DSD P3

PABLO BLANCO LOPEZ

Mayo 2024

1. Introducción

La práctica 3 de Desarrollo de Sistemas Distribuidos consiste en el uso de RMI (Remote Method Invocation) de Java para implementar varios ejemplos que nos proporcionan, y además un ejercicio que consta de un sistema de donaciones con servidores replicados.

2. Explicación de los ejemplos

Para la realización de la parte 1 de la práctica relacionada con ejecutar los tres ejemplos, como se aconseja en el guión, decidí crear un script para ejecutarlo, aunque tuve que modificar ciertos aspectos del script para los diferentes ejemplos. Para empezar, en el primero se hacen uso de dos clientes diferentes, pero a partir del segundo se utilizan hebras dentro de un mismo cliente lo que hace que quitara uno de los clientes que anteriormente tuve que añadir en el script.

El uso de los clientes va acompañado de diferentes parámetros para los distintos ejemplos. En el primer ejemplo, el cliente necesita los parámetros de dirección del servidor y el identificador del cliente, en el segundo ejemplo necesita la dirección del servidor y el número de hebras a lanzar, y por último, en el tercer ejemplo no necesita nada como parámetro.

A continuación voy a mostrar el primer script que he usado, pero como he dicho anteriormente estos scripts varían en cada ejemplo.

He de decir que el ejemplo 3 me daba ciertos problemas, porque para ejecutar todos estos ejemplos hay que tener un registro RMI lanzado en el tu sistema, yo lo lanzo con la orden *$ rmiregistry &*, que lanza el registro y lo deja en segundo plano, pues el programa, en concreto la clase servidor, crea de nuevo este registro en el mismo puerto que lo tenía yo creado como anteriormente he dicho. Para solucionar esto he comentado la línea 17 de dicho archivo y todo se solucionó y funciona correctamente.

```
#!/bin/sh -e

echo
echo "Compilando con javac ..."
javac *.java

sleep 2
echo
echo "Lanzando el servidor"
java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo &

sleep 2
echo
echo "Lanzando el primer cliente"
echo
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0

sleep 2
echo
echo "Lanzando el segundo cliente"
echo
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 3
```

A continuación voy a adjuntar capturas de la orden de salida de dichos ejemplos para comprobar su funcionamiento, así como comentar ciertos aspectos de ellos.

```
Compilando con javac ...

Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente

Buscando el objeto remoto
Invocando el objeto remoto
Recibida peticion de proceso: 0
Empezamos a dormir
Terminamos de dormir

Hebra 0

Lanzando el segundo cliente

Buscando el objeto remoto
Invocando el objeto remoto
Recibida peticion de proceso: 3

Hebra 3
```

Figura 1: Ejemplo 1

Aquí lo que ocurre es que una vez lanzado el servidor, el cliente busca el objeto remoto, lo invoca, y es el servidor el que recibe la petición del cliente. El primer cliente tiene el identificador 0, y en el servidor está configurado que al recibir el identificador 0 duerma durante cinco segundos, y una vez hecho esto que saque por pantalla: Hebra + número de hebra. Lo que ocurre con el segundo cliente, al tener el identificador 3, imprime directamente Hebra 3.

En el ejemplo 2, el número de hebras viene descrito por el segundo parámetro del cliente, he puesto 5 hebtras para que se vea clara la salida. Las hebras se van ejecutando de forma asíncrona, y si algunas de estas hebras, cada una con un identificador, termina dicho identificador en 0, esta hebra dormirá 5 segundos inmediatamente después que salga por pantalla 'Empezamos a dormir'. Hay una forma para que esto sea síncrono, y es poner en la implementación del servidor,

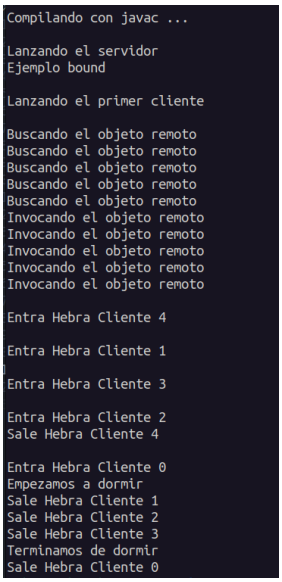

Figura 2: Ejemplo 2

en el método escribir_mensaje, synchronized, para que una vez que empiece a ejecutarse este método, termine y de espacio a otra hebra para su ejecución. Adjunto en la figura 3 su diferente comportamiento.

