UNIVERSIDAD NEBRIJA

ALUMNO:

Asignatura: Programación de Sistemas Distribuidos

Curso: 2021/2022 Fecha: 15-02-2022
Semestre: 2º

PRÁCTICA 1: Aplicación usando CORBA

Para poder realizar y ejecutar este programa necesitaremos un editor como Atom y el JDK de Java, que lo puedes descargar de http://www.java.com/es/

Para entregar la práctica hay que subir por un lado este doc en pdf y por otro lado 2 zip: HolaMundo.zip y Practica2.zip. En el comentario de la entrega de la práctica habrá que hacer la referencia del repositorio de Github.

1. **Vamos a hacer un Hola Mundo en CORBA (2 punto)**

   La aplicación contendrá un archivo IDL, un archivo servidor y uno de cliente. Todas las instrucciones de la aplicación deben estar comentadas en castellano, con nuestras palabras para argumentar que se entiende.

   Compilaremos primero el IDL, luego el servidor y luego el cliente usando los siguientes códigos: respectivamente:

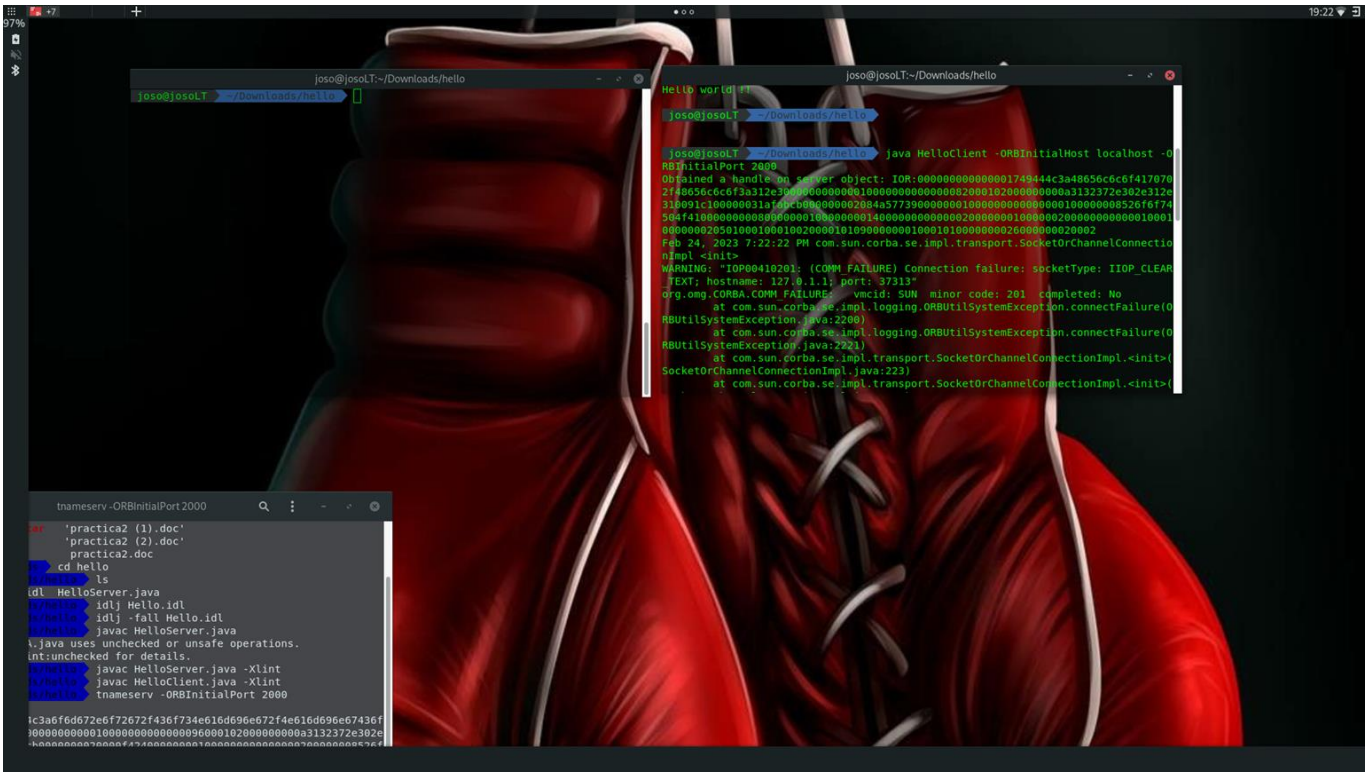
   $ idlj -fall count.idl
   $ javac Server.java
   $ javac Client.java

