GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE SISTEMAS DISTRIBUIDOS

UNIVERSIDAD
DE GRANADA

P2 : Apache Thrift

Juan Miguel Acosta Ortega (*acostaojuanmi@correo.ugr.es*)

4 de abril de 2024

Índice

Objetivos

La práctica consiste en realizar una calculadora que realice las operaciones básicas descritas en el guión de la primera parte de la práctica 2 (suma, resta, multiplicación y división) y además las siguientes:

1. Seno (grados).
2. Coseno (grados).
3. Tangente (grados).
4. Conversión de grados a radianes y viceversa.

Sin embargo, habrá que extender la calculadora para que realice operaciones más complejas.

1. Primera versión de la calculadora : Versión básica

Para comenzar esta primera versión de la calculadora con Apache Thrift en primer lugar deberemos pensar las estructuras necesarias y plasmarlas en el ".thrift", además hemos de crear las operaciones remotas de nuestro servicio, en este caso he decidido crear un struct result"para contener el resultado y la excepción correspondiente en caso de haberla. El resultado de las operaciones se representará con dicha estructura:

```
1
2    struct result {
3        1: required double data;
4        2: required i16 errnum;
5        3: required string msg;
6    }
7
8    service Calculadora{
9       void ping(),
10      result suma(1:double num1, 2:double num2),
11      result resta(1:double num1, 2:double num2),
12      result multiplicacion(1:double num1, 2:double num2),
13      result division(1:double num1, 2:double num2),
14      result seno(1:double num),
15      result coseno(1:double num),
16      result tangente(1:double num),
17      result convGradosRadianes(1:double grado),
18      result convRadianesGrados(1:double radianes)
19   }
```

Listing 1: Estructuras y procedimientos para la 1ª versión

En esta primera versión funcional de la calculadora se han implementado las llamadas a los procedimientos requeridos en la misma, estos son:

1. Suma.
2. Resta.
3. Multiplicación.
4. División.
5. Seno (grados).
6. Coseno (grados).
7. Tangente (grados).
8. Conversión de grados a radianes y viceversa.

Para crear la calculadora de manera interactiva sigo la metodología usada en la primera práctica, es decir, solicito al usuario que inserte el tipo de operación deseada (definida por un entero), y más adelante según el tipo de operación solicito los datos necesarios para realizarla.
He utilizado un diccionario con clave entero (define la operación) y valor el procedimiento para así poder simplificar el código que llame a los procedimientos.
Esto a su vez me es muy útil para ir añadiendo operaciones diversas de manera sencilla y escalable.

```
operaciones_binarias = [1,2,3,4]
operaciones_unarias = [5,6,7,8,9]

operaciones = {
    1: "suma",
    2: "resta",
    3: "multiplicacion",
    4: "division",
    5: "seno",
    6: "coseno",
    7: "tangente",
    8: "convGradosRadianes",
    9: "convRadianesGrados"
}
```

Listing 2: Diccionario utilizado para simplificar las llamadas a los procedimientos

La manera en la que gestiono las excepciones se refleja con este ejemplo de implementación de la división:

```
#PROCEDIMIENTO DEL SERVIDOR

    def division(self, num1, num2):
        print("dividiendo " + str(num1) + " con " + str(num2))

        result = Calculadora.result()
        noError(result)

        if (num2 == 0):
            result.errnum=1
            result.msg="Divisi n entre 0"
        else:
            result.data = num1 / num2
            result.msg="La divisi n " + str(num1) + " / " + str(num2) + " es: " + str(
result.data)

        return result

#PROCEDIMIENTO DEL CLIENTE

   if operacion in operaciones_binarias:
        resultado = getattr(client, operaciones[operacion])(num1, num2)
        if (resultado.errnum == 0):
            print(resultado.msg)
        else:
            print(f"Ocurri  un error con c digo {resultado.errnum}, que refiere al
problema: {resultado.msg}")
```

Listing 3: Diccionario utilizado para simplificar las llamadas a los procedimientos

Por último muestro un ejemplo de la llamada a una división errónea:

```
1. Suma
2. Resta
3. Multiplicaci n
4. Divisi n
5. Seno (grad)
6. Coseno (grad)
7. Tangente (grad)
8. Conversi n Grados-Radianes
9. Conversi n Radianes-Grados

Introduce una de las operaciones presentadas:
Operacion: 4
Primer numero: 6
Segundo numero: 0
```

```
16      Ocurrio un error con codigo 1, que refiere al problema: Division entre 0
17
18      Process finished with exit code 0
```

Listing 4: Salida de división errónea

2. Segunda versión de la calculadora : Introducimos un servidor encargado de operaciones algebraicas (aritmética entera modular)

En esta segunda versión de la calculadora he decidido implementar un segundo servidor en el puerto inmediatamente superior al primero, es decir, el 9091.

