Universidad de Granada

Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

PRÁCTICA 2
Llamada remota a procedimiento (RPC)

Desarrollo de Sistemas Distribuidos

Autor:
Raquel Molina Reche

Abril 2021

1. Introduction

Para la realización de esta práctica he tenido en cuenta 4 grupos de operaciones según la estructura de datos que estas manejan:

- Operaciones Básicas.
  Suma, resta, producto, división, logaritmo y potencia.
- Operaciones con vectores de cualquier tamaño.
  Suma y resta de vectores.suma de
- Operaciones con vectores 3D.
  Producto vectorial y producto escalar de vectores de 3 dimensiones.
- Operaciones con matrices cuadradas.
  Suma, resta y producto de matrices.

Todo esto haciendo uso de la estructura cliente servidor mediante llamadas a procedimientos remotos con **RPC Sun**.

2. Planteamiento de la solución

El programa se muestra en la Figura 1, contiene 13 procedimientos que equivalen a las distintas operaciones que se ofrecen. Para los argumentos se han utilizado estructuras de datos de tipo *struct* que contienen los operandos, y para el resultado se ha usado una estructura de datos de tipo *union* para el manejo de errores.

```
66  program CALCULADORAPROG {
67
68      version CALCULADORAVERS {
69          responseBasic SUMA (operationBasic) = 1;
70          responseBasic RESTA (operationBasic) = 2;
71          responseBasic MULTIPLICACION (operationBasic) = 3;
72          responseBasic DIVISION (operationBasic) = 4;
73          responseBasic LOGARITMO (operationBasic) = 5;
74          responseBasic POTENCIA (operationBasic) = 6;
75          responseVectores SUMAVECTORES(operationVectores) = 7;
76          responseVectores RESTAVECTORES(operationVectores) = 8;
77          responseBasic PRODESCALAR3D(operationVectores3D) = 9;
78          responseVectores PRODVECTORIAL3D(operationVectores3D) = 10;
79          responseMatrices SUMAMATRICES(operationMatrices) = 11;
80          responseMatrices RESTAMATRICES(operationMatrices) = 12;
81          responseMatrices PRODUCTOMATRICES(operationMatrices) = 13;
82
83      } = 1;
84
85  } = 0x20000001;
86
```

Figura 1: Programa y procedimientos

2.1. Operaciones Básicas

Para las operaciones básicas sólo se han tenido en cuenta las estructuras de la Figura 2, *operationBasic* para manejar los operandos y *responseBasic* para el resultado.

El servidor realiza el cálculo tal y como se muestra en la Figura 3. Primero se libera la memoria que se asignó en una ejecución previa del servidor para el resultado, después se realiza el cálculo y por último se guarda el resultado. En este caso se usan punteros para facilitar la implementación. Este procedimiento se repite para las distintas operaciones básicas, solo cambia el código asociado al cálculo.

```
struct operationBasic{
    double first;
    double second;
};

union responseBasic switch (int error){
    case 0:
        double result;
    default:
        void;
};
```

Figura 2: Estructuras de las operaciones básicas

```
responseBasic *
suma_1_svc(operationBasic operands,  struct svc_req *rqstp)
{
    static responseBasic  result;

    double *calculationp;
    double calculation;

    //Se libera la memoria que se asigno en una ejecucion previa del servidor para el resultado
    xdr_free(xdr_responseBasic, &result);

    //El puntero calculationp apunta a la direccion de memoria de result.response_u.result
    calculationp = &result.responseBasic_u.result;

    //Se calcula el resultado de la operacion
    calculation = operands.first + operands.second;


    //Se cambia el contenido hacia donde apunta calculationp por el resultado de la operacion
    //Por lo que se cambia el result.response_u.result
    (*calculationp) = calculation;

    return &result;
}
```

Figura 3: Operaciones básicas en el servidor

2.2. Operaciones con vectores de cualquier tamaño

Para las operaciones con vectores de cualquier tamaño,se han tenido en cuenta las estructuras de la Figura 4, *vectorData* como tipo vector que almacena el tamaño del vector y un puntero al primer elemento del mismo, *operationVectores* para manejar los operandos y *responseVectores* para el resultado.

