Manuel Contreras Orge

MEMORIA P2-1: CALCULADORA SUN RPC

Introducción

En esta práctica se ha hecho una calculadora que tiene 9 operaciones distintas (suma, resta, multiplicación, división, módulo, elevar, raíz cuadrada, valor absoluto y logaritmo) con números enteros y con decimales, usando la versión SunRPC de RPC (llamada a procedimiento remoto).

Todas estas son operaciones sencillas, no se pueden usar ni vectores ni matrices en la versión que he programado de esta calculadora.

Ficheros solución

calculadora.x:

Archivo creado en el lenguaje de Sun RPC en el que se crean las cabeceras de olas funciones que se describen más abajo y que son las que hace la propia calculadora.

El fichero crea un programa, llamado CALCULADORA, con número de programa x200000001 (que es el primero disponible) y que tiene una única versión llamada CALCULADORA_BASICA, cuyo número identificador es el 1. Además, las posibles operaciones están también ordenadas del 1 al 9.

```
program CALCULADORA{
    version CALCULADORA_BASICA{
        double SUMAR (double n1, double n2) = 1;
        double MULTIPLICAR (double n1, double n2) = 2;
        double RESTAR (double n1, double n2) = 3;
        double DIVIDIR (double n1, double n2) = 4;
        double MODULO (int n1, int n2) = 5;
        double ELEVAR (double n1, double n2) = 6;
        double RAIZ_CUADRADA (int n1) = 7;
        double VALOR_ABSOLUTO (int n1) = 8;
        double LOGARITMO (int n2) = 9;
    } = 1;
} =0x20000001;
```

Código del fichero calculadora.x.

calculadora_server.c:

En este archivo se definen lo que hacen todas las operaciones que puede realizar la calculadora, cuyas cabeceras están definidas en el archivo *calculadora.x*, explicado anteriormente. En la cabecera de este archivo se hace una llamada al archivo *calculadora.h*, que contiene las cabeceras de las funciones y a los archivos de cabecera *math.h* y *stadio.h*, que son necesarios para usar algunas de las operaciones, como *pow*. El esqueleto de todas las operaciones es igual:

- Todas estas operaciones devuelven un puntero de un número decimal (de tipo *double*).
- Las funciones reciben como datos dos variables de tipo decimal también (excepto módulo, que opera con enteros) todas las variables excepto las tres últimas operaciones (raíz cuadrada, valor absoluto y logaritmo), que solo usan un dato, y de tipo entero, para realizar la operación.
- Dentro de cada función se crea una variable decimal puntero *resultado*, que es la que se devuelve.
- Después se escribe el valor resultante de la operación que toca en dicha variable *resultado*.
- Por último, se devuelve la variable *resultado*.

```c
double * sumar_1_svc(double n1, double n2,  struct svc_req *rqstp) {
    static double resultado_sumar;

    resultado_sumar = (n1 + n2);

    return &resultado_sumar;
}

double * restar_1_svc(double n1, double n2,  struct svc_req *rqstp) {
    static double resultado_restar;

    resultado_restar = (n1 - n2);

    return &resultado_restar;
}

double * multiplicar_1_svc(double n1, double n2,  struct svc_req *rqstp) {
    static double resultado_multiplicar;

    resultado_multiplicar = (n1 * n2);

    return &resultado_multiplicar;
}

double * dividir_1_svc(double n1, double n2,  struct svc_req *rqstp) {
    static double resultado_dividir;

    resultado_dividir = (n1 / n2);

    return &resultado_dividir;
}

double * modulo_1_svc(int n1, int n2,  struct svc_req *rqstp) {
    static double resultado_modulo;

    resultado_modulo = (n1 % n2);

    return &resultado_modulo;
}

double * elevar_1_svc(double n1, double n2,  struct svc_req *rqstp) {
    static double resultado_elevar;

    resultado_elevar = pow(n1, n2);

    return &resultado_elevar;
}
```

Código de las operaciones con dos datos de entrada.

```c
double * raiz_cuadrada_1_svc(int n1,  struct svc_req *rqstp) {
    static double resultado_raiz_cuadrada;

    resultado_raiz_cuadrada = sqrt(n1);

    return &resultado_raiz_cuadrada;
}

double * logaritmo_1_svc(int n1,  struct svc_req *rqstp) {
    static double resultado_logaritmo;

    resultado_logaritmo = log(n1);

    return &resultado_logaritmo;
}

double * valor_absoluto_1_svc(int n1,  struct svc_req *rqstp) {
    static double resultado_valor_absoluto;

    resultado_valor_absoluto = fabs(n1);

    return &resultado_valor_absoluto;
}
```

Código de las operaciones con dos datos de entrada.

calculadora_client.c:

En este archivo se leen las variables que van a llevar al resultado final, dependiendo del operador que se introduzca, todos estos datos se leerán por pantalla en la llamada a este archivo.