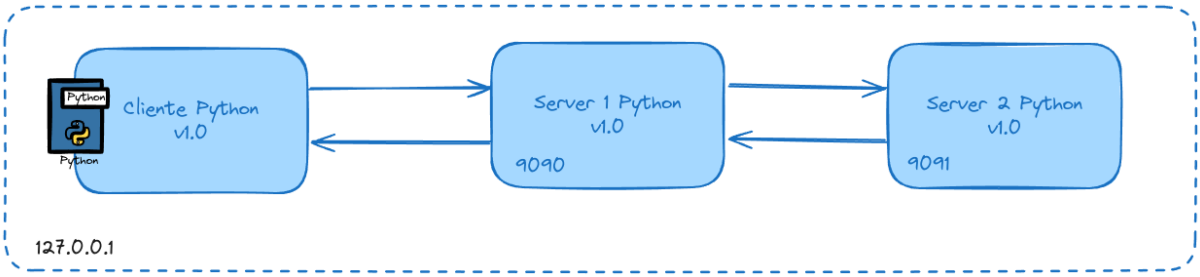

Figura 1: Arquitectura actual Cliente / Servidor.

Este tendrá el rol de hacer operaciones relacionadas con arirtmética entera modular que el primer servidor no "saberealizar, por lo tanto los procedimientos de este tipo están implementados en el servidor .ªlgebraico", mientras que el primer servidor se encarga de actuar también como cliente cuando se trata de este tipo de procedimientos.

Para comenzar con este apartado aclaramos en el archivo calculadora.thrift tanto las estructuras necesarios como los procedimientos nuevos. Por supuesto se ha de crear un nuevo servicio del que se encargará el nuevo servidor.

```
struct result {
    1: required double data;
    2: required i16 errnum;
    3: required string msg;
}

struct result2 {
    1: required list<i64> data;
    2: required i16 errnum;
    3: required string msg;
}

service Calculadora{
   void ping(),
   result suma(1:double num1, 2:double num2),
   result resta(1:double num1, 2:double num2),
   result multiplicacion(1:double num1, 2:double num2),
   result division(1:double num1, 2:double num2),
   result seno(1:double num),
   result coseno(1:double num),
   result tangente(1:double num),
   result convGradosRadianes(1:double grado),
   result convRadianesGrados(1:double radianes),
   void pingAlgebraica(),
   result algEuclides(1:i64 num1, 2:i64 num2),
   result2 algExtEuclides(1:i64 num1, 2:i64 num2),
   result2 congLineal(1:i64 num1, 2:i64 num2, 3:i64 num3),
   result2 ecuDiofantica(1:i64 num1, 2:i64 num2, 3:i64 num3)
}

service Algebraica{
    void ping(),
    result algEuclides(1:i64 num1, 2:i64 num2),
    result2 algExtEuclides(1:i64 num1, 2:i64 num2),
    result2 congLineal(1:i64 num1, 2:i64 num2, 3:i64 num3),
    result2 ecuDiofantica(1:i64 num1, 2:i64 num2, 3:i64 num3)
```

```
38      }
```

Listing 5: Estructuras y procedimientos para la 2ª versión

Se ha creado una nueva estructura result2 ya que los resultados de algunos procedimientos no sólo ofrecen un valor como solución. Además ciertos procedimientos sólo admiten tipos de datos enteros para poder ejecutarlos. Como detallé en el primer apartado en el cliente simplemente añado las nuevas operaciones al diccionario que recoge estas, así mismo catalogo las operaciones según tipo (un parámetro, dos parámetros enteros...), y así automatizo la manera en la que llamo a los procedimientos.

En el servidor uno por otra parte creo los objetos necesarios para que actúa como cliente y se comunique con el servidor algebraico, e implemento los nuevos procedimientos de manera que llame al procedimiento remoto del servidor 2.

```
1
2       def ecuDiofantica(self, num1, num2, num3):
3           print("mandando a calcular ecuaci n  diof ntica con formato ax + by = c siendo a=" +
        str(num1) + ", b=" +str(num2) + "y c="+str(num3))
4
5           result = self.client.ecuDiofantica(num1,num2,num3)
6           return result
```

Listing 6: Llamada a procedimiento remoto desde el servidor 1

Por último en el segundo servidor es donde se implementan las nuevas operaciones relacionadas con aritmética entera modular.

