Práctica 2. Chat Distribuido

1. Objetivos y Requisitos

Esta práctica tiene como objetivo la implementación de un chat distribuido peer to peer, de manera que no exista ningún proceso coordinador, sino que todos los procesos participantes tienen la misma responsabilidad en el sistema.

1.1 Objetivo

- Familiarización de los relojes lógicos de Lamport y el algoritmo de Ricart-Agrawala
- Implementación de los relojes lógicos de Lamport en un SSDD programado en Elixir.
- Utilización de los relojes lógicos de Lamport y el algoritmo de Ricart-Agrawala para enviar mensajes *multicast* con ordenación total.
- Utilización del envío multicast con ordenación total para implementar un chat distribuido

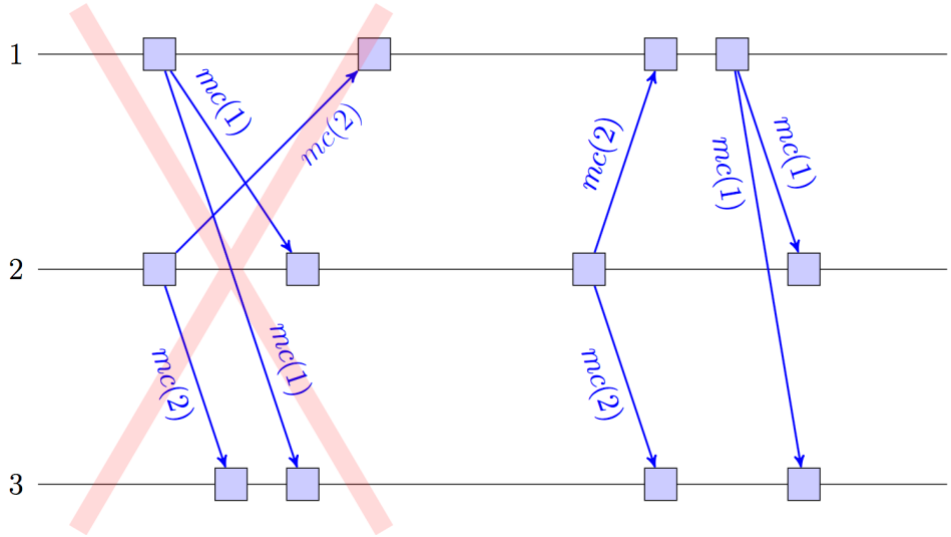
1.2 Requisitos

- Elixir y su entorno de desarrollo
- Seguir la guía de codificación de Elixir publicada aquí:
  https://github.com/christopheradams/elixir_style_guide/blob/master/README.md
- El algoritmo original de Ricart-Agrawala implementado en Algol
  https://dl.acm.org/citation.cfm?id=358537

2. Descripción del Problema y su Solución

En esta práctica vamos a implementar un chat distribuido. Una aplicación de chat distribuido permite a un grupo de usuarios enviarse mensajes de texto entre sí: lo que uno escribe, aparece inmediatamente en las pantallas de todos los participantes. Sin embargo, será requisito indispensable que todos los mensajes de texto se muestren en el mismo orden a todos los usuarios.

Figura 1. Ejemplos de protocolos de interacción de difusión amplia



Para eso, primero vais a implementar un protocolo de interacción *de difusión amplia* con ordenación total, utilizando las primitivas *send / receive* de Elixir. Este protocolo permite enviar mensajes de *difusión amplia* (cada mensaje llega todos los demás procesos), asegurando que todos llegarán a todo el mundo en el mismo orden. En la Figura 1, podéis ver una situación no deseada, en la que los mensajes del proceso 1 y 2 no llegan en el mismo orden a todo el mundo (izquierda), y una situación correcta, en la que llegan en el mismo orden a todos (derecha).

Tal y como se estudiado previamente en la asignatura, los relojes lógicos permiten ordenar los eventos en un sistema distribuido. Se estampilla cada mensaje con un valor que nos permite decidir si un cierto mensaje debe aparecer antes o después de otro. No obstante, en este caso de envío de mensajes en el chat distribuido, la ordenación solo se puede establecer una vez ha terminada la interacción entre los participantes, es decir, cuando ha terminado la aplicación. En dicho caso, se pueden recolectar todos los eventos y listarlos en el orden correcto. Pero cuando un proceso recibe un mensaje, *no tiene forma de saber si luego le llegará un mensaje con una estampilla anterior*. Por lo tanto, los relojes lógicos no son mecanismo suficiente para nuestra aplicación de chat.

