2024

Llamada remota a procedimiento (RPC)

PRÁCTICA 2.1

CARMEN CHUNYIN FERNÁNDEZ NÚÑEZ

1. INTRODUCCIÓN

Esta práctica consiste en el desarrollo de un programa distribuido, para ser más exactos una calculadora, utilizando RPC. Este programa realizará desde varias operaciones básicas como sumas, restas, multiplicaciones y divisiones hasta operaciones con estructuras más complejas como son las matrices.

2. EXPLICACIÓN DE LA SOLUCIÓN

2.1 CALCULADORA.X

Definición (calculadora.x): En este archivo se define una interfaz, datos y rutinas a las que se puede acceder remotamente, escrito en el lenguaje SunRPC. Las variables y estructuras definidas son:

- **typedef double m<>**, es un vector de números double finito, sirve como atributo en la estructura de datos matrix. Se ha usado <> para indicar que es un array sin un tamaño definido, que posteriormente se define en el cliente reservando memoria en función del tamaño de la matriz
- **struct matrix,** es una estructura con tres campos.
    - **int fil,** el número de filas que contiene la matriz
    - **int col,** el número de columnas que contiene la matriz
    - **m m**, donde se guardará los valores de la matriz.
- **typedef int v<>**, es un vector de números double finito, se usa en las operaciones con vectores. Se ha usado <> para indicar que es un array sin un tamaño definido, que posteriormente se define en el cliente reservando memoria.

```
typedef double m<>;

struct matrix{
    int fil;
    int col;
    m m;
};

typedef double v<>;

program CALCULADORA {
    version COMPLETA {
        double SUMA(double,double) = 1;
        double RESTA(double,double) = 2;
        double MULTIPLICA(double,double) = 3;
        double DIVIDE(double,double) = 4;
        int MODULO (int, int) = 5;
        int POTENCIA (int, int) =6;
        double RAIZ (double, double) = 7;
        int FACTORIAL (int) = 8;

        double LOGARITMO(int, int) = 9;
        double LOG_N (int) = 10;
        double LOG10 (int) = 11;

        double SENO (double) = 12;
        double COSENO (double) = 13;
        double TANGENTE (double) = 14;
        double SECANTE (double) = 15;
        double COSECANTE (double) = 16;
        double COTANGENTE (double) = 17;

        v SUMAVECTORIAL(v, v) = 18;
        v RESTAVECTORIAL(v, v) = 19;
        v MULTIPLICAPORESCALAR(v, double) = 20;
        double PRODUCTOESCALAR(v, v) = 21;
        v PRODUCTOVECTORIAL(v, v) = 22;

        matrix SUMAMATRICIAL(matrix, matrix) = 23;
        matrix RESTAMATRICIAL(matrix, matrix) = 24;
        matrix MULTMATRICIAL(matrix, matrix) = 25;
    }=1;
} = 0x2000001;
```

El tipo de dato m y v, al ser vectores, SunRPC los traduce a C como una estructura con dos campos, el propio vector y su tamaño.

Además, se ha creado un programa CALCULADORA, con una única versión con identificador 1. Esta versión define todas las funciones que puede realizar el servidor calculadora, y están enumeradas del 1 al 25.

2.2 MAKEFILE

Por comodidad a la hora de trabajar, he modificado el nombre del Makefile de "Makefile.calculadora" a "Makefile". Además, cambiaremos las CFLAGS y LDLIBS para que tengan en cuenta las biblioteca RPC y la de cmath

```
CFLAGS += -g -I/usr/include/tirpc
LDLIBS += -ltirpc -lnsl -lm  # Añadimos tambien la biblioteca cmath
```

