Práctica 2. Respositorio Datos Asignatura

1. Objetivos y Requisitos

Esta práctica tiene como objetivo la implementación del problema de los lectores / escritores en distribuido, de manera que no exista ningún proceso coordinador, sino que todos los procesos participantes tengan la misma responsabilidad en el sistema.

1.1 Objetivo

- Familiarización de los relojes lógicos de Lamport y el algoritmo de Ricart-Agrawala (generalizado)
- Implementación de los relojes lógicos de Lamport en un SSDD programado en Elixir.

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 una infraestructura para alojar un repositorio para el enunciado de un trabajo de asignatura, siguiendo el esquema clásico de sincronización de lectores y escritores. La infraestructura permitirá escribir en el repositorio a un profesor y leer del repositorio a un alumno. En todo momento, solo un usuario profesor (rol de escritor) puede acceder al repositorio, mientras que puede haber múltiples alumnos (rol de lector) leyéndola simultáneamente.

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 utilizar el algoritmo de Ricart-Agrawala generalizado para lectores y escritores.

El algoritmo original de Ricart-Agrawala [1] es una implementación del mutex distribuido y puede verse en la Figura 1. Un proceso distribuido que desee acceder a la sección crítica tiene que ejecutar el pre-protocol, realizado por las instrucciones (1) a (6). El pre-protocol para un Proceso Pi consiste en enviar a los N - 1 procesos distribuidos una petición de acceso a la sección crítica. Cuando se reciba la respuesta en (5) de los N – 1 entonces el Proceso Pi accede a la sección crítica. Una vez terminado el acceso a la sección crítica, el Proceso Pi realiza el post-protocol, instrucciones (7)-(9) donde se envía el ACK a los procesos cuyas peticiones habían sido postergadas. El Proceso Pi, mientras está ejecutando el pre-protocol, acceso a la sección crítica, post-protocol y sección no crítica (si la hubiera), tiene que atender simultáneamente las peticiones de otros procesos Pj. Por ello cada proceso Pi es consta de al menos 2 procesos concurrente, que ejecutan las instrucciones (10)-(15). La esencia del algoritmo reside en los relojes lógicos que se van incrementando y que permiten ordenar las peticiones de acceso a la sección crítica. Este algoritmo se puede generalizar fácilmente para resolver problemas de sincronización más complejos.

```
operation acquire_mutex() is
(1)  cs_state_i ← trying;
(2)  ℓrd_i ← clock_i + 1;
(3)  waiting_from_i ← R_i;    % R_i = {1, ..., n} \ {i}
(4)  for each j ∈ R_i do send REQUEST(ℓrd_i, i) to p_j end for;
(5)  wait (waiting_from_i = ∅);
(6)  cs_state_i ← in.

operation release_mutex() is
(7)  cs_state_i ← out;
(8)  for each j ∈ perm_delayed_i do send PERMISSION(i) to p_j end for;
(9)  perm_delayed_i ← ∅.

when REQUEST(k, j) is received do
(10) clock_i ← max(clock_i, k);
(11) prio_i ← (cs_state_i ≠ out) ∧ (⟨ℓrd_i, i⟩ < ⟨k, j⟩);
(12) if (prio_i) then perm_delayed_i ← perm_delayed_i ∪ {j}
(13)            else send PERMISSION(i) to p_j
(14) end if.

when PERMISSION(j) is received do
(15) waiting_from_i ← waiting_from_i \ {j}.
```

Figura 1: Algoritmo de Ricart-Agrawala

3. Algoritmo de Ricart-Agrawala Generalizado: de un mutex simple a un mutex de tipos de operación

De manera más general, existen problemas de sincronización que, en lugar de tener una única operación de acceso, consideran varios tipos de operaciones y reglas de exclusión entre ellas, por ejemplo, el problema de Lectores / Escritores. El algoritmo de Ricart-Agrawala puede extenderse fácilmente para implementar otros problemas más complejos, mediante dos cambios:

i) construyendo una matriz que defina las reglas de exclusión de las operaciones del problema de sincronización

ii) modificando las instrucciones (4) y (11) del Algoritmo de Ricart-Agrawala, de manera que incluya dicha matriz.

En general, en caso de que el problema conste de dos operaciones, *op1()* y *op2()*. Se puede definir una matriz simétrica booleana de exclusión de operaciones *exclude* (simétrica quiere decir que *exclude*[op1, op2] = *exclude* [op2, op1]). De manera que:

(i) si dos operaciones op1 y op2 no pueden ejecutarse simultáneamente, la matriz *exclude*[op1, op2] = true.

(ii) Si dos operaciones pueden ejecutarse simultáneamente entonces *exclude*[op1, op2] = false.

Por ejemplo, consideremos el problema de los lectores / escritores. Existen dos operaciones read() y write(). La matriz de exclusión es tal que *exclude*[read, read] = false, *exclude*[write, read] = exclude [write, write] = true.

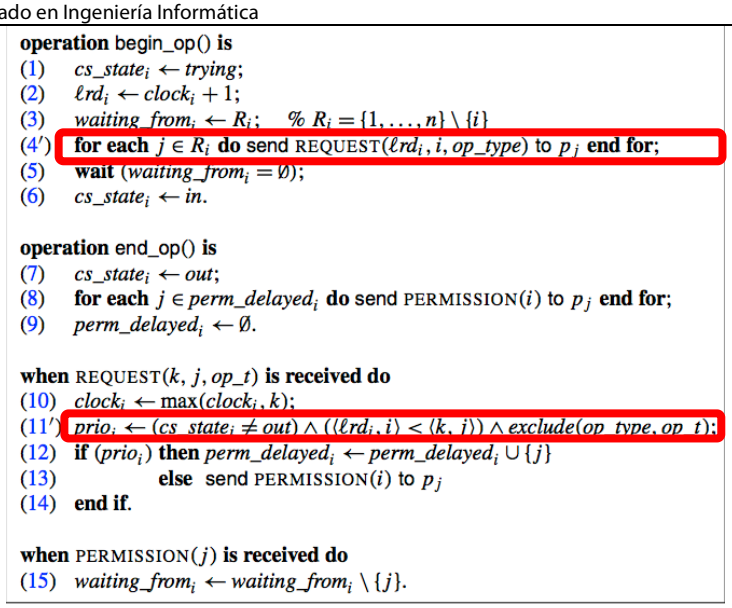

Figura 2: Algoritmo de Ricart Agrawala Generelizado.

4. Ejercicio

Se pide implementar el Algoritmo de *Ricart-Agrawala Generalizado*, para permitir el acceso al repositorio del enunciado del trabajo de asignatura siguiendo el esquema de sincronización de lectores y escritores. Además, deberéis diseñar programas de prueba con varios lectores / escritores para probarlo.

Adicionalmente, deberéis redactar una memoria donde deberéis describir 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
```

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

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 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: para_repositorio.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 día anterior al comienzo de la práctica 3.

Durante la sesión de prácticas 3 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