Práctica 3:

Multienvío no ordenado en java RMI

Alejandro Espinosa Álvarez
Jose Ángel Pardo Cerdán
Guillermo García Molina

21/11/2014

Índice de contenidos

1. Introducción 6
2. Explicaciones y comprobación del código 6
   2.1. receiveGroupMessage 6
   2.2. DepositeMessage 7
   2.3. SendGroupMessage 7
   2.4. SendingMessage -> run() 8
3. Cuestiones 9
   3.1. Cuestión 1 9
   3.2. Cuestión 2 9
   3.3. Cuestión 3 11
   3.4. Cuestión 4 12

Índice de ilustraciones

Ilustración 1: Código del método receiveGroupMessage 5
Ilustración 2: Código del método DepositeMessage 6
Ilustración 3: Código del método sendGroupMessage 6
Ilustración 4: Código del método run() de la clase SendingMessage 7
Ilustración 5: Ejemplo miembros de un grupo del servidor. 8
Ilustración 6: Ejemplo de envío de 2 mensajes 9
Ilustración 7: Ejemplo de recibo de mensajes 9
Ilustración 8: Ejemplo de envío de mensaje y borrado de miembro 10
Ilustración 9: Ejemplo de recibo de mensaje después del borrado del miembro del grupo 11
Ilustración 10: Ejemplo de recibo de un mensaje de un grupo que no existe 11

1. Introducción

En esta práctica vamos a ampliar la funcionalidad que conseguimos en la anterior para implementar un multienvío (no necesariamente ordenado) de mensajes entre los miembros de un grupo cerrado.

Para ello cada cliente implementará una cola local de mensajes, donde se depositarán los mensajes a él destinados. Los envíos se realizarán mediante un thread separado, para permitir los envíos simultáneos dentro de un mismo grupo. Asimismo, mientras haya envíos en curso habrá que inhibir las altas y las bajas del grupo, para evitar inconsistencias.

Los clientes ahora implementarán callbacks para admitir los mensajes procedentes de los grupos a los que pertenecen. Para ello será necesario crear un registro local (cada cliente en su máquina local), y darlos de alta en ellos con un nombre local.

2. Explicaciones y comprobación del código

En esta parte de la memoria explicaremos algunos aspectos del código, a nuestro parecer, más importantes, y mostraremos los resultados obtenidos tras realizar la práctica.

2.1. receiveGroupMessage

```
@Override
public synchronized ArrayList<Byte> receiveGroupMessage(String galias) throws RemoteException {
    lock.lock();
    int id = -1;

    try {
        id = stub.findGroup(galias);
        if( id == -1){ return null; }

        while(true){
            for (int i = 0; i < cola.size(); i++) {
                if(cola.get(i).emisor.idgroup == id){
                    ArrayList<Byte> array =(ArrayList<Byte>) cola.get(i).mensaje.clone();
                    cola.remove(i);
                    return array;
                }
            }
            //Bloqueo porque no tiene ningún mensaje de ese grupo
            try{
                condicion.await();
            } catch (InterruptedException ex) {
            Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
            }

        }//Fin while
    } finally {
        lock.unlock();
    }
}//Fin metodo
```

Ilustración 1: Código del método receiveGroupMessage

Este método se utiliza para recoger de la cola local el siguiente mensaje correspondiente al grupo del alias indicado.

Podemos ver el bloqueo que realizamos si no hay mensajes en la cola. Cuando leemos un mensaje de la cola del grupo elegido, lo borramos de la cola.

2.2. DepositeMessage

```
@Override
public void DepositMessage(GroupMessage m) throws RemoteException {
    lock.lock();
    try {
        cola.push(m);

        //Desbloqueo condicion, porque ha llegado un mensaje
        condicion.signalAll();

    } finally {
        lock.unlock();
    }
}
```

Ilustración 2: Código del método DepositeMessage

Tal y como nos explica la práctica, este método es un callback (método invocado desde el servidor), para depositar el mensaje en la cola local del cliente.

Este método garantiza la exclusión mutua en las operaciones.