2.3 CALCULADORA_SERVER.C

En el archivo del servidor "***calculadora_server.c***" están definidas las funciones incluidas en "***calculadora.x***", las cuales se encontrarán vacías en un principio y será nuestro trabajo incluir el código para que cada función proporcione el resultado esperado.

```
#include "calculadora.h"
#include <math.h>
#include <string.h>
#include <assert.h>
#include <ctype.h>

Comment Code | Improve Code
double *
suma_1_svc(double arg1, double arg2,  struct svc_req *rqstp)
{
    static double  result;
    result = arg1 + arg2;
    return &result;
}

Comment Code | Improve Code
double *
resta_1_svc(double arg1, double arg2,  struct svc_req *rqstp)
{
    static double  result;
    result = arg1 - arg2;
    return &result;
}

Comment Code | Improve Code
double *
multiplica_1_svc(double arg1, double arg2,  struct svc_req *rqstp)
{
    static double  result;
    result = arg1 * arg2;
    return &result;
}
```

2.4 CALCULADORA_CLIENT.C

El cliente gestiona todo lo relacionado con lo que introduce el usuario. De primeras, el main solo llama a la función calculadora_1(host) la cual gestiona todas las operaciones de la calculadora.

```
int
main (int argc, char *argv[])
{
    char *host;

    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    calculadora_1 (host);
exit (0);
}
```

Se han definido en el cliente, varias funciones que servirán para manejar las operaciones con matrices y vectores:

- **void reservarMatrix(matrix **m, int fil, int col):** Reserva memoria para una matriz m de fil número de filas y col número de columnas.
- **void liberarMatrix(matrix **m):** Libera la memoria reservada para una matriz m.
- **void imprimirMatrix(matrix *m):** Imprime la matriz m
- **void rellenarMatrix(matrix *m):** Rellena la matriz m con los valores introducidos por teclado
- **void reservarVector(v **vector, int tam):** Reserva memoria para vector de tamaño tam
- **void liberarVector(v ** vector):** Libera la memoria reservada para el vector
- **void imprimirVector(v * vector, int tam):** Imprime el vector
- **void rellenarVector(v * vector, int tam):** Rellena el vector con los valores introducidos por teclado

```
void reservarMatrix(matrix **m, int fil, int col){
    assert((*m)==NULL);

    *m=calloc(1, sizeof(matrix));

    (*m)->fil=fil;
    (*m)->col=col;

    (*m)->m.m_val=calloc(fil*col, sizeof(double));
    (*m)->m.m_len=fil*col;
}

Comment Code | Improve Code
void liberarMatrix(matrix **m){
    assert((*m)!=NULL);

    free((*m)->m.m_val);
    free(*m);

    *m=NULL;
}
```

