculadora_prog_1(host,a,b,c,d,op,clnt));
128
129     //Viene por defecto al crearse con rpcgen, lo dejo asi.
130     clnt_destroy (clnt);
131     exit (0);
132 }
133
134

```

- Funcionamiento

Para comprobar cómo funcionan hacemos el siguiente comando en ambas lo mismo:

```
claudiasalado@claudiasalado-VirtualBox:~/Desktop/DSD/P2_SUNRPC/Calculadora_Fracciones$ make -f Makefile.calculadora
cc -g -c -o calculadora_xdr.o calculadora_xdr.c
cc -g -c -o calculadora_client.o calculadora_client.o calculadora_xdr.o -lnsl
cc -g -c -o calculadora_svc.o calculadora_svc.c
cc -g -c -o calculadora_server.o calculadora_server.c
```

Se nos van a crear dos ejecutables calculadora_client y calculadora_server ejecutamos ambos cada uno en una terminal.

```
claudiasalado@claudiasalado-VirtualBox:~/Desktop/DSD/P2_SUNRPC/Calculadora_Fracciones$ sudo ./calculadora_client localhost
[sudo] password for claudiasalado:
```

```
claudiasalado@claudiasalado-VirtualBox:~/Desktop/DSD/P2_SUNRPC/Calculadora_Fracciones$ sudo ./calculadora_server
[sudo] password for claudiasalado:
```

Ahora pruebo las calculadoras:

• Calculadora Básica

Voy a probar con una potencia.

En el cliente nos aparece el menú y ponemos los numero en este caso primero la base después el exponente y el operador y nos devuelve el resultado.

```
Bienvenido estas usando la calculadora de Claudia
Recuerda:
+ para la SUMA
- para la RESTA
* para la multiplicacion
/ para la division
r para el resto
^ para la potencia
Indique el primer numero
5
Indique el segundo numero
2
Indique el operador
^
El resultado es: 25.000000
```

En el server nos muestra la operación que pedimos que realice y el resultado que manda al cliente.

```
Got Request : Eleva 5.000000 a 2.000000
Sent Response : 25.000000
```

● Calculadora de Fracciones

Voy a probar una suma.

En el cliente nos aparece el menú y ponemos las fracciones y el operador y nos devuelve el resultado.

```
Bienvenido estas usando la calculadora de Claudia
Recuerda:
+ para la SUMA
- para la RESTA
* para la multiplicacion
/ para la division
Indique el numerador de la primera fraccion
1
Indique el denominador de la primera fraccion
3
Indique el numerador de la segunda fraccion
1
Indique el denominador de la segunda fraccion
2
Indique el operador
+
El resultado es: 0.833333
```

En el server nos muestra la operación que pedimos que realice y el resultado que manda al cliente.

```
Got Request : Suma 1.000000/3.000000 y 1.000000/2.000000
Sent Response : 2.500000/3.000000 = 0.833333
```

Nota:

Al enviar los archivos fuera del zip de cada calculadora los he tenido que renombrar para identificandolos con calculadora_basica y calculadora_fracciones.

Si se cogen esos seguramente no funcionen ya que tienen nombres distintos a como vienen en el makefile.

Otra cosa que me he dado cuenta al final es que en el básico cuando puse los .txt y los .c fuera del zip hay una cosa del menú que cambia con respecto a los del zip. pone “porcentaje para el resto” mientras que en los del zip pone “r para el resto”, hay que poner r no el símbolo del porcentaje.