El servidor realiza el cálculo tal y como se muestra en la Figura 5. Primero se libera la memoria que se asignó en una ejecución previa del servidor para el resultado, después se asigna la dimensión del vector resultante y se redimensiona, se realiza el cálculo y por último se guarda el resultado. Este procedimiento se repite de la misma forma para la resta, solo cambia el código asociado al cálculo.

```
typedef double vectorData<>;

struct operationVectores{
    vectorData first;
    vectorData second;
};

union responseVectores switch (int error){
    case 0:
        vectorData vResult;
    default:
        void;
};
```

Figura 4: Estructuras de las operaciones con vectores

```
responseVectores *
sumavectores_1_svc(operationVectores operands,  struct svc_req *rqstp)
{
    static responseVectores  result;

    //Puntero a la dimension del vector
    int *dim;
    dim = &result.responseVectores_u.vResult.vectorData_len;

    //Se libera la memoria que se asigno en una ejecucion previa del servidor para el resultado
    xdr_free(xdr_responseVectores, &result);

    //Se asigna la dimension
    (*dim) =  operands.first.vectorData_len;
    //Se redimensiona
    result.responseVectores_u.vResult.vectorData_val = malloc(operands.first.vectorData_len*sizeof(double));

    //Se calcula el resultado de la operacion
    for(int i = 0; i < operands.first.vectorData_len;i++){
        result.responseVectores_u.vResult.vectorData_val[i]  = operands.first.vectorData_val[i] + operands.second.vectorData_val[i];
    }

    return &result;
}
```

Figura 5: Operaciones con vectores en el servidor

2.3. Operaciones con vectores 3D

Para las operaciones con vectores 3D, se han tenido en cuenta las estructuras de la Figura 7, *vector3D* como una estructura que contiene el valor de las 3 coordenadas (x y z) y *operationVectores3D* para manejar los operandos.

El servidor realiza el cálculo tal y como se muestra en la Figura 7. Primero se libera la memoria que se asignó en una ejecución previa del servidor para el resultado, después se realiza el cálculo y por último se guarda el resultado.

Para el producto escalar el resultado es un dato de tipo double por ello la estructura que usa para la salida es responseBasic. Para el caso de producto vectorial, el resultado es un vector. Por facilidad se ha usado la misma estructura que para los vectores de cualquier tamaño pero se ha asignado su tamaño fijo a 3.

```
union responseVectores switch (int error){
    case 0:
        vectorData vResult;
    default:
        void;
};

struct vector3D{
    double x;
    double y;
    double z;
};

typedef struct vector3D vector3D;

struct operationVectores3D{
    vector3D first;
    vector3D second;
};
```

Figura 6: Estructuras de las operaciones con vectores 3D

```
responseBasic *
prodescalar3d_1_svc(operationVectores3D operands,  struct svc_req *rqstp)
{
    static responseBasic  result;

    double *calculationp;
    double productoEscalar;


    //Se libera la memoria que se asigno en una ejecucion previa del servidor para el resultado
    xdr_free(xdr_responseBasic, &result);

    //El puntero calculationp apunta a la direccion de memoria de result.response_u.result
    calculationp = &result.responseBasic_u.result;

    //Se calcula el resultado de la operacion
    productoEscalar = (operands.first.x*operands.second.x) + (operands.first.y*operands.second.y) + (operands.first.z*operands.second.z);

    //Se cambia el contenido hacia donde apunta calculationp por el resultado de la operacion
    //Por lo que se cambia el result.response_u.result
    (*calculationp) = productoEscalar;

    return &result;
}
```

Figura 7: Operaciones con vectores 3D en el servidor

2.4. Operaciones con matrices cuadradas

Para las operaciones con matrices cuadradas, se han tenido en cuenta las estructuras de la Figura 8, *operationMatrices* para manejar los operandos, *responseMatrices* para el resultado y *matriz* como una estructura que contiene la información de la misma, es decir la estructura almacena el número de filas, el

número de columnas y un dato de tipo *vectorData*. Este vector tendrá de tamaño filas*columnas y el puntero al primer elemento del vector y por tanto de la matriz. El acceso a los elementos del vector se realiza de la siguiente manera:

*(fila a la que se quiere acceder * número de columnas de la matriz) + columna a la que se quiere acceder*

Con esta estructura se podrían manejar matrices que no fueran cuadradas ya que se almacena tanto el valor de las filas como el de las columnas pero se ha restringido en este caso a matrices cuadradas por la comodidad de las operaciones.

