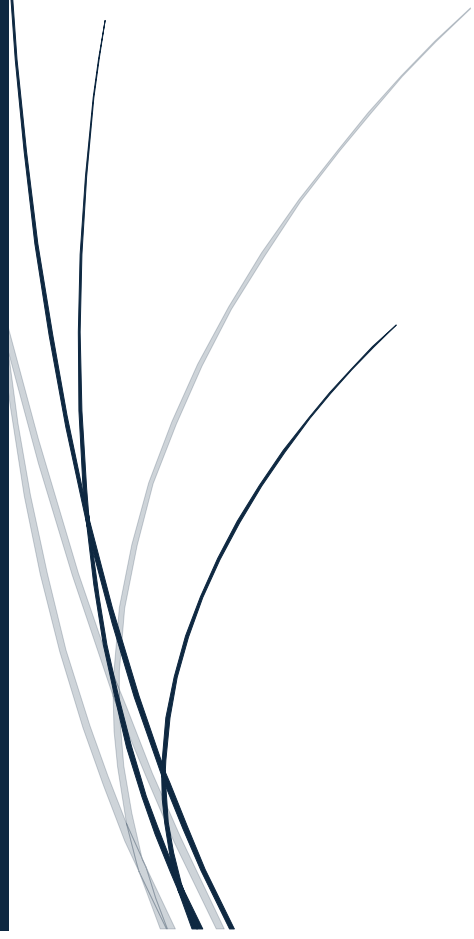
13-3-2024

Calculadora con RPC

Práctica 1



Julián Blanco González
UNIVERSIDAD DE EXTREMADURA

Contenido

Objetivo de la práctica

El objetivo de la práctica es realizar un menú que tenga funciones de calculadora, pero hay que seguir el protocolo rpcgen. El menú tiene estas 6 opciones:

- Sumar (1)
- Restar (2)
- Multiplicar (3)
- Dividir (4)
- Elevar al cuadrado (5)
- Raíz cuadrada (6)

Para ello, los métodos de 1-4 reciben dos double, y devuelven un double

Elevar al cuadrado recibe un double y devuelve un double

Raíz cuadrada recibe un entero y devuelve un double.

Los métodos para hacer los cálculos estarán en el código del servidor.

En el código del cliente, se llamará primero a una función adicional que mostrará un mensaje con las opciones disponibles del menú.

Luego, se ejecutará un bucle, el cual, solo puede salirse cuando el usuario pulse Ctrl + D (EOF).

Si el usuario introduce una opción no valida (opcion != (1-6)), se volverá a pedir un número correcto.

¿Qué es RPC GEN?

RPCGEN es un comando de RPC que al aplicarlo a un archivo con extensión.x, genera los archivos automáticamente tanto del lado del cliente como la del servidor, y la cabecera.

Estos ficheros automáticos dependen del contenido del fichero.x.

Generar archivos a partir de un .x

Este es el archivo con extensión .x que he creado para aplicarle rpcgen:

```
struct datos {
    double a;
    double b;
};
program NPROG {
    version NVERS {
        double suma(datos d) = 1;
        double resta(datos d) = 2;
        double mul(datos d) = 3;
        double div(datos d) = 4;
        double potencia(double d)=5;
        double raizCuadrada(int d)=6;
        string menu(void)=7;
    }=1;
}=0x20000008;
```

1. **Estructura de Datos**:
    - La estructura datos está definida con dos campos de tipo double: a y b. Estos campos representan los números que se utilizarán en las operaciones matemáticas de suma, resta, multiplicación y división.
2. **Definición del Programa RPC**:
    - El programa RPC se llama NPROG.
    - Dentro de NPROG, hay una única versión llamada NVERS.
    - Esta versión contiene varias funciones RPC:
        - ***Double suma(datos d):*** Realiza la suma de los valores a y b de la estructura datos y devuelve el resultado en double.
        - ***Double resta(datos d):*** Calcula la resta entre los valores a y b de la estructura datos y devuelve el resultado en double.
        - ***Double mul(datos d):*** Realiza la multiplicación de a y b de la estructura datos y devuelve el resultado en double.
        - ***Double div(datos d):*** Calcula la división entre a y b de la estructura datos y devuelve el resultado en double.
        - ***Double potencia(double d):*** Calcula la potencia de un número dado d y devuelve el resultado en double.