Por ejemplo, una solución centralizada podría usar un secuenciador, que nos diera un número de secuencia incremental y *globalmente* único para cada mensaje. Un proceso no podría enviar su mensaje hasta que no le hubieran llegado todos los mensajes con un número de secuencia menor. Sin embargo, la solución no sería escalable. Una posible solución escalable consistiría en forzar que cuando un proceso envíe un mensaje de *difusión múltiple*, éste sea recibido por todos antes de que otro proceso envíe el siguiente mensaje. Es decir, hay que acceder al resto de procesos de forma exclusiva: *exclusión mutua distribuida*.

Usando un algoritmo de exclusión mutua distribuida, un proceso solicitará entrar en la sección crítica cuando quiera enviar un mensaje de difusión amplia. Este mensaje sólo se enviará de forma efectiva cuando obtenga el acceso a la sección crítica. Por ejemplo, el escenario de la Figura 1, con esta idea, puede verse en la Figura 2.

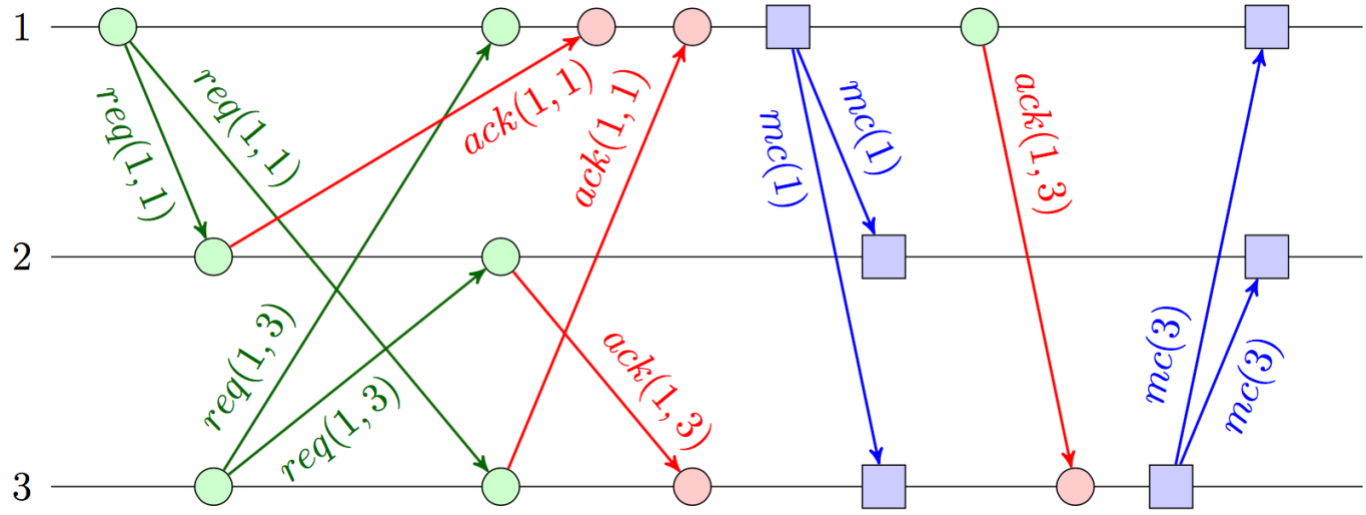


Figura 2. Escenario de envío de mensajes de difusión múltiple con exclusión mutua distribuida.

En la Figura 2, podemos ver el Algoritmo de Ricart-Agrawala [1] como algoritmo de acceso a la sección crítica. Los mensajes en verde tienen que ver con la solicitud de acceso a la sección crítica y los mensajes en rojo con el ACK de acceso. En azul, los mensajes de difusión amplia del chat. **Es interesante ver el envío de un mensaje de difusión amplia como un único evento, de manera que todos los mensajes enviados a los participantes se etiquetarán con la misma estampilla temporal**.

3. Ejercicio

Se pide implementar un chat distribuido tal cual se ha descrito en la sección anterior utilizando los relojes lógicos de *Lamport* y el Algoritmo de *Ricart-Agrawala*. El chat se formará por todas aquellos nodos que participen en la configuración. Por simplicidad, podéis hacer que el número de participantes sea estático y conocido a priori, podéis hacer que sea una constante en el programa o bien que los participantes se lean de un fichero de texto.