1. Algoritmo de Euclides (Resolución de mcd mediante Algoritmo de Euclides)
2. Algoritmo Extendido de Euclides
3. Congruencia lineal
4. Ecuación Diofántica

Por ejemplo esta sería la implementación de una congruencia lineal en el servidor 2:

```
1
2       def ecuDiofantica(self, num1, num2, num3):
3           print("calculando ecuaci n  diof ntica con formato ax + by = c siendo a=" + str(num1)
         + ", b=" +str(num2) + "y c="+str(num3))
4           #Ejemplo 123x +93y = 6 -> x=25 + 31 k, y= 33+41k
5
6           result = Calculadora.result2()
7           noError2(result)
8
9           if type(num1) != int or type(num2) != int or type(num3) != int:
10              result.errnum=1
11              result.msg="Los datos deben ser enteros para trabajar con esta operaci n"
12          else:
13              if (num2 < 0):
14                  num2 = num2 * -1
15
16              mcd = self.algEuclides(num1,num2)
17              mcd = mcd.data
18              if num3 % mcd == 0:
19                  x_m = self.congLineal(num1,num3,num2)
20                  x, m = x_m.data[0], x_m.data[1]
21
22                  aux1 = num1 * x
23                  aux2 = num1 * m
24
25                  aux1 = aux1 - num3
```

```
                aux1 = aux1 // num2
                aux2 = aux2 // num2

                y = aux1
                n = aux2

                result.data = [m,n]
                result.msg="{El conjunto de todas las soluciones es (x,y) = (" + str(x) + " + (" + str(m) + ")k, " + str(
                    y) + " + (" + str(n) + ")k), con k     Z}\n"
            else:
                result.errnum=2
                result.msg="La ecuaci n no tiene soluci n: " + str(num1) + "x + " + str(num2) + "y = " + str(num3) + "\n"

        return result
```

Listing 7: Implementación de ecuación diofántica

Este procedimiento usa a su vez otros procedimientos del mismo servidor como congruecias lineales y algoritmo de euclides.

Este sería la interactuación con la terminal y salida final del procedimiento.

```
Hacemos ping al server...
Hacemos ping al server algebraico...
1. Suma
2. Resta
3. Multiplicaci n
4. Divisi n
5. Seno (grad)
6. Coseno (grad)
7. Tangente (grad)
8. Conversi n Grados-Radianes
9. Conversi n Radianes-Grados
10. Algoritmo de Euclides (a, b)
11. Algoritmo Extendido de Euclides (a, b)
12. Congruencia Lineal (a*x cong b (mod m)
13. Ecuaci n diof ntica ( a*x + b*y = c)
Introduce una de las operaciones presentadas:
Operaci n: 13
Primer n mero: 123
Segundo n mero: 93
Tercer n mero:  6
{El conjunto de todas las soluciones es (x,y) = (25 + (31)k, 33 + (41)k), con k     Z}


Process finished with exit code 0
```

Listing 8: Implementación de ecuación diofántica

Por supuesto para hacer uso de los dos servicios se ha de levanmtar en primer lugar el servidor algebraico, en segundo lugar el otro servidor, y por último el cliente.

Ya que una parte muy importante de esta versión también son los métodos algebraicos voy a pasar a mostrar la resolución de un ejemplo de Algoritmo Extendido de Euclides, ya que las congruencias lineales y las ecuaciones diofánticas lo necesitan, y cómo el procedimiento remoto da la solución correcta.