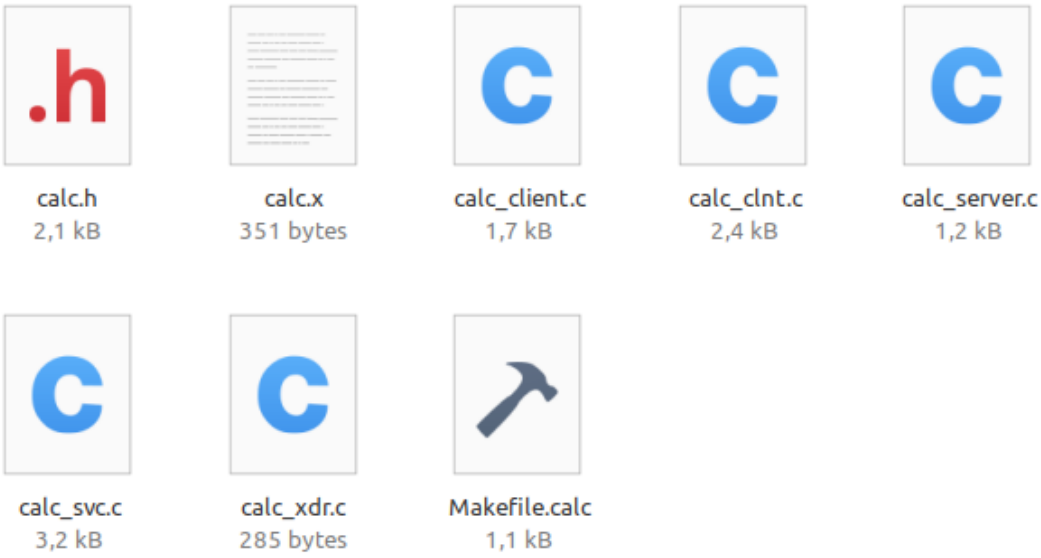
- ***Double raizCuadrada(int d):*** Calcula la raíz cuadrada de un número entero d y devuelve un resultado en double.
- ***menu(void):*** Devuelve un menú (String) con las opciones disponibles en el programa.

- El valor hexadecimal 0x20000008 es el identificador único para este programa RPC.

Una vez creado, hay que usar el comando:

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/Carpeta sin título$ rpcgen -a calc.x
```

Cuando se ejecuta el comando, se crean los siguientes archivos:



Calc.h: archivo de cabecera que se utiliza en el código fuente del cliente y el servidor para acceder a las definiciones de las estructuras y funciones RPC.

Calc_client.c: archivo del cliente el cual recoge las llamadas a las funciones del servidor.

Calc_clnt.c: esqueleto del cliente (no tocar)

Calc_server.c: archivo del servidor, el cual crea el código de las funciones descritas en el archivo.x, para que luego el cliente llame a cada una de ellas como él quiere.

Cal_svc.c: esqueleto de servidor (no tocar).

Cal_xdr.c: rutinas de filtro eXternal Data Representation (XDR) para parámetros y resultados (no tocar).

Makefile.calc: archivo para compilar todos los archivos.

*Los archivos marcados en amarillo son los que hay que modificar.

Servidor.c

Código del servidor. Aquí se recogen las funciones descritas en el archivo.x. El objetivo es crear el código de cada función.

Función Suma

```c
double *
suma_1_svc(datos *argp, struct svc_req *rqstp)
{
    static double  result;

    /*
     * insert server code here
     */
    result = argp->a + argp->b;
    return &result;
}
```

Dos argumentos:

- argp: Un puntero a una estructura de tipo datos.
- rqstp: Un puntero a una estructura que representa la solicitud del cliente.

Y realiza la suma de los dos double de la estructura pasada por puntero, por ello, se pone "argp->a" y no "argp.a".

Devuelve la suma como puntero, para que el cliente pueda acceder sin errores.

Función Resta y Multiplicación

Igual a suma, solo que cambian estas líneas:

- **Resta:** *`result = argp->a - argp->b;`*
- **Multiplicación:** *`result = argp->a * argp->b;`*

Función División

```
double *
div_1_svc(datos *argp, struct svc_req *rqstp)
{
    static double  result;
    /*
     * insert server code here
     */
    if(argp->b == 0){
        result = 0;
    }
    else{
        result = argp->a / argp->b;
    }
    return &result;
}
```

Si el segundo campo es 0, se devuelve 0, ya que no se puede dividir por 0. Si no, se realiza la división normal.

Función Potencia (elevar al cuadrado)