Una vez entrados en la función **calculadora_1(host)** se declaran una variedad de variables que se utilizarán en las diferentes operaciones. Primero se crea el cliente con **clnt_create()**, posteriormente aparecen diferentes menús en los que elegiremos que operación queremos realizar, mediante switch se llama a la operación seleccionada, después de haber introducido los datos correspondientes. No se saldrá del programa a menos que se lo indiquemos, es decir, este seguirá reproduciendo el menú actual en bucle hasta que se selecciona la opción de salir.

```
void
calculadora_1(char *host)
{
    CLIENT *clnt;
    char opcion;
    char operacion;

    int tam, fil, col;

    double *result_d;
    double arg1_d, arg2_d;
    int *result_i;
    int arg1_i, arg2_i;
    v *result_v=NULL;
    v *v1=NULL, *v2=NULL;
    matrix *result_m = NULL;
    matrix *m1=NULL, *m2=NULL;

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

3. COMPILACIÓN

Como se ha indicado previamente, primero he generado el archivo **calculadora.x**, cuyo contenido ha sido explicado. Con el archivo ya terminado, se ha ejecutado el siguiente comando:

rpcgen -NCa calculadora.x

Este comando genera los siguientes archivos:

- **calculadora_client.c**, el programa que ejecutará el cliente y cuyo contenido se ha explicado. Se ejecuta en un terminal distinto al terminal del servidor.
- **calculadora_clnt.c**, stub del cliente.
- **calculadora_server.c**, el programa que ejecutará el servidor y cuyo contenido se ha explicado. Se ejecuta en un terminal distinto del terminal del cliente.
- **calculadora_svc.c**, stub del servidor que escucha al stub del cliente.
- **calculadora_xdr.c**, rutinas xdr para los tipos de datos definidos. Convierten datos al formato XDR y viceversa
- **calculadora.h**, cabecera con definiciones comunes al cliente y servidor, en el que se pueden ver las variables y funciones definidas.

La opción **N** sirve para generar el código pasando los argumentos por valor al estilo C, y poder pasar más de un parámetro en las funciones. Es la opción a usar cuando se cambia algo en **calculadora.x** y que no se sobrescriban los archivos de cliente y servidor, el resto los genera RPC automáticamente y no hay que tocarlos. La opción **C** indica que genera el código en ANSI-C y **a** es para indicar que se generen plantillas de cliente y servidor (por tanto esto solo se hace la primera vez)

El funcionamiento consiste en abrir dos terminales, en una se ejecutará el servidor y en la otra el cliente (con localhost). Una vez ejecutado el servidor, se mantiene activo indefinidamente escuchando peticiones del cliente.

Para ejecutarlos es necesario compilarlos, rpcgen genera un makefile el cual genera y compila todos los ejecutables.

4. FUNCIONAMIENTO

En este apartado, voy a explicar el funcionamiento de mi calculadora, proporcionando ejemplos e imágenes. El uso de mi programa es bastante intuitivo, igualmente realizaré una guía detallada.

Una vez ejecutado el servidor en una terminal **./calculadora_server** en otra terminal se ejecuta el cliente **./calculadora_client localhost**

```
alissea@alissea-VirtualBox:~/DSD/P2_1/calculadora v2.0$ ./calculadora_client localhost
```

Una vez ejecutado, aparecerá un menú principal con 6 opciones para elegir:

```
ELIJA UNA OPCION DE LA CALCULADORA:
        1. Operaciones Basicas
        2. Operaciones Avanzadas
        3. Operaciones Trigonometricas
        4. Operaciones Vectoriales
        5. Operaciones Matriciales
        6. Salir del Programa
```

A continuación, deberá introducir el número de la opción que desea seleccionar, en este ejemplo seleccionaremos el 4. Cada una de estas opciones te llevará a un submenú de operaciones donde habrá que volver a seleccionar el número de la operación deseada.

```
ELIJA UNA OPERACION:
        1. Suma Vectorial
        2. Resta Vectorial
        3. Multiplicacion por Escalar
        4. Producto Escalar
        5. Producto Vectorial
        6. Volver al menu principal
```

Nosotros vamos a realizar una SUMA, por lo que elegiremos el número 1

Una vez seleccionada la operación que queremos que realice nuestra calculadora, el programa nos va a pedir que introduzcamos diversos datos, dependiendo de que operación vamos a realizar. Una vez introducidos los datos, genera el resultado de la operación (el cual imprime por pantalla)

```
Tamaño del vector: 3
Primer vector:
1
2
3
Segunda vector:
5
2
3

Resultado: (1.000000, 2.000000, 3.000000) + (5.000000, 2.000000, 3.000000) = (6.000000, 4.000000, 6.000000)
```

El programa vuelve al submenú de operaciones en el cual nos encontrábamos por defecto. Si deseásemos elegir otro tipo de operación solo debemos elegir la opción de “Volver al menú principal”. Y si quisiésemos salir del programa, elegir la opción “Salir del Programa” en el menú principal.

```
ELIJA UNA OPERACION:
        1. Suma Vectorial
        2. Resta Vectorial
        3. Multiplicacion por Escalar
        4. Producto Escalar
        5. Producto Vectorial
        6. Volver al menu principal
6

Volviendo al menu principal...