2.3. SendGroupMessage

```
@Override
public boolean sendGroupMessage(GroupMember gm, ArrayList<Byte> msg) throws RemoteException {
    GroupMessage mensaje;
    for (int i = 0; i < GroupList.size(); i++) {
        if(GroupList.get(i).isMember(gm.alias) != null){

        mensaje = new GroupMessage(gm,msg);
        GroupList.get(i).Sending();
        SendingMessage hilo = new SendingMessage(mensaje, GroupList.get(i));
        hilo.start();

        //Se termina de enviar los mensajes de grupo
        return true;
        }
    }
    return false;
}
```

Ilustración 3: Código del método sendGroupMessage

Con este método se pretende hacer el multienvío del mensaje por parte del miembro de grupo que le pasamos por parámetro al grupo que pertenece.

Podemos ver que se crea un hilo de la clase SendingMessage que se ejecutará con start y que llamará al método run() de esta clase, enviando así el mensaje.

Cuando ha terminado de enviar el mensaje a todos los miembros del grupo en el que se encuentra, el método devuelve true.

2.4. SendingMessage -> run()

```
@Override
public void run(){
    Registry registry = null;

    try {
        registry = LocateRegistry.getRegistry(1099);
    } catch (RemoteException ex) {
        Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
    }

        //A continuación envio el mensaje a los miembros del grupo, menos al emisor
        for (int i = 0; i < grupo.members.size(); i++) {
            //Compruebo que no sea el emisor
            if(grupo.members.get(i) != mensaje.emisor){
                try {
                    cliente = (ClientInterface) registry.lookup( grupo.members.get(i).alias);
                try {

                     Thread.sleep((int)(Math.random()*30+30)*1000);
                } catch (InterruptedException ex) {
                Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
                }
                    cliente.DepositMessage(mensaje);
                }catch ( RemoteException | NotBoundException  ex) {
                Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    grupo.EndSending(mensaje.emisor);
}
```

Ilustración 4: Código del método run() de la clase SendingMessage

En este método es en el que se realizan los envíos, invocando el método DepositeMessage de los destinatarios.

Podemos ver en el código del método que buscamos cuál de los miembros del grupo es el emisor para no enviarle el mensaje. Utilizamos el método EndSending para finalizar el envío al grupo.

Como en esta versión no contemplamos un envío fiable u ordenado, si falla algún envío no lo tenemos en cuenta.

3. Cuestiones

3.1. Cuestión 1

¿Es bloqueante o potencialmente bloqueante la invocación a sendGroupMessage? Indica las posibles causas, si las hay, de bloqueo potencial.

Los bloqueos de un send se refieren a la espera de este sobre la copia de datos al área del kernel o sobre la comunicación completa.

En nuestro caso es potencialmente bloqueante, ya que no siempre se bloquea para la comunicación completa con los hilos.

3.2. Cuestión 2

Incluir dentro de SendingMessage, justo antes de invocar a DepositMessage para cada objeto miembro un delay aleatorio (entre 30 y 60 segundos), para comprobar que mensajes diferentes procedentes de un mismo emisor pueden llegar en orden distinto a los difrentes miembros del grupo.

```
Tread.sleep((int)(Math.random()*30+30)*1000);
```

Con esta línea genero números aleatorios entre 30 y 60. Y estos son los miembros del grupo que tengo:

```
        --------------------MIEMBROS DEL GRUPO--------------------

Inserte id de grupo: 0
Alias: valen Hostname: local
Alias: alejandro Hostname: local
Alias: blanca Hostname: local
```

Ilustración 5: Ejemplo miembros de un grupo del servidor.

Voy a enviar dos mensajes, primero envío mensaje uno y después mensaje dos.

```
Inserte id de grupo a enviar el mensaje: 0

Inserte un mensaje: mensaje uno
        ----------------------------------------------------
Elija opción:
1. Crear grupo
2. Eliminar grupo
3. Añadir miembro al grupo
4. Eliminar miembro del grupo
5. Bloquear altas/bajas
6. Desbloquear altas/bajas
7. Número de grupos en servidor
8. Miembros en grupo a indicar
9. Enviar mensaje
10. Recibir mensaje
11. Terminar ejecución
9
        --------------------ENVIO MENSAJE--------------------

Inserte id de grupo a enviar el mensaje: 0

Inserte un mensaje: mensaje dos
        ----------------------------------------------------
```

