Memoria técnica

Sistemas distribuidos – Práctica 2

Grado de Ingeniería Informática, 2019-2020

Universidad de Zaragoza

Alberto Calvo Rubió
760739

Óscar Baselga Lahoz
760077

Introducción

Esta práctica tiene como objetivo la implementación del problema 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.
La solución propuesta es utilizar el algoritmo de Ricart-Agrawala generalizado para lectores y escritores.

Diseño del sistema

Para simplificar el diseño, se ha dividido el sistema en distintos módulos:

Semáforo

Proporciona la implementación en elixir de un semáforo binario como proceso. Dispone de las operaciones de creación, wait y signal. De esta forma, se consigue abstraer al usuario de los mensajes entre los procesos y el semáforo.

GlobalVars

Permite mantener el estado de unas variables globales predefinidas para el problema de lectores/escritores. Se ha implementado mediante un **Agent**. Este tipo de proceso ya proporcionado por elixir esta dedicado específicamente a guardar variables abstrayendo el envío de mensajes. Además, se ha decidido que el Agent guarde un tipo de dato **Map** para que las operaciones de get y set sean más sencillas de utilizar indicando con un átomo la variable que se quiere pedir o guardar.

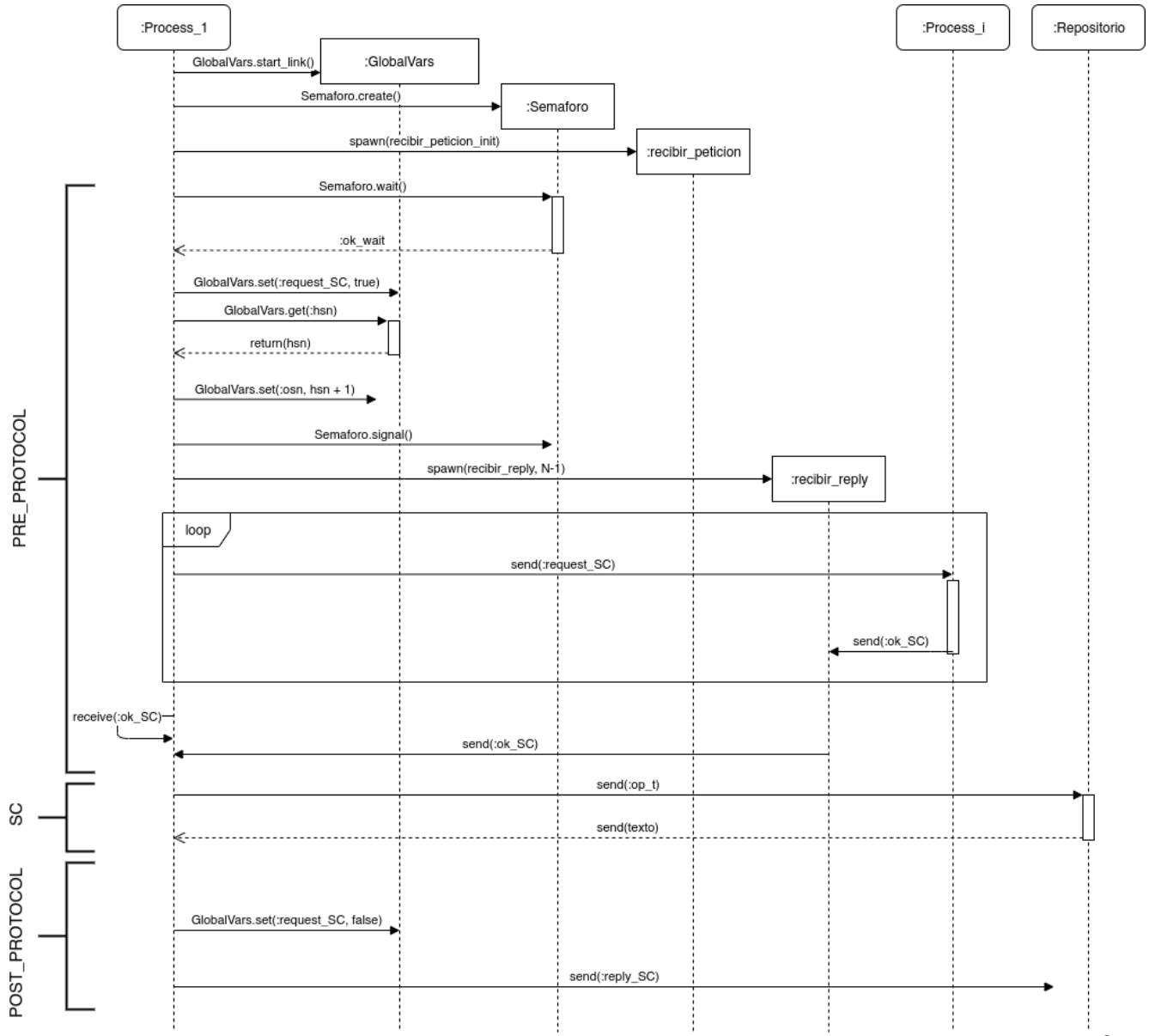
Para_Repositorio

Módulo principal del sistema. Por una parte se encuentran los lectores y escritores y por otro las funciones privadas que son utilizadas por estos.
Tanto lectores como escritores siguen un patrón:

1. Se inicializan mediante init(), función en la que se spawnea el semáforo y el proceso de variables globales, y se prepara alguna variable necesaria durante su ejecución, simplificando la función principal.
2. Realizan el spawn del proceso de escucha de peticiones (recibir_peticion_init). Esta función se registra con el átomo :recibir_peticion facilitando la comunicación entre lectores/escritores sin tener que saber el pid del proceso, únicamente el nombre del nodo. A la hora de decidir si la operación que solicita otro proceso entra en conflicto con la que quiere realizar uno mismo, se utiliza la función exclude. Esta

contiene una matriz que **relaciona** entre sí **las 6 operaciones** disponibles. Posteriormente, realiza el comportamiento establecido por el algoritmo de Ricart-Agrawala.