ELIJA UNA OPCION DE LA CALCULADORA:
        1. Operaciones Basicas
        2. Operaciones Avanzadas
        3. Operaciones Trigonometricas
        4. Operaciones Vectoriales
        5. Operaciones Matriciales
        6. Salir del Programa
6

Saliendo del programa...
alissea@alissea-VirtualBox:~/DSD/P2_1/calculadora v2.0$
```

A continuación, adjunto una imagen de todas las opciones disponibles

```
OPERACIONES:
    1.BÁSICAS:
        1.1.Suma
        1.2.Resta
        1.3.Multiplicacion
        1.4.Division

    2.AVANZADAS:
        2.1.Logaritmo base X
        2.2.Logaritmo neperiano
        2.3.Logaritmo base 10
        2.4.Módulo
        2.5.Potencia
        2.6.Raíz
        2.7.Factorial
```

```
    3.TRIGONOMETRÍA:
        3.1.Seno
        3.2.Coseno
        3.3.Tangente
        3.4.Secante
        3.5.Cosecante
        3.6.Cotangente

    4.VECTORES:
        4.1.Suma
        4.2.Resta
        4.3.Multiplicacion por Escalar
        4.4.Producto Escalar
        4.5.Producto Vectorial

    5.MATRICES:
        5.1.Suma
        5.2.Resta
        5.3.Multiplicacion
```

5. ESPECIFICACIONES DEL PROGRAMA

En este último apartado voy a explicar algunas funciones que he visto necesarias de aclarar ciertos aspectos.

5.1 FACTORIAL

La mayoría de las operaciones básicas, avanzadas y trigonométricas se realizan utilizando operadores simples o funciones de la librería cmath. La operación factorial puede alcanzar valores extremadamente grandes con números pequeños, así que si en algún momento el valor por el cual se va a multiplcar es mayor que INT_MAX/result (lo cual llevaría a overflow) se devuelve -1.

```
int *
factorial_1_svc(int arg1,  struct svc_req *rqstp)
{
    static int  result = 1;
    for(int i=1; i<=arg1 && result!=-1; i++){
        if(i>(INT_MAX/result)){     //En caso de que haya overflow se pone a -1 y se sale
            result=-1;
        }
        else
            result*=i;
    }
    return &result;
}
```

5.2 FUNCION XDR_FREE()

Todas las operaciones que conllevan vectores y matrices empiezan haciendo **xdr_free(result)**, la cual es una función de la librería de SunRPC. Se utiliza para liberar la memoria reservada por XDR. Al igual que al finalizar cualquier operación, se realizará xdr_free(result) a la variable que albergaba el resultado.

Cuando se utiliza XDR para codificar o decodificar, la propia librería reserva memoria para almacenar los datos codificados/decodificados. Esta memoria se reserva dinámicamente usando “malloc” o “calloc”.

Se hace uso de esta función para asegurarnos de que la variable en la que vamos a guardar la operación está liberada, para después proceder a reservar la memoria y por último la operación

```
v *
sumavectorial_1_svc(v arg1, v arg2,  struct svc_req *rqstp)
{
    static v result;
    xdr_free((xdrproc_t)xdr_double, result.v_val);

    result.v_len=arg1.v_len;
    result.v_val=malloc(arg2.v_len*sizeof(int));

    for(int i=0; i<arg1.v_len; i++)
        result.v_val[i] = arg1.v_val[i] + arg2.v_val[i];

    return &result;
}
```

```
if (result_d == (double *) NULL) {
    clnt_perror (clnt, "call failed");
}
else{
    printf("Resultado: %f + %f = %f\n\n",arg1_d,arg2_d,*result_d);
    xdr_free((xdrproc_t) xdr_double, result_d);
}
```

5.3 FUNCIONES EXTRAS

Tanto los vectores o las matrices usan funciones extras para reservar, liberar, imprimir y rellenar. Voy a explicar como funcionan en las matrices al ser una estructura más compleja.

5.3.1 ReservarMemoria

La función de reservar memoria verifica primero que la matriz sea NULL, y después se reserva memoria para la estructura. A continuación, se asigna el número de filas y columnas de la matriz, y se reserva memoria dinámicamente para cada valor de la matriz y asigna m_len

```c
void reservarMatrix(matrix **m, int fil, int col){
    assert((*m)==NULL);

    *m=calloc(1, sizeof(matrix));

    (*m)->fil=fil;
    (*m)->col=col;

    (*m)->m.m_val=calloc(fil*col, sizeof(double));
    (*m)->m.m_len=fil*col;
}
```

5.3.2 Liberar Memoria

La función de liberar memoria verifica que la matriz sea no nula y después libera la memoria de cada valor de la matriz, para después liberar la memoria ocupada por la estructura.