```
double *
potencia_1_svc(double *argp, struct svc_req *rqstp)
{
    static double  result;

    /*
     * insert server code here
     */
    result = pow(*argp,2);
    return &result;
}
```

Para poder realizar la potencia, hay que incluir la librería math.h:

```
#include <math.h>
```

Una vez incluida, se llama a la función pow(base, exponente):

- Base: puntero a un double introducido por el usuario en el cliente
- Exponente: 2 (elevar al cuadrado).

Función Raíz cuadrada

```c
double *
raizcuadrada_1_svc(int *argp, struct svc_req *rqstp)
{
    static double  result;

    /*
     * insert server code here
     */
    result = sqrt(*argp);
    return &result;
}
```

Parecida a la anterior, para poder utilizar la función sqrt, hay que incluir la librería “math.h”. la función sqrt(numero):

- Numero: puntero a un entero introducido por el usuario en el cliente.

Función Menú

```c
char **
menu_1_svc(void *argp, struct svc_req *rqstp)
{
    static char * result;

    /*
     * insert server code here
     */
    result = ANSI_COLOR_CYAN "*********************************************************************\n" ANSI_COLOR_RESET
             "1. Suma\n"
             "2. Resta\n"
             "3. Multiplicacion\n"
             "4. Division\n"
             "5. Potencia\n"
             "6. Raiz Cuadrada\n"
             ANSI_COLOR_CYAN "*********************************************************************" ANSI_COLOR_RESET;
    return &result;
}
```

Devuelvo una cadena de caracteres (string) que contiene las opciones del menú. Las anotaciones en morado son colores para la salida por la terminal.

Cliente.c

Una vez con las funciones creadas, hay que crear el código del cliente. Aquí, dependiendo de la lógica que se quiera hacer, se llamarán a las funciones del servidor.

Al usar RPCGEN, se genera por defecto las variables de cada función y las llamadas a realizar de cada función:

Variables

```
printf("Hola soy el cliente \n");
CLIENT *clnt;
// Variables generadas automáticamente
double *result_1;
datos suma_1_arg;
double *result_2;
datos resta_1_arg;
double *result_3;
datos mul_1_arg;
double *result_4;
datos div_1_arg;
double *result_5;
double potencia_1_arg;
double *result_6;
int raizcuadrada_1_arg;
char **result_7;
char *menu_1_arg;
```

Dependiendo del archivo.x, tienen un distinto tipo de datos tanto los datos de entrada como los de salida.

Llamadas:

```
#endif /* DEBUG */
    /*
    result_1 = suma_1(&suma_1_arg, clnt);
    if (result_1 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_2 = resta_1(&resta_1_arg, clnt);
    if (result_2 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_3 = mul_1(&mul_1_arg, clnt);
    if (result_3 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_4 = div_1(&div_1_arg, clnt);
    if (result_4 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_5 = potencia_1(&potencia_1_arg, clnt);
    if (result_5 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_6 = raizcuadrada_1(&raizcuadrada_1_arg, clnt);
    if (result_6 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    result_7 = menu_1((void *)&menu_1_arg, clnt);
    if (result_7 == (char **)NULL)
    {
        clnt_perror(clnt, "call failed");
    }*/
    //------------------------------------------------
```

Aquí está el código de cada llamada. Por ejemplo, para ejecutar la suma de dos números, si el usuario ya ha introducido los dos números, se copia el código de la suma, y se añade un else, donde se imprime el resultado.

Estructura de la práctica en el cliente

He creado un bucle do-while, el cual, se sale cuando el usuario pulsa Ctrl +D. Lo primero que se muestra es la función del menú (procedimiento 7) que devuelve el mensaje con las opciones del programa.

```c
do
{
    //llamada al menú
    result_7 = menu_1((void *)&menu_1_arg, clnt);
    if (result_7 == (char **)NULL)
    {
        clnt_perror(clnt, "call failed");
        continue;
    }
    else
    {
        system("clear");
        printf("MENÚ recibido del servidor:\n%s\n", *result_7);
    }
```

Luego, se pide al usuario un número. Si el número es entre 1 a 6, se ejecuta la función designada con ese número, y, si se introduce otro número u otro carácter, te vuelve a pedir otro número. Num1 y num2 son para las primeras 4 opciones. Utilizo también num1 para la potencia. Num3 es para la raíz cuadrada (entero de entrada).