El servidor realiza el cálculo tal y como se muestra en la Figura 9. Primero se libera la memoria que se asignó en una ejecución previa del servidor para el resultado, después se asigna el valor de las filas y las columnas de la matriz resultante (en este caso será el mismo valor que las filas y columnas de los operandos), se asigna al vector su tamaño como el producto de las filas y las columnas y se redimensiona. Se realiza el cálculo y por último se guarda el resultado.

Para la resta y el producto la lógica del procedimiento es la misma solo varía la implementación asociada al cálculo del resultado.

```
typedef struct matriz matrizData;

struct matriz{
    int f;
    int c;
    vectorData m;
};

struct operationMatrices{
    matrizData first;
    matrizData second;
};

union responseMatrices switch (int error){
    case 0:
        matrizData mResult;
    default:
        void;
};
```

Figura 8: Estructuras de las operaciones con matrices

```c
responseMatrices *
sumamatrices_1_svc(operationMatrices operands,  struct svc_req *rqstp)
{
    static responseMatrices  result;

    //Se libera la memoria que se asigno en una ejecucion previa del servidor para el resultado
    xdr_free(xdr_responseMatrices, &result);

    //Se asigna al resultado en valor de las columnas
    int c = operands.first.c;
    result.responseMatrices_u.mResult.c = c;

    //Se asigna al resultado en valor de las filas
    int f = operands.first.f;
    result.responseMatrices_u.mResult.f = f;

    //Se asigna al resultado en valor del leng que será el numero de filas*columas
    result.responseMatrices_u.mResult.m.vectorData_len= f*c;

    //Se redimensiona por tando el vector que compone la matriz
    result.responseMatrices_u.mResult.m.vectorData_val = malloc( f*c*sizeof(double));

    int indice;
    double suma;


    //Se calcula el resultado de la operacion de producto de matrices
    for(int i = 0; i < f; i++){
        for(int j = 0; j < c; j++){
            //El valor del indice vendrá dado por la fila actual (i)* numero de columnas(c) + j
            indice = (i*c)+j;
            suma = operands.first.m.vectorData_val[indice] + operands.second.m.vectorData_val[indice];
            result.responseMatrices_u.mResult.m.vectorData_val[indice] = suma;
        }
    }

    return &result;
}
```

Figura 9: Operaciones con matrices en el servidor

3. Compilación y ejemplos de ejecución

3.1. Compilación

$\$gcc\ calculadora\_server.c\ calculadora\_svc.c\ calculadora\_xdr.c\ -o\ servidor\ -lnsl\ -lm$

$\$gcc\ calculadora\_client.c\ calculadora\_clnt.c\ calculadora\_xdr.c\ -o\ cliente\ -lnsl$

3.2. Ejemplos de ejecución

Para la ejecución en el cliente se han manejado las operaciones a partir de un menú en el que se van introduciendo las distintas peticiones y cuya implementación llama a los métodos necesarios para obtener los resultados.