Este sería el ejemplo de un algoritmo extendido de euclides con m=97, y a=35:

		u	v
m=97		1	0
a=35		0	1
27	2	1	-2
8	1	-1	3
3	3	4	-11
2	2	-9	25
1	1	u=13	v=-36

Y este es el resultado que arroja la calculadora al mismo procedimiento:

```
10. Algoritmo de Euclides (a, b)
11. Algoritmo Extendido de Euclides (a, b)
12. Congruencia Lineal (a*x cong b (mod m)
13. Ecuaci n diof ntica ( a*x + b*y = c)
Introduce una de las operaciones presentadas:
Operaci n: 11
Primer n mero: 97
Segundo n mero: 35
El MCD(97,35) es igual a 1, adem s u y v tal que 35*u+97*v=1 son u=13, y v=-36

Process finished with exit code 0
```

Listing 9: Resultado algoritmo extendido de Euclides

3. Tercera versión : Creando un cliente en otro lenguaje (Java)

Una vez diseñado el .thrift con las estructuras, datos y servicios necesarios para el desempeño de nuestra calculadora siempre compilamos los archivos con el comando ->thrift -gen py calculadora.thrift.
Sin embargo y como el "pyïndica, esto genera los archivos y conversiones necesarias para hacer el marshalling, enviar y recibir datos con python. En el caso de querer utilizar otro lenguaje para un cliente adicional o servidor se ha de compilar también para el mismo.
En este caso al querer tener otro cliente en java se ha de compilar también con ->thrift -r –gen java calculadora.thrift

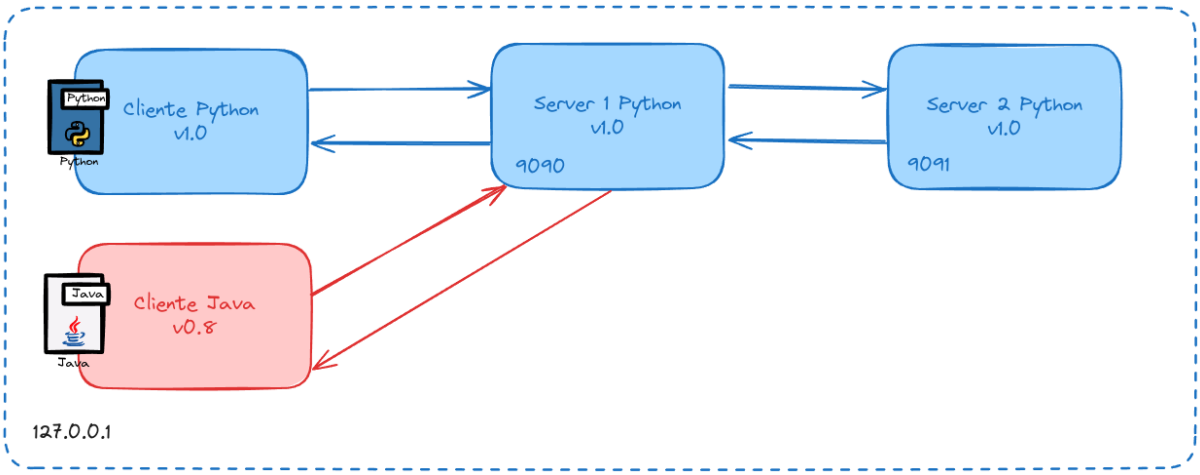

Figura 2: Arquitectura actual Cliente / Servidor con dos clientes (en versiones diferentes).

Además al ser en Java se pueden obtener la librerías de diferentes maneras, en mi caso he enlazado todo mediante maven incluyendo las dependencias necesarias, entre ellas las necesarias para usar thrift:

```
<!-- https://mvnrepository.com/artifact/org.apache.thrift/libthrift -->
        <dependency>
            <groupId>org.apache.thrift</groupId>
            <artifactId>libthrift</artifactId>
            <version>0.19.0</version>
        </dependency>
```

Listing 10: Dependencia de thrift en java

La adaptación del cliente de python al de java es prácticamente inmediata. Aunque he implementado ciertas funciones necearias como la de encontrar una operación en una lista de operaciones y he cambiado el método de escoger operación , en este caso he usado un switch case.

Además la creación de un cliente en cualquier lenguaje está bien documentado en la documentación de Apache Thrift.

```
switch (operacion) {
                case 1:
                case 2:
                case 3:
                case 4:
                    System.out.print("Primer n mero: ");
                    num1 = Double.parseDouble(scanner.nextLine().trim());
                    System.out.print("Segundo n mero: ");
                    num2 = Double.parseDouble(scanner.nextLine().trim());
                    break;
                case 5:
                case 6:
                case 7:
```

```
                    case 8:
                    case 9:
                        System.out.print("N mero: ");
                        num1 = Double.parseDouble(scanner.nextLine().trim());
                        break;
                }
```

Listing 11: Switch case para decidir cuantos parámetros insertar dependiendo de la operación

El cliente en java sólo tiene implementada la llamada a las operaciones de la primera versión del servidor. De esta manera se demuestra, como vimos en teoría, que se pueden tener diferentes clientes (versiones diferentes) funcionando con un mismo servicio.

