Práctica 2

Chat distribuido peer-to-peer

César Martín Génova - 649711

José Félix Longares Moreno - 696215

2 de noviembre de 2017

Introducción

En esta práctica 2, se plantea el problema de crear un chat distribuido para n participantes en el cual los mensajes lleguen a todos ellos en el orden enviado. La implementación se ha realizado a través del algoritmo de Ricart Agrawala[1] y cada uno de los participantes será un nodo dentro de nuestro algoritmo.

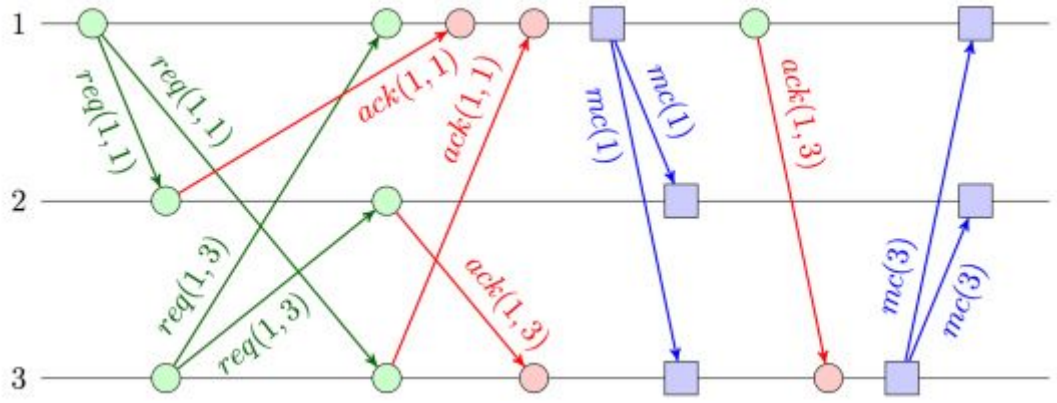
A lo largo de la memoria se explicará la implementación del algoritmo además de los problemas y soluciones que se han encontrado a la hora de realizarla implementación en Elixir.

Algoritmo de Ricart-Agrawala

En primer lugar se estudia el funcionamiento del propio algoritmo y a continuación se describe la implementación del mismo en Elixir. Como ya se ha comentado antes, se cuenta con n procesos distribuidos que se comunican exclusivamente mediante el paso de mensajes.

En general un proceso tiene dos opciones, acceder a la sección crítica para enviar un mensaje o estar simplemente escuchando los mensajes de los demás. En el caso de que no quiera enviar un mensaje, cuando le llegue una petición para enviar un mensaje por parte de otro proceso, le enviará al proceso correspondiente un ack para permitir a dicho proceso mandar el mensaje.

En el caso de que quiera enviar un mensaje, el proceso enviará una petición al resto de procesos y a continuación esperará a que todos le confirmen que puede enviarlo, y procederá a enviarlo a todos procesos del “chat”. Se puede dar el caso de que un proceso intente enviar en el momento en el cual hay otro ya en el proceso de envío, en ese caso se postergará la petición en caso de que ésta tenga una “prioridad” menor que la tuya (timestamp mayor) o se enviará el ack en caso contrario. Una vez que el proceso que está dentro de la sección crítica termine y siempre que no haya otro proceso con mayor prioridad, el proceso postergado entrará en la misma tras la recepción del ack que le faltaba.



Implementación en Elixir

El primer problema que se ha encontrado a la hora de implementar dicho algoritmo es como hacer que un participante pueda estar escuchando y enviando a la vez; la decisión adoptada fue la creación de threads[2][3] uno para escuchar y otro para recibir los mensajes del teclado y mandarlos al chat. Así mismo, se tenían que utilizar distintas variables globales por lo que se creó un nuevo hilo que hacía las veces de servidor de variables. Con esto se conseguía que los participantes pudieran leer y escribir a la vez y que mediante paso de mensajes un proceso pudiera consultar y modificar las variables que necesitara en cada momento.

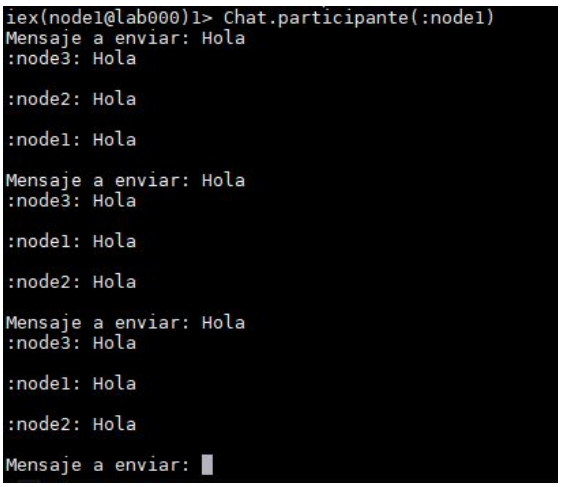
Cuando un participante quiere escribir, envía al servidor su pid y una petición de las variables, en caso de que mutex esté a 1 el servidor enviará al participante los datos ya que la sección crítica está vacía, en caso contrario, lo guardará en una lista a la espera de que el proceso que ocupa la sección crítica salga de ésta. En el caso de que entre, enviará una petición de envío de mensaje al resto de participantes, que contestarán con un ack. En el momento en el cual el participante tenga todas las contestaciones, enviará a todos los participantes el mensaje y actualizará las distintas variables antes de volver a enviárselas al servidor, comenzando de nuevo el proceso (leer de teclado, pedir acceso a la sección crítica,...).

