Marc Ugas

Práctica 3: RMI

Desarrollo de Sistemas Distribuidos

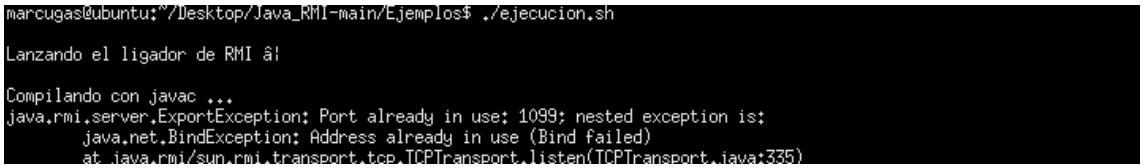
Ejercicio 1:

Ejemplo 1.

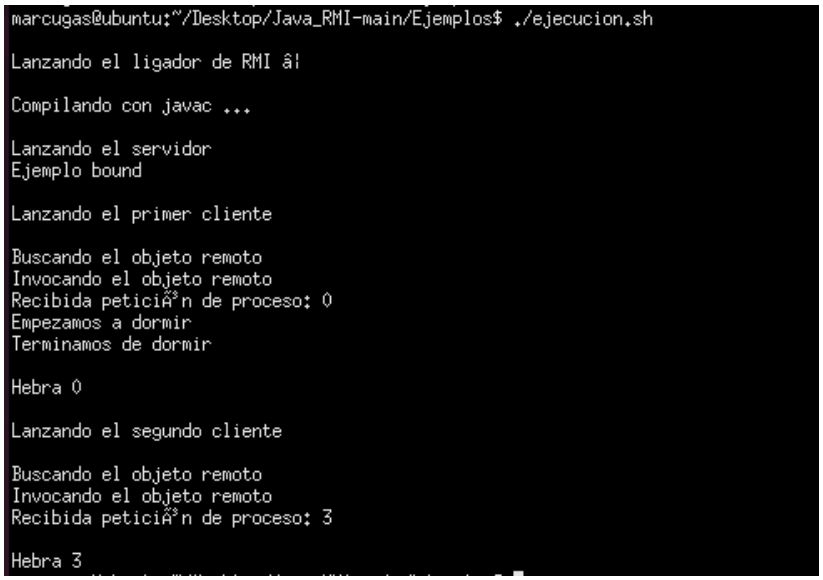
He experimentado algún problema: al ejecutar el fichero *ejecucion.sh* me salía que el puerto 1099 estaba ocupado, usando los siguientes comandos, se soluciona el problema:

&lsof -i :1099 #identificador del proceso

&kill PID

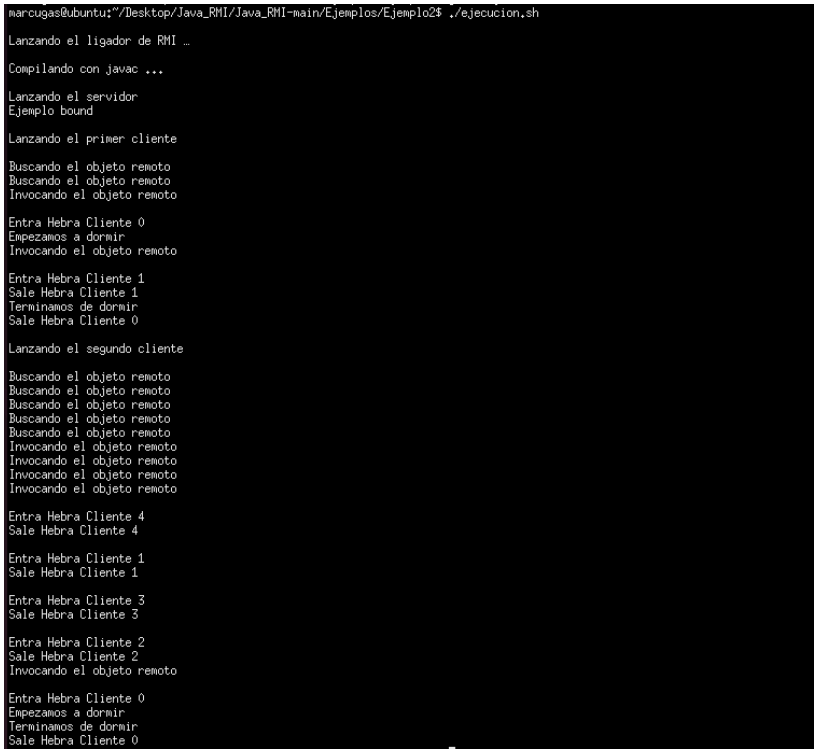


Al ejecutar el programa correctamente, nos da la salida siguiente:



Ejemplo 2.