   Para ejecutar el programa necesitamos tener abiertas tres ventanas del Símbolo del sistema. La primera iniciará el puerto, la segunda ejecutará el servidor y la tercera el cliente. El código para ejecutarla es, respectivamente:

   $ tnameserv -ORBInitialPort 2000
   $ java Server -ORBInitialHost localhost -ORBInitialPort 2000
   $ java Client -ORBInitialHost localhost -ORBInitialPort 2000

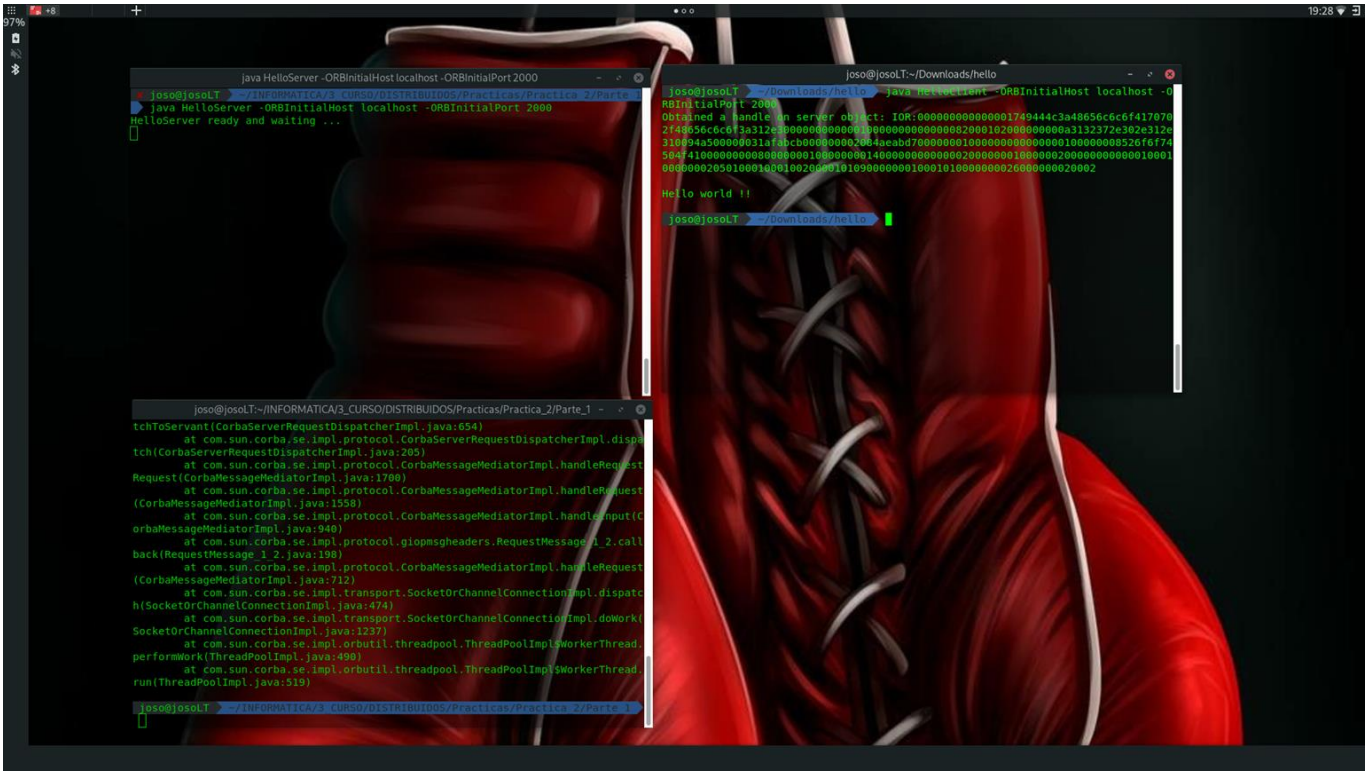
2. **Preguntas sobre Hola Mundo en CORBA (puedes añadir capturas) (2 puntos):**