Para utilizar este cliente se han de levantar, al igual que con el cliente de python, previamente los dos servicios. Así ejecutando el cliente de java podríamos hacer uso de él.

```
SLF4J(W): See https://www.slf4j.org/codes.html#noProviders for further details.
Hacemos ping al server...
Hacemos ping al server algebraico...
1. suma
2. resta
3. multiplicacion
4. division
5. seno
6. coseno
7. tangente
8. convGradosRadianes
9. convRadianesGrados
Introduce una de las operaciones presentadas:
1
Primer n mero: 2
Segundo n mero: 2
La suma 2.0 + 2.0 es: 4.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  12.910 s
Finished at: 2024-03-24T14:03:20+01:00
------------------------------------------------------------------------
```

Listing 12: Salida en el cliente de java

Para la comprensión de los demás ficheros generados el cliente en Java ha sido desarrollado en Apache Netbeans.

Por último y aunque nuestra estructura de datos de respuesta manejaba las distintas excepciones esperadas en las operaciones, es necesario controlar también las excepciones de caída del servidor, timeout, del socket ...
Y es por eso que en este apartado también se le añade al cliente el código necesario para esa tarea:

```
try:
    #Configuro la conexion

    #Creo el cliente

    #Abro la conexion

    #Realizo las llamadas pertinentes a los servicios

    #Cierro la conexion

except TTransport.TTransportException as ex:
    print(f"Error de transporte: {ex}")

except TProtocol.TProtocolException as ex:
    print(f"Error de protocolo: {ex}")

```

```
    except Exception as ex:
        print(f"Error inesperado: {ex}")

    finally:
        if 'transport' in locals():
            transport.close()
```

Listing 13: Control de excepciones en python

Además también se le añade al cliente de Java:

```
    TTransport tTransport = null;
    TBinaryProtocol protocol = null;
    try {
        //Configuro la conexion

        //Creo el cliente

        //Abro la conexi n

        //Realizo las llamadas pertinentes a los servicios

        //Cierro la conexi n

    } catch (TTransportException ex) {
        System.out.println("Error de transporte: " + ex.getMessage());
        // Manejar excepci n de transporte, como la ca da del servidor

    } catch (TException ex) {
        System.out.println("Error de Thrift: " + ex.getMessage());
        // Manejar excepci n de Thrift, que puede ocurrir durante la comunicaci n con el
    servidor

    } finally {
        // Cerrar la conexi n
        if (tTransport != null) {
            tTransport.close();
        }
    }
```

Listing 14: Control de excepciones en java

4. Cuarta versión : Añadiendo operaciones con vectores

Para terminar con esta última versión de la calculadora hecha en thrift he decidido añadir más llamadas a procedimientos remotos. Estas consisten en operaciones con vectores. En concreto son:

1. Suma de vectores.
2. Producto escalar.
3. Producto cruz.

Para hacer esto sigo la metodología adoptada en los pasos anteriores, es decir, primero ideo y reflejo tanto las llamadas como las operaciones necesarias en la interfaz de servicio (.thrift), y a continuación genero los ficheros necesarios.

```
struct result_vector {
    1: required list<double> data;
    2: required i16 errnum;
    3: required string msg;
}

service Calculadora{

   ...

   result_vector sumaVectores(1: list<double> vec1, 2: list<double> vec2),
   result_vector productoEscalar(1: list<double> vec1, 2: list<double> vec2),
   result_vector productoCruz(1: list<double> vec1, 2: list<double> vec2)
}
```

