PRÁCTICAS DE SISTEMAS DISTRIBUIDOS.

Práctica 4: Multienvío no ordenado en Java RMI

PRÁCTICA ENTREGABLE

En esta práctica ampliaremos la funcionalidad de la práctica 3 para implementar un multienvío (no necesariamente ordenado) de mensajes entre los miembros de un grupo (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 *threads* separados, para permitir los envíos simultáneos dentro de un mismo grupo y simular tiempos diferentes de transmisión para cada destinatario. Asimismo, mientras haya envíos en curso se inhibirán las altas y bajas de miembros 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 en el caso de los clientes remotos (los que están en la máquina del servidor usan el mismo registro que el servidor), y darlos de alta en ellos con un nombre local (**se propone utilizar el alias del cliente**).

PASOS A SEGUIR:

1. Incluiremos un nuevo campo en la clase *GroupMember* para identificar el número de puerto del registro de cada cliente. Habrá que incluir esta información como parámetro en el constructor. También habrá que incluir esta información en los argumentos de los métodos de creación de grupos y de alta de nuevos miembros en los grupos.

2. Crear una nueva clase serializable *GroupMessage*, cuyos objetos son mensajes de un grupo. Campos: Mensaje (array de bytes) y emisor (*GroupMember*). El grupo ya está indicado dentro del objeto de clase *GroupMember*.

3. Quitaremos (comentaremos) los servicios *StopMembers* y *AllowMembers* del interface *GroupServerInterface*, del servidor y de *ObjectGroup*.

   En dicho interface *GroupServerInterface* incluimos un nuevo servicio:

   - *boolean sendGroupMessage(GroupMember gm, byte[] msg)*: Multienvío del mensaje *msg* por parte de *gm*, al grupo al que pertenece *gm*. Localiza el objeto de clase *ObjectGroup* e invoca el método del mismo nombre sobre él.

4. En la interface *ClientInterface* añadiremos dos servicios:

   - *void DepositMessage(GroupMessage m)*: Es un *callback* (método invocado desde el servidor), para depositar el mensaje *m* en la cola local del cliente. Como puede haber varios objetos enviando mensajes a la vez al cliente, **éste debe garantizar la exclusión mutua en dichas operaciones**[1]. Para ello se utilizará un *Lock* (implementación: *ReentrantLock*).

[1] La implementación de Java RMI podría desplegar un thread separado por cada petición, depende de la implementación concreta de Java RMI que usemos.

- *byte[] receiveGroupMessage(String galias)*: Para recoger de su cola local el siguiente mensaje correspondiente al grupo de alias indicado. **Si no hay ninguno se bloquea al invocador** hasta que llegue uno. Si no existe ese grupo o el cliente no pertenece a dicho grupo retorna *null* sin bloquearse.

  Observe que eso no evita que un grupo sea borrado mientras un cliente está esperando un mensaje de ese grupo, lo que dejaría indefinidamente bloqueado al cliente, pero no tratamos este problema.

5. Crear en *ObjectGroup* una variable entera privada que controle el número actual de miembros del grupo. Crear también los tres métodos siguientes, cuidando de su **ejecución en exclusión mutua**:

   - *void Sending()*: Para controlar el número de envíos en curso, incrementando un contador a tal efecto **en el número de miembros del grupo menos uno**. Se invoca al iniciar un envío, y permite activar el bloqueo de altas/bajas de miembros del grupo (si el contador es mayor que cero deben quedar bloqueados).
   - *void EndSending()*: Avisa del final de un envío, decrementando el contador en una unidad, y desbloqueando altas y bajas de miembros del grupo si el contador llega a cero.
   - *boolean sendGroupMessage(GroupMember gm, byte msg[])*: Se encarga del multienvío del mensaje *msg* a los miembros del grupo, exceptuando al emisor (gm). Invoca el método *Sending* para anotar los envíos que van a desplegarse y crea un *thread* separado de la clase *SendingMessage* para cada uno de los envíos. Una vez creados los *threads* devuelve *true*, es decir, finaliza antes de que los mensajes lleguen a sus destinos (envío asíncrono).

6. Crear una nueva clase *SendingMessage*, como extensión de *Thread* (hilo separado), para el envío de un mensaje (*GroupMessage*) a un miembro del grupo. Deben identificarse los argumentos requeridos en el contructor, y en su método *run* debe realizarse el envío, invocando el método *DepositMessage* del destinatario tras un **retraso arbitrario de entre 30 y 60 segundos**. Al finalizar cada envío se invocará el método *EndSending* del grupo.

7. Implementar el método *sendGroupMessage* en el servidor. Su ejecución se realiza en exclusión mutua y deberá comprobar que el grupo del emisor existe realmente, en caso contrario retorna *false*. Invoca el método *sendGroupMessage* del grupo considerado para realizar el envío, retornando lo que éste devuelva.

8. En la clase *Client* habrá que implementar los nuevos servicios indicados en su interface. Debe implementarse la cola local de mensajes (se propone usar *Queue*, con la implementación *LinkedList*), y un *Lock* para controlar la exclusión mutua en su manejo. El bloqueo a la espera de mensajes **debe hacerse mediante una variable de tipo *Condition* asociada a dicho *lock***.

   En el programa principal del cliente hay que incluir la creación del registro local en el puerto 1099 (o en otro libre diferente a elección del estudiante) en el caso de estar el cliente en una máquina distinta al servidor, y darse de alta en él con su alias como identificador del servicio. Después desplegará el menú de opciones, del cual habremos eliminado las opciones de bloquear/desbloquear altas y bajas de miembros del grupo, y se incluirán las de envío y recogida de mensajes de grupos, con sus correspondientes implementaciones.

9. Prueba de la práctica:

   - Crear al menos dos grupos y cuatro clientes, dos de ellos remotos, que podrán darse de alta/baja en los grupos.
   - Comprueba que se pueden realizar envíos simultáneos al mismo grupo, y que los mensajes no llegan necesariamente en el mismo orden a los destinatarios.
   - Mientras se realizan los envíos puedes probar el bloqueo de altas y bajas, así como el desbloqueo de los invocadores al completarse los envíos.
   - Comprueba que varios clientes que quieren recibir de un mismo grupo para el que no tienen mensajes quedan bloqueados esperando los mensajes, y que al enviar un mensaje al grupo todos ellos se desbloquean y lo obtienen.
   - Comprueba que un cliente no puede recibir mensajes de un grupo al que no pertenece, y el bloqueo indefinido de un cliente que espera un mensaje de un grupo que ha sido eliminado mientras el cliente esperaba dicho mensaje.

10. Entrega de la práctica. Debe entregarse en un fichero ZIP por uno de los miembros del grupo de prácticas, incluyendo el proyecto **exportado** de Netbeans y la memoria en PDF de la práctica, describiendo la participación de cada miembro del grupo. Dicha memoria debe incluir capturas de pantalla mostrando el correcto funcionamiento de la práctica.

11. Prueba ante el profesor. Se realizará preferiblemente al finalizar la práctica, si da tiempo a ello en las sesiones de prácticas. En su defecto se realizará en la evaluación final presencial de las prácticas, siempre que se haya aprobado la entrega. En dicha evaluación presencial se determinará el correcto funcionamiento de la práctica y el trabajo individual de cada miembro del grupo.