```c
int opcion;
printf(ANSI_COLOR_GREEN "Opcion: " ANSI_COLOR_RESET);
scanf("%d", &opcion);

switch (opcion)
{
    double num1, num2;
    int num3;
```

Voy a usar el caso de la suma, pero es igual o muy parecido al resto. Lo primero, llamo a una función auxiliar que me he creado, que lo que hace es pedir dos números por teclado y asignarles ese valor a las variables num1 y num2. Una vez recogido el valor, se los paso a los argumentos de suma (los dos campos de la estrcutura). Luego, copio la llamada del código generado automáticamente, y en el else, es donde devuelvo el resultado.

```c
case 1:
    printf(ANSI_COLOR_BLUE "SUMA DE DOS NÚMEROS\n" ANSI_COLOR_RESET);
    pedirDosNumeros(&num1, &num2);

    suma_1_arg.a = num1;
    suma_1_arg.b = num2;

    result_1 = suma_1(&suma_1_arg, clnt);
    if (result_1 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    else
    {
        printf("\nResultado de la suma: %f\n", *result_1);
    }
    break;
```

Aquí está la función de pedir 2 números:

```c
void pedirDosNumeros(double *num1, double *num2)
{
    printf("\n-->Ingrese el primer número: ");
    scanf("%lf", num1);
    printf("-->Ingrese el segundo número: ");
    scanf("%lf", num2);
}
```

Voy a enseñar la división, la cual tiene el caso especial de que el divisor sea 0. Como del servidor se devuelve 0, si el divisor es 0, con un if se controla el caso:

```c
case 4:
    printf(ANSI_COLOR_BLUE "DIVISIÓN DE DOS NÚMEROS\n" ANSI_COLOR_RESET);
    pedirDosNumeros(&num1, &num2);

    div_1_arg.a = num1;
    div_1_arg.b = num2;

    result_4 = div_1(&div_1_arg, clnt);
    if (result_4 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    else
    {
        if (*result_4 == 0)
        {
            printf(ANSI_COLOR_RED"\nEl divisor es 0. No puede hacerse la divisi
        }
        else
        {
            printf("\nResultado de la division: %f\n", *result_4);
        }
    }
    break;
```

Ahora, voy a enseñar el caso de la potencia y de la raíz cuadrada. En el caso de la potencia, hay que introducir un número double, por eso, utilizo num1 como he explicado arriba.

```c
case 5:
    printf(ANSI_COLOR_BLUE "POTENCIA DE UN NÚMERO\n" ANSI_COLOR_RESET);
    printf("-->Introduce un número: ");
    scanf("%lf", &num1);

    potencia_1_arg = num1;

    result_5 = potencia_1(&potencia_1_arg, clnt);
    if (result_5 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    else
    {
        printf("\nResultado de la potencia: %f\n", *result_5);
    }
    break;
```

Y la raíz cuadrada, también necesita un número, la única diferencia es que tiene que ser un número entero:

```c
case 6:
    printf(ANSI_COLOR_BLUE "RAÍZ CUADRADA DE UN NÚMERO\n" ANSI_COLOR_RESET);
    printf("-->Introduce un número entero: ");
    scanf("%d", &num3);

    raizcuadrada_1_arg = num3;

    result_6 = raizcuadrada_1(&raizcuadrada_1_arg, clnt);
    if (result_6 == (double *)NULL)
    {
        clnt_perror(clnt, "call failed");
    }
    else
    {
        printf("\nResultado de la raíz cuadrada: %f\n", *result_6);
    }
    break;
```

Al final, he añadido un getchar(), acompañado de la función system (“clear”) del inicio del do, para limpiar la consola, y que el menú quede siempre fijo arriba.

```c
        break;
    default:
        printf(ANSI_COLOR_RED "Opción incorrecta. Vuelve a introducir una opción cc
        break;
    }

    printf(ANSI_COLOR_GREEN"\nPulsa cualquier tecla para continuar...\n"ANSI_COLOR_
    getchar();

} while (getchar() != EOF);

printf("\nSe ha pulsado CTRL + D. Sale del programa\n");
```

Compilar con Makefile

Una vez creado tanto el cliente como el servidor, hay que compilar todos los archivos. Esto se hace rápido gracias al archivo de MakeFile. Pero primero hay que modificarlo:

```
CFLAGS += -g -I/usr/include/tirpc
LDLIBS += -lnsl -ltirpc -lm
RPCGENFLAGS =
```

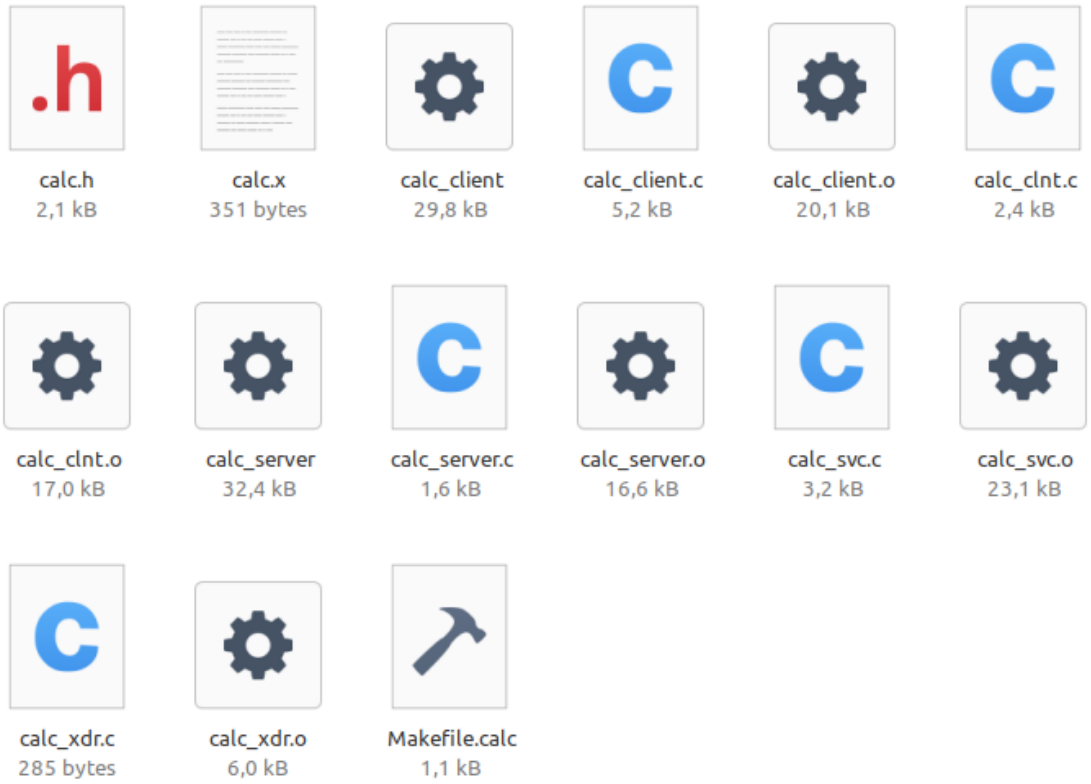
Incluir en CFLAGS la ruta de tirpc, para que la biblioteca rpc.h se pueda utilizar en los archivos e incluir en LDLIBS para usar las bibliotecas de NSL, RPC y la de matemáticas (raíz cuadrada y potencia).

Una vez modificado el archivo, para compilar todos los archivos, se utiliza el siguiente comando:

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/Carpeta sin título$ make -f Makefile.calc
```

Esto genera todos los archivos .o.

Y queda la carpeta así:



Ejecución de la práctica

Para ejecutar el código, una vez compilado los archivos con el comando MakeFile, hay que abrir la terminal de Linux. Luego hay que abrir otra pestaña de terminal, para ejecutar por un lado el servidor y por otro lado el cliente.

Ejecución del servidor. Hay que usar el siguiente comando:

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/Calculadora$ ./calc_server
```

Ejecución del cliente. Siguiente comando:

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/Calculadora$ ./calc_client localhost
```

* En vez de localhost, puede ser la dirección IP asignada de la máquina virtual (es lo mismo).

A continuación, voy a mostrar un ejemplo de cada opción de la calculadora:

Menú

```
MENÚ recibido del servidor:
**************************************************************************
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Potencia
6. Raiz Cuadrada
**************************************************************************
Opcion:
```

Suma

```
Opcion: 1
SUMA DE DOS NÚMEROS

-->Ingrese el primer número: 5.5
-->Ingrese el segundo número: 10.1

Resultado de la suma: 15.600000

Pulsa cualquier tecla para continuar...
```

Resta

```
Opcion: 2
RESTA DE DOS NÚMEROS

-->Ingrese el primer número: 6.8
-->Ingrese el segundo número: 106.5

Resultado de la resta: -99.700000

Pulsa cualquier tecla para continuar...
```