Listing 15: Estructuras necesarias para las operaciones con vectores

Tras haber completado este paso y comprobar que los ficheros generados son correctos y nos proveen de las herramientas necesarias para modificar tanto el cliente con el servidor procedemos con estas tareas:

```
def sumaVectores(self, v1, v2):
    print("mandando a calcular suma de vectores " + str(v1) + " y " + str(v2))

    result = Calculadora.result_vector()
    noErrorVector(result)

    print(type(v1))

    if len(v1) != len(v2):
        result.errnum=1
        result.msg="Los vectores deben tener la misma longitud"
    else:
        result.data = [x + y for x, y in zip(v1, v2)]
        result.msg= "La suma de los vectores " + str(v1) + " y " + str(v2) + " es: " + str
(result.data)

    return result

def productoEscalar(self, v1, v2):
    print("mandando a calcular producto escalar de vectores " + str(v1) + " y " + str(v2))

    result = Calculadora.result()
    noError(result)

    if len(v1) != len(v2):
        result.errnum = 1
        result.msg = "Los vectores deben tener la misma longitud"
```

```
        else:
            result.data = sum([x * y for x, y in zip(v1, v2)])
            result.msg = "El producto escalar de los vectores " + str(v1) + " y " + str(v2) +
    " es: " + str(result.data)

        return result

    def productoCruz(self, v1, v2):
        print("mandando a calcular producto cruz de vectores " + str(v1) + " y " + str(v2))

        result = Calculadora.result_vector()
        noErrorVector(result)

        if len(v1) != 3 or len(v2) != 3:
            result.errnum = 1
            result.msg = "Los vectores deben tener longitud 3"
        else:
            result.data = [v1[1] * v2[2] - v1[2] * v2[1], v1[2] * v2[0] - v1[0] * v2[2], v1[0]
     * v2[1] - v1[1] * v2[0]]
            result.msg = "El producto cruz de los vectores " + str(v1) + " y " + str(v2) + "
    es: " + str(result.data)

        return result
```

Listing 16: Operaciones con vectores en el servidor

5. Manual de uso

En este manual de uso se pretende mostrar el funcionamiento de la calculadora, y para ello se mostrarán un par de ejemplos de las operaciones más peculiares, ya que en las operaciones simples como la suma sólo hay que ingresar tipo de operación (1), primer número (x), y segundo número (y), y esta arrojaría el resultado (x+y=z).

En primer lugar veamos el ejemplo de entradas en la terminal para trabajar con vectores:

```
Hacemos ping al server...
Hacemos ping al server algebraico...
1. Suma
2. Resta
3. Multiplicaci n
4. Divisi n
5. Seno (grad)
6. Coseno (grad)
7. Tangente (grad)
8. Conversi n Grados-Radianes
9. Conversi n Radianes-Grados
10. Algoritmo de Euclides (a, b)
11. Algoritmo Extendido de Euclides (a, b)
12. Congruencia Lineal (a*x cong b (mod m)
13. Ecuaci n diof ntica ( a*x + b*y = c)
14. Suma de los vectores (v1 + v2)
15. Producto escalar de los vectores (v1 . v2)
16. Producto cruz de los vectores (v1 x v2)
Introduce una de las operaciones presentadas:
Operaci n: 14
Introduce los vectores en formato de lista [x,y,z]
Primer vector: [1,2,3]
Segundo vector: [1,2,3]
La suma de los vectores [1.0, 2.0, 3.0] y [1.0, 2.0, 3.0] es: [2.0, 4.0, 6.0]

Process finished with exit code 0
```

Listing 17: Inputs para operación con vectores

Como se puede ver siempre se sigue la metodología de primero insertar el tipo de operación, y a continuación insertar los datos que se solicitan.

Por otro lado para usar el cliente de java se han de levantar primero el servidor algebraico, a continuación el servidor, y por último (en mi caso en netbeans) ejecutar el cliente en java. De ese modo podemos usar el set de operaciones básicas en Java.

Para las operaciones algebraicas se expone un ejemplo en su propio apartado, por ello se acaba el manual enseñando el ejemplo de qué pasaría si levantamos los servidores y el cliente, y por algun motivo se caen los servidores:

```
Hacemos ping al server...
Hacemos ping al server algebraico...
1. Suma
2. Resta
3. Multiplicaci n
4. Divisi n
5. Seno (grad)
6. Coseno (grad)
7. Tangente (grad)
8. Conversi n Grados-Radianes
9. Conversi n Radianes-Grados
10. Algoritmo de Euclides (a, b)
11. Algoritmo Extendido de Euclides (a, b)
12. Congruencia Lineal (a*x cong b (mod m)
13. Ecuaci n diof ntica ( a*x + b*y = c)
```

```
    14. Suma de los vectores (v1 + v2)
    15. Producto escalar de los vectores (v1 . v2)
    16. Producto cruz de los vectores (v1 x v2)
    Introduce una de las operaciones presentadas:
    Operaci n: 5
    N mero: 90
    Error de transporte: TSocket read 0 bytes

    Process finished with exit code 0
```

Listing 18: Ejemplo excepción de transporte