3. Ejecutan el pre_protocol. Aquí se continua el algoritmo y se lanza el proceso que va a escuchar las respuestas de los demás lectores/escritores. A continuación se envían las peticiones con el pid del proceso creado para que envíen sus respuestas. La función de enviar_peticiones() realiza los envíos y después espera la respuesta del proceso que lleva la cuenta de procesos que han contestado. En caso de que no se hayan recibido todas las respuestas puede significar que algún proceso no ha recibido la petición (Por ejemplo: no ha iniciado aún su proceso de escuchar peticiones), por lo que se establece un **timeout** para reenviar los mensajes de petición, permitiendo cierta **tolerancia a fallos**.
4. Entran en la sección critica escribiendo/leyendo en el repositorio
5. Ejecutan el post_protocol



Pruebas

Se han añadido tiempos de espera para simular la sección crítica.

- 4 nodos(1 escritor y 3 lectores): todos eligen osn = 1 porque no les ha dado tiempo de enviarse mensajes unos a los otros, por lo que el nodo 1 gana a los demás y aplaza la respuesta a los demás, en cuanto los demás le dan permiso entra en la SC y escribe. Mientras los demás procesos esperan. Una vez termina el nodo 1, envía a los demás que pueden continuar. Al ser los otros 3 lectores, entran en SC sin esperarse entre sí.

Traza de procesos (los mensajes no salen en orden cronológico):

```
"Nodo 3 -> Peticion de 4: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(4), op1(read_entrega), op2(read_entrega)"
"Nodo 1 -> Peticion de 2: defer_it(true) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(update_entrega), op2(read_entrega)"
#PID<13696.112.0>
"Nodo 3 -> Peticion de 1: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(1), op1(read_entrega), op2(update_entrega)"
"Nodo 3 -> Peticion de 2: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(read_entrega), op2(read_entrega)"
"Nodo 4 -> Peticion de 1: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(1), op1(read_entrega), op2(update_entrega)"
"Nodo 4 -> Peticion de 2: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(read_entrega), op2(read_entrega)"
"Nodo 4 -> Peticion de 3: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(3), op1(read_entrega), op2(read_entrega)"
"Nodo 2 -> Peticion de 1: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(1), op1(read_entrega), op2(update_entrega)"
"Nodo 2 -> Peticion de 3: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(3), op1(read_entrega), op2(read_entrega)"
"Nodo 2 -> Peticion de 4: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(4), op1(read_entrega), op2(read_entrega)"
"Nodo 3 -> Peticion de 4: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(4), op1(read_entrega), op2(read_entrega)"
"Nodo 1 -> Peticion de 4: defer_it(true) --> request_sc(true), osnVecino(1), osn(1), idVecino(4), op1(update_entrega), op2(read_entrega)"
"Nodo 1 -> Peticion de 2: defer_it(true) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(update_entrega), op2(read_entrega)"
"Nodo 4 -> Peticion de 2: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(read_entrega), op2(read_entrega)"
"Nodo 3 -> Peticion de 2: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(2), op1(read_entrega), op2(read_entrega)"
"Nodo 2 -> Peticion de 3: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(3), op1(read_entrega), op2(read_entrega)"
"Nodo 1 -> Peticion de 3: defer_it(true) --> request_sc(true), osnVecino(1), osn(1), idVecino(3), op1(update_entrega), op2(read_entrega)"
"Nodo 4 -> Peticion de 3: defer_it(false) --> request_sc(true), osnVecino(1), osn(1), idVecino(3), op1(read_entrega), op2(read_entrega)"
"Nodo 1 -> he escrito"
"Nodo 4 -> he leido: 1"
"Nodo 2 -> he leido: 1"
"Nodo 3 -> he leido: 1"
iex(nodo6@127.0.0.1)2>
```

Traza del repositorio (se muestra la representación del estado cada vez que se recibe un mensaje, por lo que se puede observar en orden cronológico):

```
iex(nodo5@127.0.0.1)2> Repositorio.init
, , 1
, , 1
, , 1
, , 1
```

Además se han realizado pruebas con diferente número de lectores y escritores garantizando que siguen el orden dejando pasar o no a los demás procesos mediante la operación:

$$(11)\ \ prio_i \leftarrow (cs\_state_i \neq out) \wedge ((\ell rd_i, i) < (k, j));$$

En elixir se ha implementado como:

```
defer_it = request_SC && ((osnVecino > osn) || (osnVecino == osn && idVecino > me)) && exclude(op1, op2)
```

En las pruebas se han comprobado específicamente los siguientes apartados:

- Comprobar orden variando los osn e ID del proceso lector/escritor
- Comprobar orden según si realizan operaciones que se excluyen o no
- Comprobar que se aplazan o no peticiones si se quiere entrar o no en SC