$ ./servidor
$ ./cliente localhost

3.2.1. Operaciones Básicas

```
ubuntu@dsd1:~/practica2/repositorio/DSD-UGR/Practica2/rpc$ ./cliente localhost

Opciones disponibles:
  1: Operaciones Básicas
  2: Operaciones con vectores
  3: Operaciones con vectores 3D
  4: Operaciones con matrices cuadradas
  5: Salir

--Introducce una opción: 1

       ----OPERACIÓN BÁSICA----
       Introduce la operacion  +  -  *  /  log  ^ : +
       Introduce el primer operando: 55.5
       Introduce el segundo operando: 5.5

-----------------------------------------------------------------
  El resultado de la operación 55.500000 + 5.500000 = 61.000000
-----------------------------------------------------------------
```

Figura 10: Suma

```
 Opciones disponibles:
   1: Operaciones Básicas
   2: Operaciones con vectores
   3: Operaciones con vectores 3D
   4: Operaciones con matrices cuadradas
   5: Salir

 --Introducce una opción: 1

         ----OPERACIÓN BÁSICA----
         Introduce la operacion  +  -  *  /  log  ^ : -
         Introduce el primer operando: 65
         Introduce el segundo operando: 5

 -----------------------------------------------------------------
   El resultado de la operación 65.000000 - 5.000000 = 60.000000
 -----------------------------------------------------------------
```