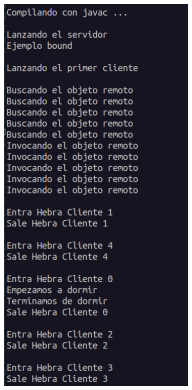

Figura 3: Ejemplo 2 con synchronized

En este ejemplo, se implementa una clase contador aparte, y este objeto remoto se invoca en la implementación del servidor. He cambiado el tamaño del bucle del cliente de 1000 a 10000 para que tarde más tiempo en ejecutar todas las invocaciones.

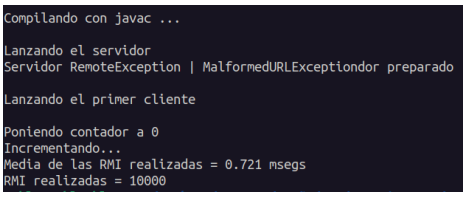

Figura 4: Ejemplo 4

3. Realización del ejercicio

Este ejercicio consiste en la implementación de un sistema de donaciones en base a una arquitectura Cliente/Servidor, pero en vez de con un único servidor, con más de uno y replicados, por lo que el cliente podrá acceder a cuál sea de estos servidores replicados y tendrá la misma funcionalidad que un servidor central.

He tenido que buscar ciertas soluciones para guardar una tupla de donantes con sus nombres y sus cantidades donadas respectivamente para ver si podían acceder a ciertos métodos tal y como se plantea en la propuesta del ejercicio. Al principio cree mi propia clase llamada TuplaDonante con dos atributos, nombre y cantidadDonada, pero esto me daba ciertos problemas. Al investigar un poco sobre los tipos de estructuras de datos en java, comprobé que existía una estructura llamada Map que me contenía la información tal y como yo deseaba, en el Map tienes una clavem única, que en este caso sería nuestro nombre, y un valor asociado a dicha clave, que en mi caso es la cantidad donada. Con la implementación de esto, todos mis problemas para almacenar la tupla donante y cantidad donada se solucionaban.

En cuanto a las normas de la entrega, el ejercicio funciona sin problema al ejecutar ambos scripts, 'server.sh' y 'client.sh', uno para lanzar los servidores y otro para lanzar el cliente, respectivamente. Además he añadido más operaciones aparte de las pedidas, como pueden ser eliminar un donante y consultar el top 5 de donantes de todo el sistema.

Para asegurar que se puedan ejecutar varios clientes a la vez, he precisado de añadir un algoritmo de exclusión mutua entre servidores distribuidos. He optado por elegir uno que hemos visto en teoría, el algoritmo de exclusión mutua basado en relojes lógicos, o también llamado algoritmo de Ricart-Agrawala. Si ve la figura 5, el primer método es para solicitar la entrada a la sección crítica, lo que se hace es llamar a al método 'recibirSolicitudSeccionCritica' de todas las réplicas excepto la misma que llama al método para entrar a la sección crítica. El método para recibir la solicitud, devuelve false si se encuentra en una sección

crítica, o si la réplica que lo ejecuta está solicitando entrar a una sección crítica y el tiempo recibido por parámetro en el método es mayor que el tiempo de la réplica que solicita otra sección crítica. En otro caso devuelve true. Por lo que volviendo a la solicitud para entrar dentro de una sección, si alguna réplica devuelve false, inmediatamente se devuelve false y dicha réplica no puede acceder a la sección crítica, en caso contrario de que acabe el bucle y ninguna devuelva false, se devuelve true y se puede acceder a la sección crítica, por lo que cambia el estado de dicha réplica a 'EN_SECCION_CRITICA'.

```
308     @Override
309     public boolean solicitarSeccionCritica() throws RemoteException {
310         this.estado = Estado.SOLICITANDO;
311         long tiempo = obtenerTiempoTranscurrido();
312         boolean resultado = true;
313         for (int i=0; i<replicas.size(); i++) {
314             if (i == this.id) continue;
315             resultado = replicas.get(i).recibirSolicitudSeccionCritica(tiempo, this.id);
316             if (!resultado) {
317                 return false;
318             }
319         }
320         this.estado = Estado.EN_SECCION_CRITICA;
321         return true;
322     }
323 
324     @Override
325     public boolean recibirSolicitudSeccionCritica(long tiempoRecibido, int idRecibido) {
326 
327         long tiempo = obtenerTiempoTranscurrido();
328         if (this.estado == Estado.EN_SECCION_CRITICA || (this.estado == Estado.SOLICITANDO && tiempo < tiempoRecibido)) {
329             return false;
330         } else {
331             return true;
332         }
333     }
334 
335     @Override
336     public void liberarSeccionCritica() throws RemoteException {
337         this.estado = Estado.LIBRE;
338     }
339 
```