2.a. **¿Qué sucede si lanzo antes el cliente que el servidor?**



Salta un error, ya que al intentar conectarse a un servidor que no existe la comunicación falla y se cierra el cliente. (Terminal de la derecha)

2.b. **¿Qué sucedería si lanzase varios servidores a la vez y un solo cliente?**



Se conecta al srvidor que ha sido lanzado mas tarde.

2.c. **¿Puedes conectarte al servidor de un compañero? ¿Cómo lo harías?**

Indicandole la dirección IP y puerto en la linea de comando del cliente, siempre y cuando estemos en la misma red local

java HelloClient -ORBInitialHost {direccion ip compañero} -ORBInitialPort {puerto compañero}

3. **Actualiza un repositorio de Github con una aplicación Java CORBA (7 puntos)**
   Aquí debéis hacer un fork de una aplicación en Github y realizar modificaciones en ella. Por ejemplo, imaginemos que tenemos una calculadora que funciona con CORBA y únicamente tiene las funciones de suma, resta, multiplicar y dividir. Podéis añadir por ejemplo: operar con raíces cuadradas o añadir que utilice decimales. Pautas:
   - **El código que se añada debe ser por un lado pegado en este documento y por otro lado, se deben realizar los commits en el repositorio.**
   - **El código debe contener comentarios propios respecto a como funciona la aplicación.**
   - **Toda la información que tenga el README.md nunca está demás.**
   - **Intenta que sea una aplicación/funcionalidad diferente la que modificas (3 puntos) No todos los compañeros vamos a tener Calculadoras, busca otras aplicaciones y diferénciate.**

Se ha creado un servidor en Java y un cliente en Python se ha tomado como base el repositorio anterior de hello world, pero se han implementado funciones de calculo, las cuales se muestran a continuación.

```
long suma(in long a, in long b);
long restar(in long a, in long b);
long multiplicar(in long a, in long b);
long dividir(in long a, in long b);
long factorial(in long a);
long potencia(in long a, in long b);
long logaritmo(in long a);
long raiz(in long a);
string derivar(in long a, in arr b);
```

Para la mayoría de funciones sólo se les pasan uno o dos parámetros de tipo entero, con los cuales se operará, para la última función: derivada, se pretende hacer un ejemplo sencillo de la derivada de un polinomido de cierto grado, se le proporcionará el grado del polinomio sencillo y un array que corresponderá a los coeficientes de cada término, en orden ascendente con respecto a su grado.

Para poder comunicar estos dos programas se ha de hacer uso de versiones específicas de Java y OmniORBpy, estas se proporcionan en forma de archivo comprimido con los binarios necesarios .
En primer lugar, necesitaremos iniciar el puerto con el comando del ejercicio anterior.