Figura 11: Resta

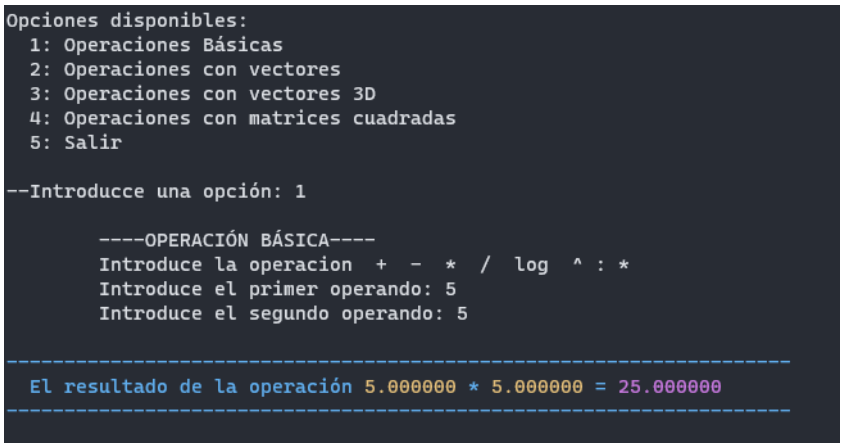

Figura 12: Multiplicación

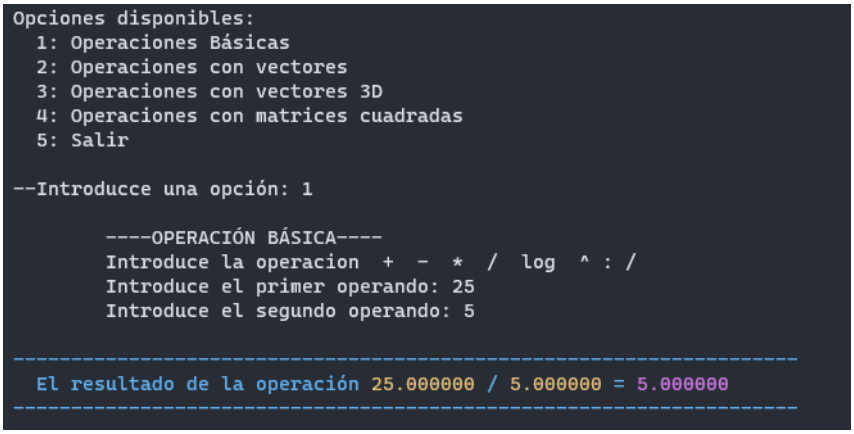

Figura 13: División

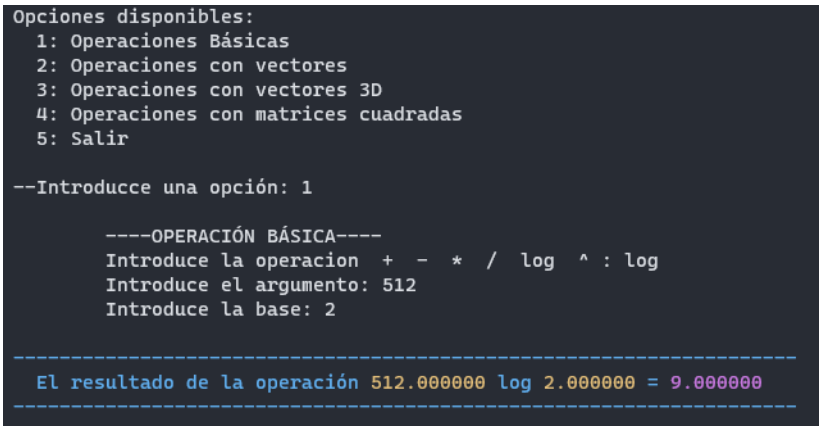

Figura 14: Logaritmo

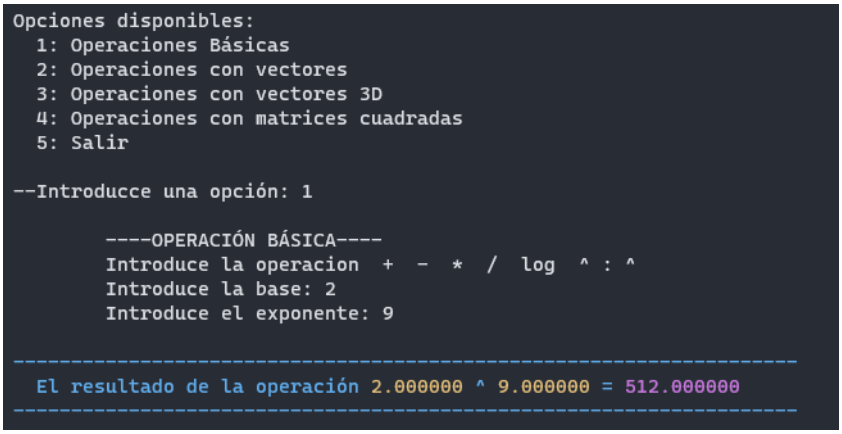

Figura 15: Potencia

3.2.2. Operaciones con vectores de cualquier tamaño

```
Opciones disponibles:
  1: Operaciones Básicas
  2: Operaciones con vectores
  3: Operaciones con vectores 3D
  4: Operaciones con matrices cuadradas
  5: Salir

--Introducce una opción: 2

        ----OPERACIÓN CON VECTORES----
        Introduce el tamaño de los vectores: 6
        Contenido del primer vector (v1):
        v1[0]: 2.3
        v1[1]: 5.6
        v1[2]: 9.5
        v1[3]: 1.2
        v1[4]: 6.2
        v1[5]: 3.8
        Contenido del segundo vector (v2):
        v2[0]: 8.3
        v2[1]: 8.2
        v2[2]: 9.5
        v2[3]: 7.3
        v2[4]: 7.4
        v2[5]: 5.

        Opciones disponibles con v1 y v2:
            1: Suma
            2: Resta
            3: Salir

            --Introducce una opción: 1
-----------------------------------------------------------------------
 El resultado de la operación:
        2.300000 5.600000 9.500000 1.200000 6.200000 3.800000
                 +
        8.300000 8.200000 9.500000 7.300000 7.400000 5.000000
        = 10.600000  13.800000  19.000000  8.500000  13.600000  8.800000
-----------------------------------------------------------------------

        Opciones disponibles con v1 y v2:
            1: Suma
            2: Resta
            3: Salir

            --Introducce una opción: 2
-----------------------------------------------------------------------
 El resultado de la operación:
        2.300000 5.600000 9.500000 1.200000 6.200000 3.800000
                 -
        8.300000 8.200000 9.500000 7.300000 7.400000 5.000000
        = -6.000000  -2.600000  0.000000  -6.100000  -1.200000  -1.200000
-----------------------------------------------------------------------
```