Una vez leídas estas variables se crea el cliente en el que estamos, y dependiendo del símbolo de la operación, se calcula el resultado final, y se seca por pantalla el resultado en el caso de que el resultado no sea nulo.

```c
#ifndef DEBUG
    clnt = clnt_create (server, CALCULADORA, CALCULADORA_BASICA, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (server);
        exit (1);
    }
#endif  /* DEBUG */

    /* Operación resultante según el operador seleccionado en la llamada */
    if (op == '+') {
        resultado_operacion = sumar_1(n1, n2, clnt);
    }
    else if (op == '-') {
        resultado_operacion = restar_1(n1, n2, clnt);
    }
    else if (op == '*' || op == 'x') {
        resultado_operacion = multiplicar_1(n1, n2, clnt);
    }
    else if (op == '/') {
        resultado_operacion = dividir_1(n1, n2, clnt);
    }
    else if (op == '%') {
        resultado_operacion = modulo_1(n1, n2, clnt);
    }
    else if (op == '^') {
        resultado_operacion = elevar_1(n1, n2, clnt);
    }
    else if (op == 'r') {
        resultado_operacion = raiz_cuadrada_1(n1, clnt);
    }
    else if (op == 'l') {
        resultado_operacion = logaritmo_1(n1, clnt);
    }
    else if (op == 'a') {
        resultado_operacion = valor_absoluto_1(n1, clnt);
    }
    else {
        fprintf(stderr, "ERROR, se ha introducido operador no existente (elegir entre +, -, *-x, /, %, ^, r, a, l)\n");
        exit(1);
    }

    if (resultado_operacion == (double *) NULL) {
        clnt_perror (clnt, "call failed");
        exit(1);
    }

    /* Imprime el resultado */
    printf("El resultado de la operacion solicitada es: %f\n", *resultado_operacion);

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

Código de calculadora_client.c.

Modo de uso

Para usar este programa primero tenemos que compilar el archivo *calculadora.x* con la orden *rpcgen -NCa calculadora.x*, con esto se generarán los archivos con extensión .c (*calculadora_client, calculadora_clnt, calculadora_server, calculadora_svc y calculadora_xdr*), el archivo *calculadora.h* y el makefile (*Makefile.calculadora*).

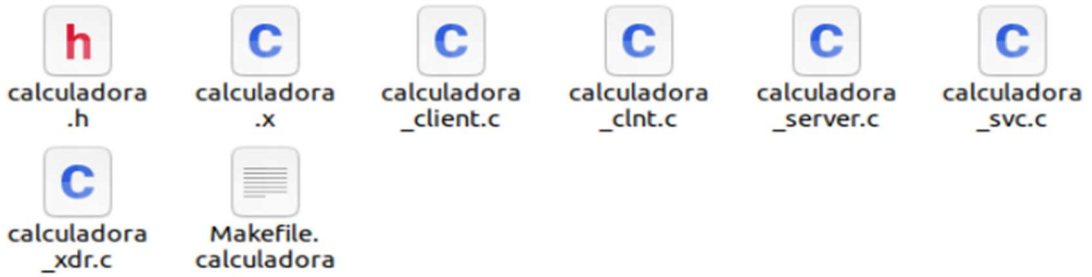


Imagen de los archivos generados con esta orden.

A continuación, cambiamos los archivos *calculadora_client.c* y *calculadora_server.c* por los archivos generados por mí mismo, que se han explicado y mostrado con anterioridad; una vez hecho esto generamos los archivos ejecutables y los archivos .o del programa con el makefile, usando la orden make -f Makefile.calculadora, lo que dará un error, como el que se muestra a continuación.