Multiplicación

```
Opcion: 3
MULTIPLICACIÓN DE DOS NÚMEROS

-->Ingrese el primer número: -4.5
-->Ingrese el segundo número: -2

Resultado de la multiplicación: 9.000000

Pulsa cualquier tecla para continuar...
```

División

```
Opcion: 4
DIVISIÓN DE DOS NÚMEROS

-->Ingrese el primer número: 100
-->Ingrese el segundo número: 4

Resultado de la division: 25.000000

Pulsa cualquier tecla para continuar...
```

```
Opcion: 4
DIVISIÓN DE DOS NÚMEROS

-->Ingrese el primer número: 100
-->Ingrese el segundo número: 0

El divisor es 0. No puede hacerse la división

Pulsa cualquier tecla para continuar...
```

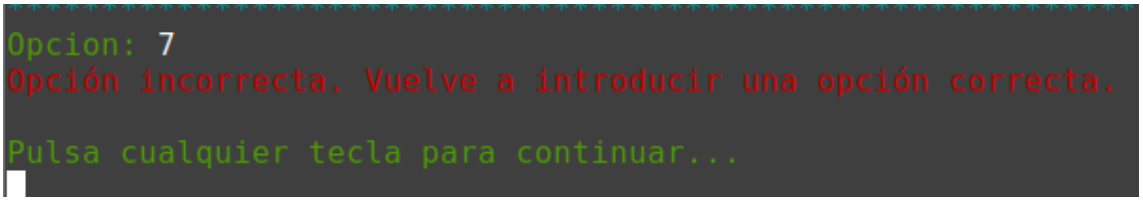
Potencia

```
Opcion: 5
POTENCIA DE UN NÚMERO

-->Introduce un número: 9

Resultado de la potencia: 81.000000

Pulsa cualquier tecla para continuar...
```

Raíz Cuadrada

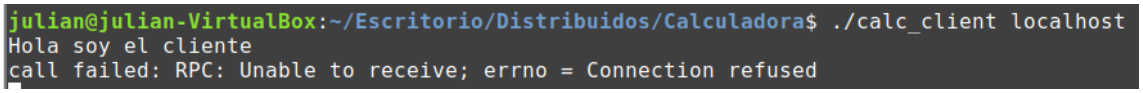
```
Opcion: 6
RAÍZ CUADRADA DE UN NÚMERO

-->Introduce un número entero: 81

Resultado de la raíz cuadrada: 9.000000

Pulsa cualquier tecla para continuar...
```

Opción incorrecta

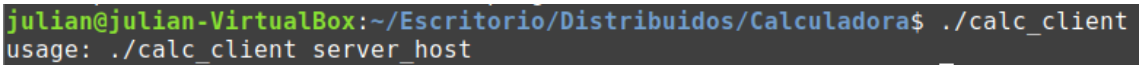


Ejecuciones malas

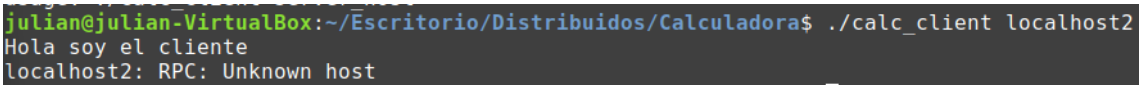
Ejecutar el cliente sin tener el cliente abierto



Ejecutar el cliente sin el argumento de la ip del server



Ejecutar el cliente con una ip incorrecta del server



Problemas encontrados

No me compilaba bien al usar MakeFile, y el problema era porque, la biblioteca de math.h, la estaba añadiendo así:

#include "math.h", en vez de: *#include <math.h>*

También, una vez que tenía el servidor creado bien, y el cliente también, al ejecutar el cliente, y hacer la primera llamada del menú, se me quedaba en espera el cliente. El problema es que hay que comentar las llamadas que te crean por defecto rpcgen.

Bibliografía

Para hacer la práctica he usado la siguiente documentación:

PDF de la práctica: S1.pdf

Y para informarme y mejorar la documentación, he seguido estas páginas:

Compilador de protocolo rpcgen - Documentación de IBM

Mandato rpcgen - Documentación de IBM

Construcción de un servidor de archivos remoto con rpcgen (studylib.es)

Colores por consola en c

https://gist.github.com/Alfonzzoj/db207b89d56f24d9d0b17ff93e091be8

https://www.somosbinarios.es/como-usar-colores-en-c/