Figura 16: Suma y resta de vectores

3.2.3. Operaciones con vectores 3D

Figura 17: Producto escalar y producto vectorial de vectores 3D

3.2.4. Operaciones con matrices cuadradas

```
Opciones disponibles:
  1: Operaciones Básicas
  2: Operaciones con vectores
  3: Operaciones con vectores 3D
  4: Operaciones con matrices cuadradas
  5: Salir

--Introducce una opción: 4

        ----OPERACIONES CON MATRICES CUADRADAS----
        Introduce el numero de filas y columnas: 4
        Contenido de la primera matriz (m1):
                m1[0][0]: 1
                m1[0][1]: 4
                m1[0][2]: 7
                m1[0][3]: 1
                m1[1][0]: -1
                m1[1][1]: 2
                m1[1][2]: 2
                m1[1][3]: 5
                m1[2][0]: 8
                m1[2][1]: 2
                m1[2][2]: -1
                m1[2][3]: 2
                m1[3][0]: 0
                m1[3][1]: 0
                m1[3][2]: 0
                m1[3][3]: 3
        Contenido de la segunda matriz (m2):
                m2[0][0]: 6
                m2[0][1]: -4
                m2[0][2]: 5
                m2[0][3]: 6
                m2[1][0]: -2
                m2[1][1]: 1
                m2[1][2]: 1
                m2[1][3]: 2
                m2[2][0]: 8
                m2[2][1]: 1
                m2[2][2]: 2
                m2[2][3]: 0
                m2[3][0]: 0
                m2[3][1]: 0
                m2[3][2]: 1
                m2[3][3]: 1
```