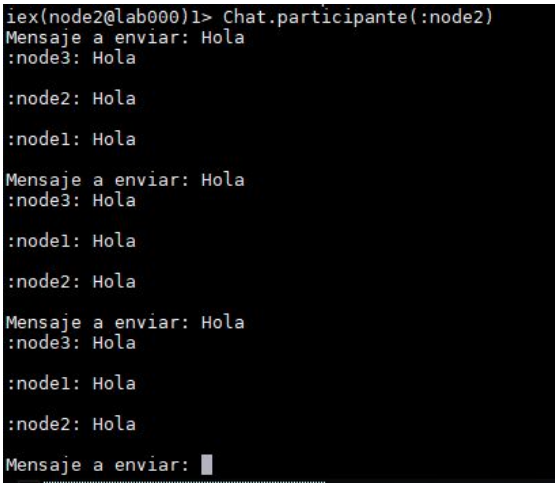
En el módulo escuchar, el participante está a esperas de la recepción de las peticiones del resto de procesos de enviar un mensaje (peticiones que se tratarán como se ha indicado más arriba), y de los propios mensajes del resto de integrantes del chat (mensaje que se mostrará por pantalla, junto con el remitente del mensaje).

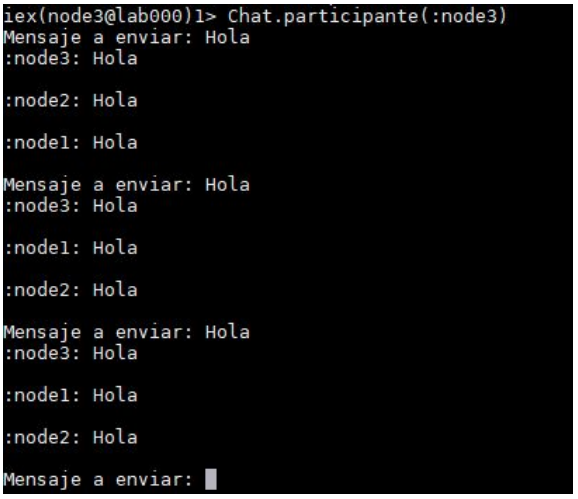
Validación Experimental

Se han realizado pruebas con tres participantes, uno de ellos en una máquina de laboratorio y los otros dos desde los portátiles de los integrantes de la pareja. Se han realizado distintas pruebas, enviando mensajes por individual y realizando envíos simultáneos, en ambos casos los resultados eran satisfactorios, es decir, llegaban a todos los participantes y los mensajes llegaban en el mismo orden a todos los procesos. Además de cara a comprobar el correcto funcionamiento a la hora de realizar envíos simultáneos se ha adjuntado un archivo de pruebas en el cual se duerme durante 2 segundos al proceso justo antes de enviar el mensaje, dando tiempo a los otros procesos a realizar una petición de envío de mensaje y así comprobar que la lista de postergados se trata correctamente, además de comprobar el orden de llegada de los mensajes a los diferentes integrantes del chat.

A continuación se muestran capturas de las pruebas realizadas, donde se observa que los mensajes llegan en el mismo orden en todos nodos.







Conclusiones

Realizar una solución completamente distribuida del sistema no es tan sencillo como se pudiera llegar a pensar. Las variables globales tienen que “centralizarse” en un proceso que las almacene. Además la sincronización mediante el algoritmo de Ricart-Agrawala no es trivial, es necesario el intercambio de una gran cantidad de mensajes sólo para conseguir y liberar el permiso para realizar la tarea objetivo (siendo un porcentaje mucho mayor de trabajo que la propia tarea propuesta).

Bibliografía

[1] Glenn Ricart, Ashok K. Agrawala, “An optimal algorithm for mutual exclusion in computer networks” in Communications of the ACM, Volume 24 Issue 1, Jan. 1981, pages 9-17

[2] Arronategui Unai, Tolosana Rafael, “Introducción al lenguaje Elixir” Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 26 Sept 2017

[3] Elixir web https://elixir-lang.org/