Práctica 2
Chat distribuido peer-to-peer

Sergio Herrero Barco - 698521
Alex Oarga Hategan - 718123

1 Introducción

Un chat distribuido peer-to-peer consiste en un grupo de usuarios que se comunican entre ellos sin ningún proceso coordinador. Para ello cada usuario va a enviar su mensaje a todos los demás. Un requisito de todo chat distribuido es que los mensajes lleguen en el orden correcto. Para eso el envío de mensaje se va a realizar en exclusión mutua, usando el algoritmo de Ricart-Agrawala. Así un usuario que quiera enviar un mensaje, solicita entrar en la Sección Critica, y cuando haya recibido las confirmaciones de todos los demás, podrá enviar su mensaje.

2 Desarrollo

Se ha creado un fichero en lenguaje Elixir en el que esta implementado el chat distribuido.

2.1 Arquitectura del Sistema

El sistema se compone de una serie de nodos que van a actuar siguiendo una Arquitectura Peer-to-Peer. Dicha arquitectura consiste en que cada nodo de la red actúa simultáneamente como cliente y como servidor.

Los procesos que actúan en cada nodo son:

1. Un proceso llamado receive chat que se va a encargar solamente de recibir los mensajes que han enviado por el chat, y mostrarlos por pantalla.
2. Un proceso llamado shared databse que se va a encargar de almacenar, entregar y/o modificar las variables del sistema, sea en Exclusión Mutua o no.
3. Un proceso llamado receive request que se va a encargar de gestionar los request que llegan y la lista de los diferidos.
4. Un proceso llamado mutex exclusion que se va a encargar de leer por pantalla el mensaje que se quiere enviar, pedir el acceso a la sección critica, enviar el mensaje a todos los usuarios, y la liberación de la Sección critica.

2.2 Implementación del algoritmo Ricart-Agrawala

En el algoritmo original de Ricart-Agrawala se usa el paradigma de memoria compartida y de semáforos. En Elixir no existe memoria pero el método de comunicación es el intercambio de mensajes, por lo que se ha tenido que hacer una modificaron en la implementación del algoritmo. Para ello, se ha creado un proceso llamado shared database que va a almacenar todas las variables compartidas. Los procesos

modifican y leen las variables a través del intercambio de mensajes con este proceso.

El semáforo se ha incluido también en el proceso shared database. Se ha implementado mediante el patrón de mutex distribuido que aprovecha que las llamadas receive son bloqueantes. Los procesos acceden a este semáforo mediante envío de mensajes como en el resto de variables.

El resto del algoritmo se ha implementado sin modificaciones.

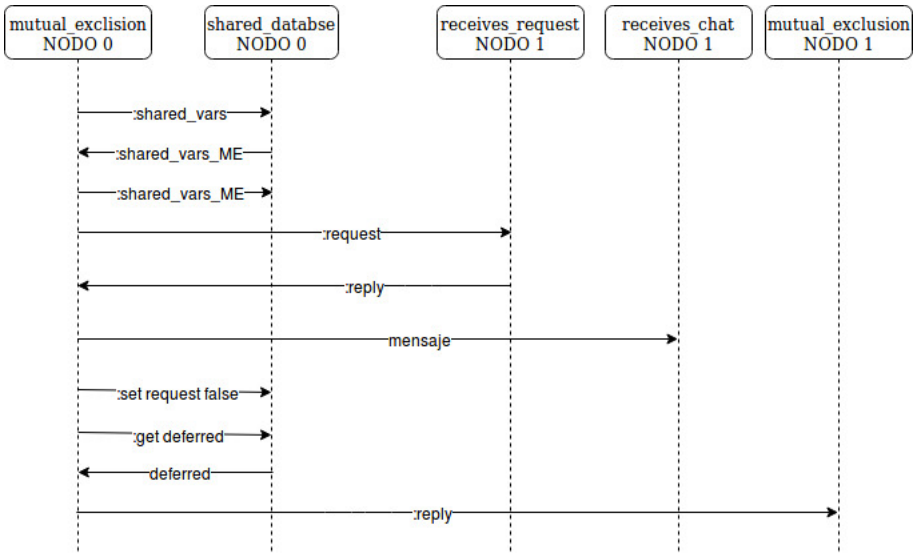
Toda la serie de acciones que se deben de hacer para solicitar la entrada a la Sección Critica, llamadas Pre-Protocol se han encapsulado en una función llamada con el mismo nombre, y que gestionara el envío de la petición a los demás usuarios, y la recepción de las confirmaciones.

Todas las acciones de liberación de la Sección Critica, llamadas Post-Protocol se han encapsulado en una función llamada con el mismo nombre, y que gestionara el envío de las recepciones a todos aquellos usuarios que van por detrás del usuario.

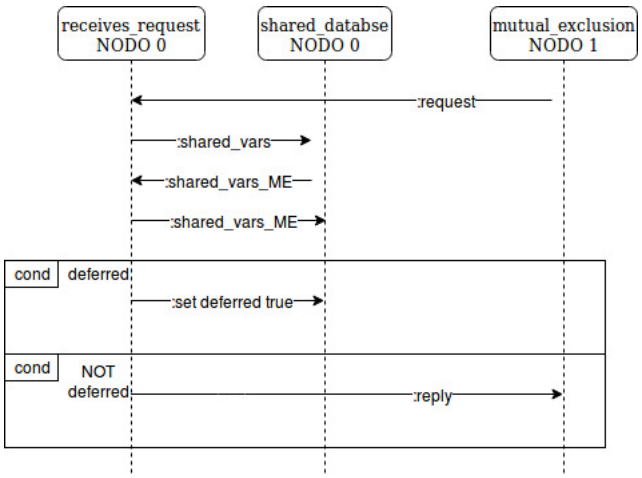
En cuanto al numero e identificación de nodos, se ha decidido que estos sean una constante del sistema conocida de antemano

2.3 Protocolo de comunicación

A continuacion se muestran los mensajes intercambiados entre procesos cuando se solicita el envio de un mensaje (1) y cuando se recibe una señal request (2):



(1). En este escenario solo existen 2 nodos comunicandose y Nodo 0 almacena en deferred a Nodo 1.



(2)

3 Validación experimental

Se ha probado el correcto funcionamiento del sistema de se a ejecutado el algoritmo variando el numero de nodos: 2, 3, 4 y 6 nodos

Para comprobar la implementación del almacenamiento de nodos en deferred, se ha bloqueado un proceso que se encontraba dentro de la SC (mediante la llamada bloqueante IO.gets). La finalidad de esto es forzar que se agreguen los demás nodos

que envían request a la lista de deferred. Notar que esto no modifica el comportamiento del algoritmo.

Por ultimo se ha recurrido a la herramienta 'Terminator' para simular un escenario parecido al envío simultaneo de varios mensajes al mismo tiempo. Se puede comprobar la corrección observando que el orden resultante del envío de los mensajes es siempre el mismo, ordenados por id ascendiente (por la asignación de prioridad).

4 Conclusión

En esta practica se ha implementado el algoritmo de Ricart-Agrawala y se ha aplicado a la coordinacion en un chat de mensajes.