Figura 18: Incluir datos de matrices cuadradas

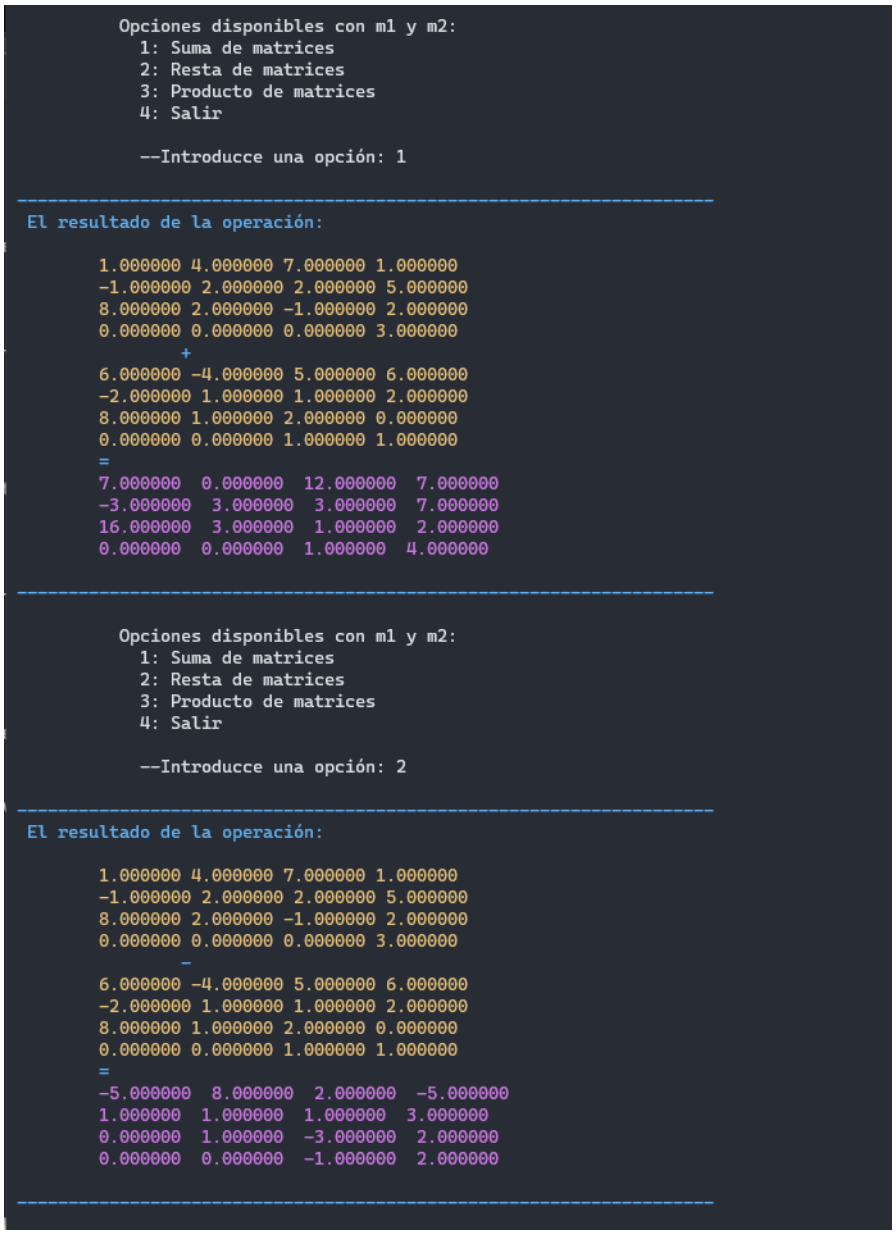

Figura 19: Suma y resta de matrices cuadradas

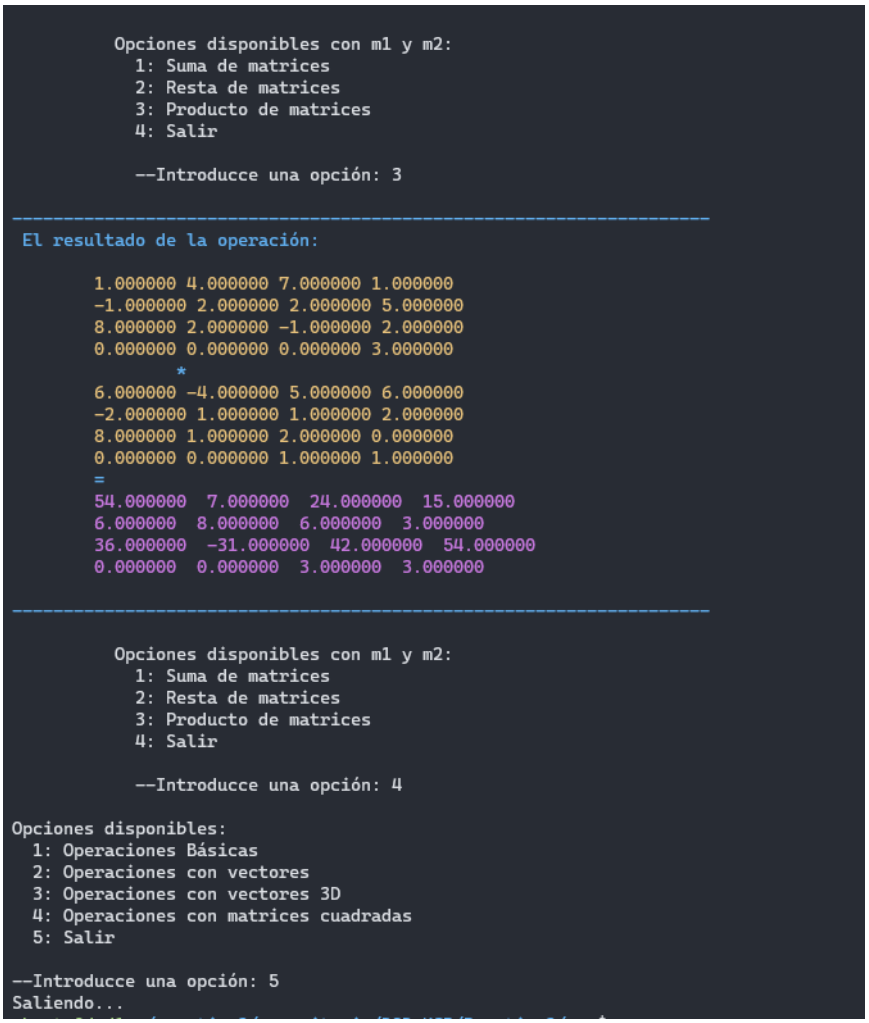

Figura 20: Producto de matrices cuadradas