Figura 5: Algoritmo de Ricart-Agrawala

He utilizado dicho algoritmo en todos los métodos, ya que al no estar centralizado, es necesario parar el resto de réplicas para que no añadan más valores a las listas porque sino la fiabilidad de cualquier operación no se podría asegurar.

3.1. Funcionamiento

En cúanto al funcionamiento, para lanzar el servidor, basta con ejecutar el script 'server.sh', este te pedirá que ingrese un número de réplicas, y acto seguido se lanzan todas y cada una de ellas con nombres dentro del registro 'Replica[0..N]'. Una vez quiera parar todas las réplicas, basta con que le de al enter para que borre el proceso demonio y así liberar el registro RMI.

Para lanzar el cliente debe ejecutar el script 'client.sh' que de nuevo pedirá el número de réplicas, en este caso asegúrese de poner el mismo número, o al menos menor, que el número de réplicas del servidor. Si observa la figura 6 podrá darse cuenta de como está configurado el menú del cliente. Lo voy a comentar

paso a paso:

1. Elegir Réplica. Aquí puede elegir el servidor al que quiere conectarse.
2. Registrar donante. Aquí puede registrar un donante, si lo crea, pone por pantalla 'Donante registrado', y si no lo crea, 'Donante no registrado, ya existe'.
3. Eliminar donante. Aquí puede eliminar un donante que ya existe, por lo que si realiza de forma exitosa el borrado del donante pone por pantalla 'Donante eliminado', en caso contrario, 'Donante no eliminado, no se ha encontrado'.
4. Donar. Aquí deberá introducir el nombre del donante que quiere donar, y la cantidad que desea donar, si el donante es válido, pone por pantalla 'Donación realizada', en caso contrario, 'Donación no realizada, no se ha encontrado al donante'.
5. Ver total donado. Aquí deberá introducir un nombre de donante que se encuentre registrado y que haya donado algo, si no introduce estos criterios se pone por pantalla 'No tienes permisos para ver el total donado, recuerda registrarte primero y donar algo'. En caso contario pone el total donado.
6. Ver lista donantes. Aquí pasa igual que anteriormente, deberá introducir el nombre de un donante que haya realizado alguna donación, en caso de que no, pone por pantalla lo mismo que arriba, y en caso contrario pone la lista de donantes al servidor que pertenecen.
7. Ver top 5 donantes. Aquí pasa igual que los otros dos para el tema de permisos, y en caso de que tenga permisos para ver dicha lista, pone por pantalla una lista ordenada de mayor a menor donado de los mejores cinco donantes.
8. Salir. Aquí se sale del programa.

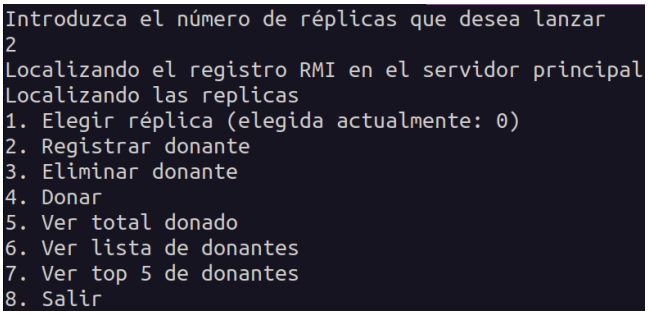

Figura 6: Menú del cliente