Ilustración 6: Ejemplo de envío de 2 mensajes

Y después recibimos mensaje en un cliente:

```
        --------------------RECIBO MENSAJE--------------------

Inserte alias de grupo, para recibir mensaje de este: grupo0
El mensaje obtenido es:
mensaje dos
        ----------------------------------------------------
Elija opción:
1. Crear grupo
2. Eliminar grupo
3. Añadir miembro al grupo
4. Eliminar miembro del grupo
5. Bloquear altas/bajas
6. Desbloquear altas/bajas
7. Número de grupos en servidor
8. Miembros en grupo a indicar
9. Enviar mensaje
10. Recibir mensaje
11. Terminar ejecución
10
        --------------------RECIBO MENSAJE--------------------

Inserte alias de grupo, para recibir mensaje de este: grupo0
El mensaje obtenido es:
mensaje uno
        ----------------------------------------------------
```

Ilustración 7: Ejemplo de recibo de mensajes

En este cliente, tras esperar unos largos segundos nos encontramos con que el mensaje dos llega antes que el primero.

3.3. Cuestión 3

¿Qué ocurre si un cliente intenta recoger un mensaje de un grupo al cual ya no pertenece? ¿Y si lo intenta de un grupo existente al que no ha pertenecido nunca?

Si un cliente intenta recoger un mensaje de un grupo al cual ha pertenecido pero ya no pertenece no tendrá ningún problema en hacerlo, ya que si ha recibido el mensaje mientras todavía era miembro, el mensaje se almacena en la cola local del cliente, por lo que podrá recogerlo y leerlo.

En cambio, no podrá recoger un mensaje de un grupo existente en el servidor pero al cual no ha pertenecido, ya que son grupos cerrados y el envío de mensajes se hace a los miembros del grupo, por lo que no estará en su cola.

Para probar esta cuestión tomamos como ejemplo la cuestión 2, tenemos un grupo llamado grupo0 con miembros valen, blanca y alejandro. Valen manda un mensaje al grupo, y más tarde borra a blanca del grupo

```
        --------------------ENVIO MENSAJE--------------------

Inserte id de grupo a enviar el mensaje: 0

Inserte un mensaje: mensaje
        -------------------------------------------------
Elija opción:
1. Crear grupo
2. Eliminar grupo
3. Añadir miembro al grupo
4. Eliminar miembro del grupo
5. Bloquear altas/bajas
6. Desbloquear altas/bajas
7. Número de grupos en servidor
8. Miembros en grupo a indicar
9. Enviar mensaje
10. Recibir mensaje
11. Terminar ejecución
4
        --------------------BORRAR MIEMBRO--------------------

Inserte alias de miembro nuevo: blanca

Inserte id de grupo: 0
Miembro eliminado
        -------------------------------------------------
```

Ilustración 8: Ejemplo de envío de mensaje y borrado de miembro

Blanca después de que haya sido borrada lee la cola de mensajes y puede acceder a leer el mensaje:

```
Elija opción:
1. Crear grupo
2. Eliminar grupo
3. Añadir miembro al grupo
4. Eliminar miembro del grupo
5. Bloquear altas/bajas
6. Desbloquear altas/bajas
7. Número de grupos en servidor
8. Miembros en grupo a indicar
9. Enviar mensaje
10. Recibir mensaje
11. Terminar ejecución
10
        --------------------RECIBO MENSAJE--------------------

Inserte alias de grupo, para recibir mensaje de este: grupo0
El mensaje obtenido es:
mensaje
        ------------------------------------------------------
```

Ilustración 9: Ejemplo de recibo de mensaje después del borrado del miembro del grupo

3.4. Cuestión 4

¿Qué sucede si un objeto no recoge un mensaje procedente de un grupo y posteriormente desaparece el grupo? ¿Podría posteriormente recoger el mensaje? Indica soluciones posibles a este problema.

```
        ------------------------------------------------------
Elija opción:
1. Crear grupo
2. Eliminar grupo
3. Añadir miembro al grupo
4. Eliminar miembro del grupo
5. Bloquear altas/bajas
6. Desbloquear altas/bajas
7. Número de grupos en servidor
8. Miembros en grupo a indicar
9. Enviar mensaje
10. Recibir mensaje
11. Terminar ejecución
10
        --------------------RECIBO MENSAJE--------------------

Inserte alias de grupo, para recibir mensaje de este: grupo0
Grupo grupo0 no existe...
```

Ilustración 10: Ejemplo de recibo de un mensaje de un grupo que no existe

Después de haber enviado el mensaje y borrado el grupo ocurre lo siguiente: no deja leer la cola porque el grupo ya no existe.

La solución puede ser no comprobar si el grupo existe, solo comprobar los mensajes de la cola si corresponden con el alias.