tnameserv -ORBInitialPort 2000

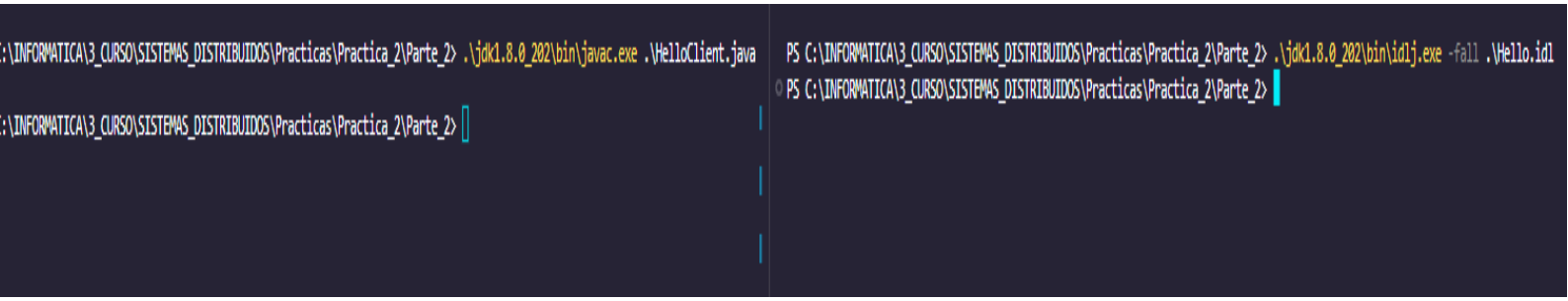
```
PS C:\INFORMATICA\3_CURSO\SISTEMAS_DISTRIBUIDOS> tnameserv -ORBInitialPort 2000
Contexto de Nomenclatura Inicial:
IOR:000000000000002b49444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e746578744578743a312e3000000000000100000000000000e000102000000000f3136392e3235342e36312e323231000007d0000000000045afabcb0000000020000f4
6f6f74504f4100000000d544e616d65536572766963650000000000000000800000001000000011400000000000002000000010000002000000000000100010000000205010001000100200001010900000001000101000000002600000002000200
TransientNameServer: definiendo puerto para referencias a objeto iniciales en: 2000
Listo.
```

A continuación compilaremos el servidor escrito en Java con los binarios de compilación proporcionados

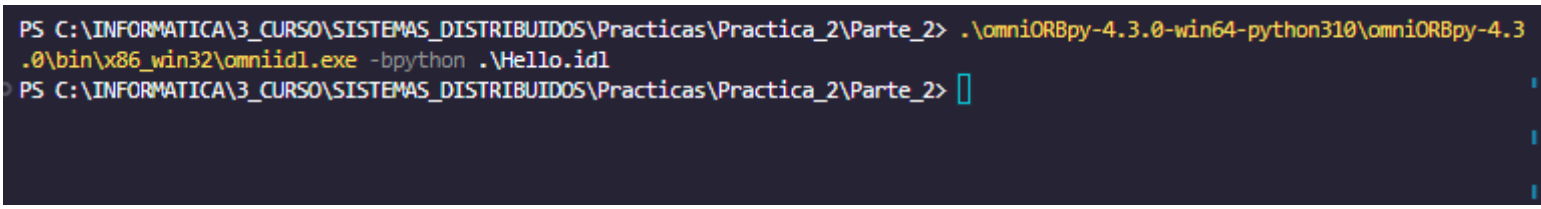
.\jdk1.8.0_202\bin\javac.exe .\HelloClient.java

Posteriormente se compilará el idl tanto en Java como en Python para que ambos programas puedan sincronizar sus objetos comunes con los binarios proporcionados.

.\jdk1.8.0_202\bin\idlj.exe -fall .\Hello.idl



.\omniORBpy-4.3.0-win64-python310\omniORBpy-4.3.0\bin\x86_win32\omniidl.exe -bpython .\Hello.idl



Se ejecutará el servidor en java con el binario proporcionado:

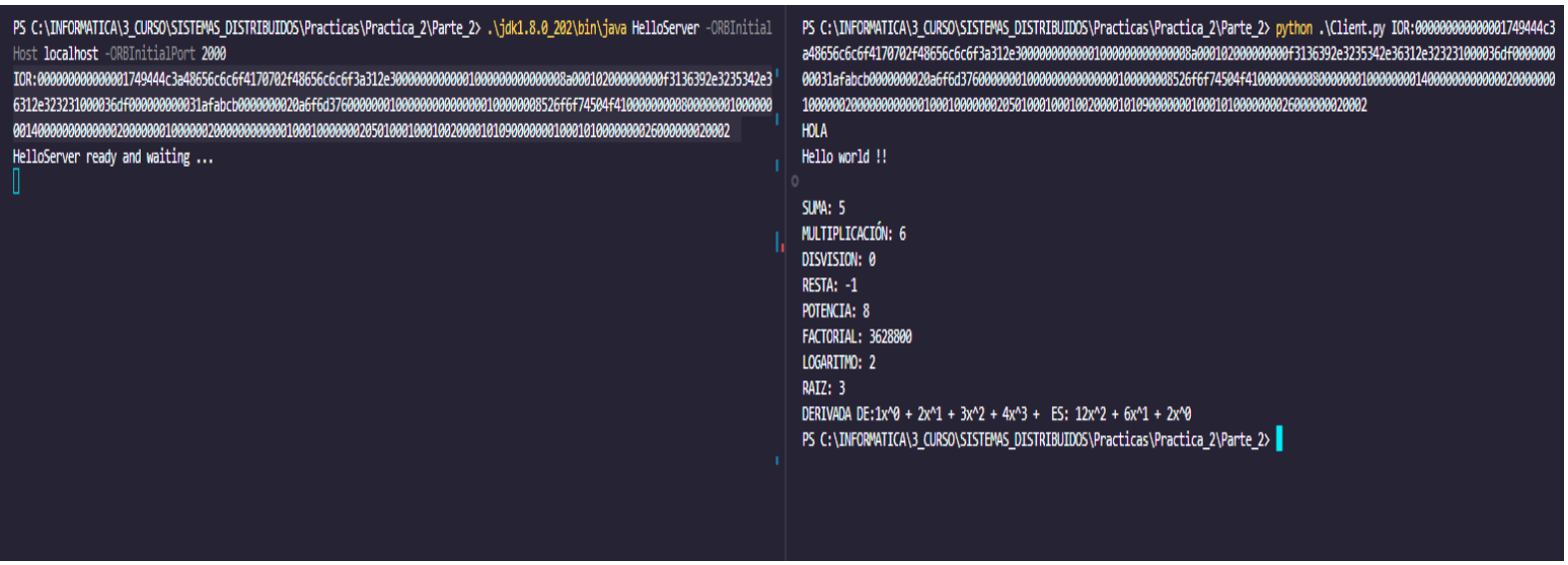
.\jdk1.8.0_202\bin\java HelloServer -ORBInitialHost localhost -ORBInitialPort 2000

Y a continuación, se nos mostrará por consola un objeto de referencia IOR, el cual es necesario para poder establecer la conexión del cliente, la salida deberá ser similar a esto, cambiando el número del IOR.

IOR:000000000000001749444c3a48656c6c6f4170702f48656c6c6f3a312e3000000000000010000000000000008a000102000000000f3136392e3235342e36312e3232310000c8ed000000000031afabcb0000000020a6e3f2630000000100000000000000010000000852 6f6f74504f4100000000080000000100000000140000000000000020000000100000020000000000001000100000002050100010001002000010109000000010001010000000026000000020002
HelloServer ready and waiting ...

Por último, se ejecutará el cliente con Python proporcionándole como argumento el objeto IOR previo.

python .\Client.py
IOR:000000000000001749444c3a48656c6c6f4170702f48656c6c6f3a312e3000000000000010000000000000008a000102000000000f3136392e3235342e36312e3232310000c8ed000000000031afabcb0000000020a6e3f2630000000100000000000000010000000852 6f6f74504f4100000000080000000100000000140000000000000020000000100000020000000000001000100000002050100010001002000010109000000010001010000000026000000020002



Se proporcionan capturas de pantalla de todo este proceso.

Repo original https://github.com/KudrinMatvey/java-distributed-programming/tree/f7924df02337c9cf6d5339fb8dcc859364ecad61/CORBA

UNIVERSIDAD
NEBRIJA