Para este caso, he puesto 2 hebras para el primer cliente y 5 hebras para el segundo, como se puede apreciar en la imagen. Para las hebras cuyo nombre acaban en 0, cuando esta entra, el servidor no atiende ninguna otra hasta que salga, por lo tanto cuando una hebra acabada en 0 duerme, el servidor también duerme hasta que termine.



Ejemplo 3.

Para este ejemplo la ejecución del código da la siguiente salida:

Marc Ugas

Ejercicio 2.

Para hacer este ejercicio, he usado la base del ejemplo anterior. Aunque he añadido un archivo llamado *contador_I*, este es el encargado de contener las funciones para el funcionamiento sincronizado de los servidores. Para el archivo de *servidor.java*, también he hecho una pequeña modificación, como se ve en la siguiente, en que se inicializan la cantidad de servidores que vayamos a utilizar. Como vemos es más sencillo, que crear un archivo para cada servidor, de esta manera es más escalable.

```
try {
    //creamos las dos instancias contador (servidores)
    icontador contador1 = new contador(id1);
    icontador contador2 = new contador(id2);

    // Crea un registro RMI en el puerto 1099
    Registry reg = LocateRegistry.createRegistry(1099);
    // Crea una instancia de contador
    Registry registry = LocateRegistry.getRegistry();

    // Vincula la instancia al registro RMI con un nombre
    registry.rebind(id1,contador1);
    registry.rebind(id2,contador2);

    //Se vincula tambien cada instancia contador con el otro servidor
    contador_I rep1 = (contador_I) contador1, rep2 = (contador_I) contador2;
    rep1.setreplica(id2);
    rep2.setreplica(id1);
```

Interfaz de contador_I:

Vemos que contiene la definición de las funciones que son necesarias entre ambos servidores.

```
1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3
4 public interface contador_I extends Remote {
5         public String getID() throws RemoteException;
6         public boolean existeCliente(String dni) throws RemoteException;
7         public void setreplica(String id) throws RemoteException;
8 }
```

Interfaz de icontador:

Contiene la definición de las funciones que usa el cliente.

```
1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3 import java.rmi.registry.Registry;
4 import java.rmi.NotBoundException;
5
6 public interface icontador extends Remote {
7     public int InicioSesion(String dni, Registry r) throws RemoteException, NotBoundException;
8     public int registrar(String dni, Registry r) throws RemoteException, NotBoundException;
9     public void imprimir() throws RemoteException;
0     public void donar(String dni, int valor) throws RemoteException;
```

Marc Ugas

Interfaz de contador:

Implementación de las funciones y inicialización de los contadores para su uso en el cliente y en el servidor.

```
12 public class contador extends UnicastRemoteObject implements icontador, contador_I {
13     private int suma;
14     private int indice;
15     Map<String,Integer> clientes;
16     private String ID;
17     Registry reg;
18     contador_I replica;
19
20
21
22     //Funcion inicializadora
23     public contador(String id) throws RemoteException {
24         this.clientes = new HashMap<>();
25         this.suma = 0;
26         this.indice = 0;
27         this.ID = id;
28         this.reg = LocateRegistry.getRegistry();
29     }
```

Para la interfaz de usuario (*cliente.java*), he creado un menú desplegable, como se ve en la imagen del programa, es obligatorio el inicio de sesión y/o registro para poder hacer las donaciones.

Para poder ejecutar los archivos, se realiza lo siguiente:

- Compilación archivos: ejecutar *compilar.sh*.
- Matar proceso de compilación: he añadido una función *matarproceso.sh*, que tiene de entrada un puerto y elimina el proceso que se está realizando (1099).
- Ejecutar el servidor: *exe_servidor.sh*.
- Ejecutar el cliente: *exe_cliente.sh*.

Si todo está correcto la interfaz gráfica será como la de la figura siguiente, se puede compilar el cliente en una o mas ventanas. Podemos ver que se cumple los requisitos de las donaciones. Para ver la lista de clientes, seleccionar opción Mostrar lista.

```
marcugas@ubuntu:~/Desktop/Practica3/RMIReplica$ ./exe_cliente.sh

Lanzando el cliente
Bienvenido!
1. Iniciar Sesion
2. Registrarse
3. Salir
2
Introduzca su dni:
1235432
Cliente 1235432 aÃ±adido con exito!
Elige una opcion:
1. Donar
2. Mostrar lista
3. Salir
1
Introduzca cantidad a donar:
100
El cliente 1235432 ha donado 100$
Elige una opcion:
1. Donar
2. Mostrar lista
3. Salir
1
Introduzca cantidad a donar:
463
El cliente 1235432 ha donado 463$
Elige una opcion:
1. Donar
2. Mostrar lista
3. Salir
2
Cliente 1: 1235432Cantidad donada: 563
```