```
manuel@manuel:~/Escritorio/DSD/a$ make -f Makefile.calculadora
cc -g    -c -o calculadora_clnt.o calculadora_clnt.c
cc -g    -c -o calculadora_client.o calculadora_client.c
calculadora_client.c: In function 'calculadora_1':
calculadora_client.c:52:96: warning: unknown conversion type character ',' in format [-Wformat=]
   52 | troducido operador no existente (elegir entre +, -, *-x, /, %, ^, r, a, l)\n");
      |                                                                                ^

cc -g    -c -o calculadora_xdr.o calculadora_xdr.c
cc -g     -o calculadora_client  calculadora_clnt.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
cc -g     -o calculadora_server  calculadora_svc.o calculadora_server.o calculadora_xdr.o -lnsl
/usr/bin/ld: calculadora_server.o: en la función `elevar_1_svc':
/home/manuel/Escritorio/DSD/a/calculadora_server.c:53: referencia a `pow' sin definir
/usr/bin/ld: calculadora_server.o: en la función `raiz_cuadrada_1_svc':
/home/manuel/Escritorio/DSD/a/calculadora_server.c:61: referencia a `sqrt' sin definir
/usr/bin/ld: calculadora_server.o: en la función `logaritmo_1_svc':
/home/manuel/Escritorio/DSD/a/calculadora_server.c:69: referencia a `log' sin definir
collect2: error: ld returned 1 exit status
make: *** [Makefile.calculadora:42: calculadora_server] Error 1
```

Error resultante de ejecutar el makefile.

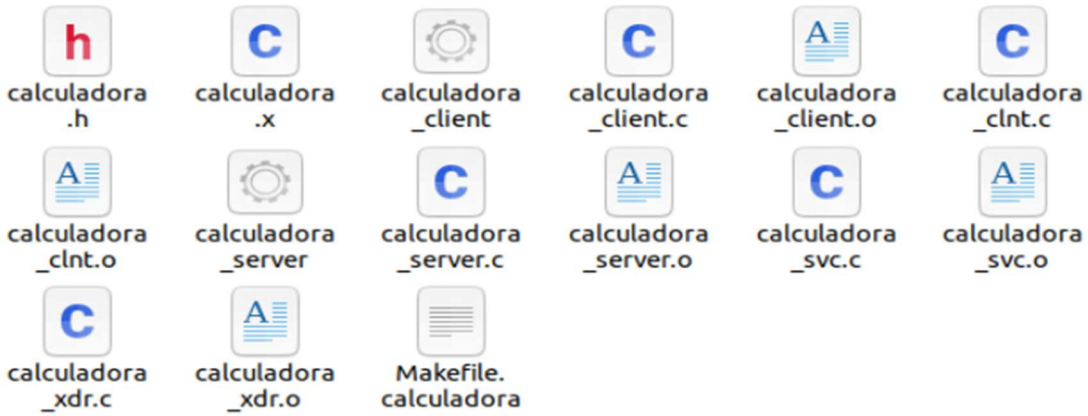
Para solucionar este error debemos de modificar en dicho makefile, cambiando la biblioteca que se usa para ejecutar por *-lm* (orden para llamar a la biblioteca m), como se puede ver en la imagen de abajo.

```
$(CLIENT) : $(OBJECTS_CLNT)
        $(LINK.c) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS)

$(SERVER) : $(OBJECTS_SVC)
        $(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LM)
```

Creación del server, anteriormente, llamábamos a la misma biblioteca que el cliente.

Una vez hecho este cambio, se crea el ejecutable del server y ya tenemos todos los archivos necesarios para la ejecución del programa.



Archivos necesarios para el programa.

Por último, vamos a probar a ejecutar el programa, para comprobar que todo haya salido bien y que no tengamos ningún fallo a la hora de haber hecho las funciones o sus llamadas. Para esto, primero ejecutaremos el servidor, que en este caso lo vamos a hacer en segundo plano con &, de la siguiente forma: *./calculadora_server &*; después, llamaremos al cliente, de la siguiente manera: *./calculadora_client localhost n1 op n2*; siendo n1 el primer número, op el símbolo de la operación que se va a usar , n2 el segundo número y localhost la variable para usar esta máquina también como server; si todo está bien, debería aparecer una cosa parecida a la imagen que se observa justo abajo.

```
manuel@manuel:~/Escritorio/DSD/practica-2-1-calculadora$ ./calculadora_server &
[2] 5955
manuel@manuel:~/Escritorio/DSD/practica-2-1-calculadora$ ./calculadora_client localhost 3 ^ 2
El resultado de la operacion solicitada es: 9.000000
```

Ejemplo de ejecución del programa con la operación de elevar.