```c
void liberarMatrix(matrix **m){
    assert((*m)!=NULL);

    free((*m)->m.m_val);
    free(*m);

    *m=NULL;
}
```

5.3.3 Imprimir y Rellenar

Las funciones de imprimir y rellenar, van pasando por todas las filas y columnas de la matriz e imprimen/guardan el valor guardado en esta, quedando la estructura de la siguiente manera

```c
void imprimirMatriz(matrix *m){
        for (int i = 0; i < m->fil; i++)
        {
            for (int j = 0; j < m->col; j++)
            {
                printf("%lf ", m->m.m_val[i*m->col+j]);
            }

            printf("\n");

        }
}
```

```c
Comment Code | Improve Code
void rellenarMatriz(matrix *m1){
    for(int i=0; i<m1->fil; i++){
        printf("Fila %d valores: ", i);
        for (int j = 0; j < m1->col; j++)
        {
            scanf("%lf", &(m1->m.m_val[i*m1->col+j]));
        }
    }
}
```

```
Primera matriz:
1.000000 2.000000 3.000000
4.000000 3.000000 2.000000
1.000000 2.000000 1.000000


Segunda matriz:
3.000000 2.000000 4.000000
1.000000 2.000000 3.000000
4.000000 3.000000 2.000000
```

5.4 ESTRUCTURA DE LAS OPERACIONES

Como explicado anteriormente, primero se definen todas las variables que vamos a necesitar para todas las operaciones. Después se crea el cliente y entramos en el bucle do while() El cual muestra el menú correspondiente mientras no se indique lo contrario.

```
case '4': // OPERACIONES VECTORIALES
    do{
        printf("ELIJA UNA OPERACION:\n");
        printf("\t1. Suma Vectorial\n");
        printf("\t2. Resta Vectorial\n");
        printf("\t3. Multiplicacion por Escalar\n");
        printf("\t4. Producto Escalar\n");
        printf("\t5. Producto Vectorial\n");
        printf("\t6. Volver al menu principal\n");
        scanf("%c", &operacion);
        printf("\n");
```

Una vez elegida la operación, el programa pide los diferentes datos necesarios. En el caso de las matrices y de los vectores reserva la memoria de los vectores/matrices que se pasan como argumentos; al final la libera.

Una vez obtenidos todos los datos necesarios, llama a la operación correspondiente, comprueba que se ha realizado con éxito e imprime el resultado.

Por último libera la memoria de todas las variables utilizadas

```
switch(operacion)
{
    case '1':
        printf("Tamaño del vector: ");
        scanf("%d", &tam);

        reservarVector(&v1, tam);
        reservarVector(&v2, tam);

        printf("Primer vector: \n");
        rellenarVector(v1, tam);

        printf("Segundo vector: \n");
        rellenarVector(v2, tam);
        printf("\n");

        result_v = sumavectorial_1(*v1, *v2, clnt);
        if (result_v == (v *) NULL) {
            clnt_perror (clnt, "call failed");
        }
        else{
            printf("Resultado: ");
            imprimirVector(v1, tam);
            printf(" + ");
            imprimirVector(v2, tam);
            printf(" = ");
            imprimirVector(result_v, tam);

            printf("\n\n");
            xdr_free((xdrproc_t) xdr_double, result_v->v_val);
        }

        liberarVector(&v1);
        liberarVector(&v2);
    break;
```