El algoritmo de Ricart-Agrawala deberá tener la misma estructura que el original propuesto en ALGOL, tal cual está descrito en [1]. No obstante, la implementación en ALGOL cuenta con mecanismos de memoria compartida, para traducirlos a ELIXIR deberéis utilizar mecanismos equivalentes en paso de mensajes. En particular, para implementar la memoria compartida tendrá que haber un proceso exclusivamente dedicado a implementar el patrón mutex, que almacene en memoria las variables compartidas y que garantice su acceso en exclusión mutua.

En la memoria deberéis describir la arquitectura de vuestro sistema distribuido de chat, cómo habéis implementado el Algoritmo de Ricart-Agrawala en Elixir y cómo habéis probado la corrección de vuestro sistema (distinto número de participantes, baterías de pruebas, etc.).

4. Evaluación

La realización de las prácticas es por parejas, pero los dos componentes de la pareja deberán entregarla de forma individual. En general estos son los criterios de evaluación:

- Deben entregarse *todos* los programas, se valorará de forma negativa que falte algún programa.

- Todos los programas deben compilar correctamente, se valorará de forma muy negativa que no compile algún programa.
- Todos los programas deben funcionar correctamente como se especifica en el problema.
- Todos los programas tienen que seguir el manual de estilo de Elixir, disponible en[1] (un 20% de la nota estará en función de este requisito). Además de lo especificado en el manual de estilo, cada fichero fuente deberá comenzar con la siguiente cabecera:

```
# AUTORES: nombres y apellidos
# NIAs: números de identificaci'on de los alumnos
# FICHERO: nombre del fichero
# FECHA: fecha de realizaci'on
# TIEMPO: tiempo en horas de codificación
# DESCRIPCI'ON: breve descripci'on del contenido del fichero
```

4.1. Rúbrica

Con el objetivo de que, tanto los profesores como los estudiantes de esta asignatura por igual, puedan tener unos criterios de evaluación objetivos y justos, se propone la siguiente rubrica en la Tabla 1. Los valores de las celdas son los valores mínimos que hay que alcanzar para conseguir la calificación correspondiente y tienen el siguiente significado:

A+ (excelente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea correctamente el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, sin errores. En el caso de la memoria, se valorará una estructura y una presentación adecuadas, la corrección del lenguaje, así como el contenido explica de forma precisa los conceptos involucrados en la práctica. En el caso del código, este se ajusta exactamente a las guías de estilo propuestas.

A (bueno). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea correctamente el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, con ciertos errores no graves. Por ejemplo, algunos pequeños casos (marginales) no se contemplan o

[1] https://github.com/christopheradams/elixir_style_guide/blob/master/README.md

no funcionan correctamente. En el caso del código, este se ajusta casi exactamente a las guías de estilo propuestas.

B (suficiente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No aplica el método de resolución adecuado y / o identifica la corrección de la solución, pero con errores. En el caso de la memoria, bien la estructura y / o la presentación son mejorables, el lenguaje presenta deficiencias y / o el contenido no explica de forma precisa los conceptos importantes involucrados en la práctica. En el caso del código, este se ajusta a las guías de estilo propuestas, pero es mejorable.

B- (suficiente, con deficiencias). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No se aplica el método de resolución adecuado y/o se identifica la corrección de la solución, pero con errores de cierta gravedad y/o sin proporcionar una solución completa. En el caso de la memoria, bien la estructura y / o la presentación son manifiestamente mejorables, el lenguaje presenta serias deficiencias y / o el contenido no explica de forma precisa los conceptos importantes involucrados en la práctica. En el caso del código, hay que mejorarlo para que se ajuste a las guías de estilo propuestas.

C (deficiente). El software no compila o presenta errores graves. La memoria no presenta una estructura coherente y/o el lenguaje utilizado es pobre y/o contiene errores gramaticales y/o ortográficos. En el caso del código, este no se ajusta exactamente a las guías de estilo propuestas.

Calificación	Código	Memoria
10	A+	A+
9	A	A
8	A	A
7	A	A
6	B	B
5	B-	B-
suspenso	1C	

Tabla 1. Rúbrica

5. Entrega

Deberéis entregar un fichero zip que contenga: (i) los fuentes: chat.exs, y (ii) la memoria en pdf (3 páginas). La entrega se realizará a través de moodle2 en la actividad habilitada a tal efecto. La fecha de entrega será no más tarde del jueves anterior al comienzo de la siguiente práctica, esto es, el 29 de octubre de 2018 para los grupos A y el 8 de noviembre de 2018 para los grupos B.

Los días 30 de octubre y 9 de noviembre durante la sesión de prácticas se realizará una defensa "in situ" de la práctica.

6. Bibliografía

[1] Ricart, G., & Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 24(1), 9-17. https://dl.acm.org/citation.cfm?id=358537