Sistemas Distribuidos

- práctica 2 -

Por:

Diego Caballé Casanova (738712)

Algoritmo de ricart-agrawala

La traducción del algoritmo de Argol a Go ha sido parcialmente literal.

En una librería he creado la función request junto con la operación “exclude”, esta función será utilizada por una goroutine por cada uno de los procesos participantes, de forma que puedan escuchar y responder a los procesos en todo momento.

Dado que request necesita una variables, (como lrd o el state del proceso) y el main (que intenta entrar en la SC) también necesita de otras variables, (como $clock_i$) se ha decidido que las variables que cambie Request y utilice main y viceversa sean globales y punteros, de forma que, cuando la go rutina sea cree, ambos dos procesos (main y go rutina) pueden acceder y cambiar los valores.

Entre las condiciones para el funcionamiento de ricart-agrawala, el único cambio ha sido la traducción de {lrd, i} < {k, j} se ha creado una variable booleana tuples en donde, si los relojes son iguales, decidirá el PID, si los relojes no son iguales, decidirá si el mío es menor que el suyo.

En la ejecución, para hacer pruebas, todos los procesos estarán esperando a que el proceso de PID 3 de la salida para empezar la ejecución. Así que hay que desplegar todos los procesos antes de lanzar el proceso 3. Se puede ver en la zona del código con el comentario *sincronización.*

Pruebas

Una vez terminado el algoritmo, se han creado una batería de problemas que pudieran cubrir todos los casos.

Las partes críticas a probar son:

Correcta decisión sobre PID, correcta decisión sobre clock, correcta decisión sobre estado y correcta decisión sobre tipo de operación.

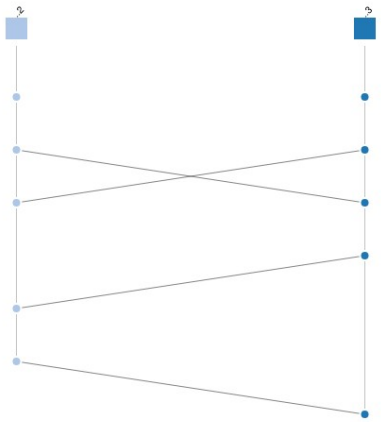
Para visualizar estas pruebas se han utilizado las herramientas del enunciado.

Shiviz se ha configurado completando los campos con:

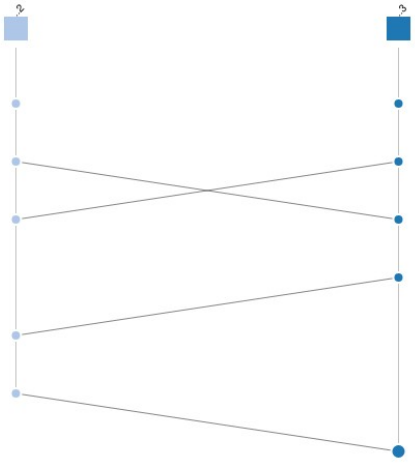
(?<host>\w+ [1-4]) (?<clock>\{.*\})\n(?<event>.*) y {Complete,Message}

Tipo de operación y PID

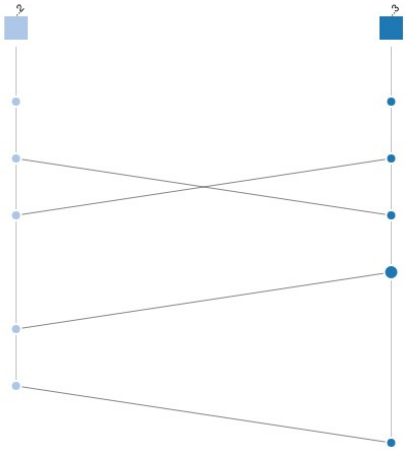
Dos readers



Un reader y un writer



Dos writers

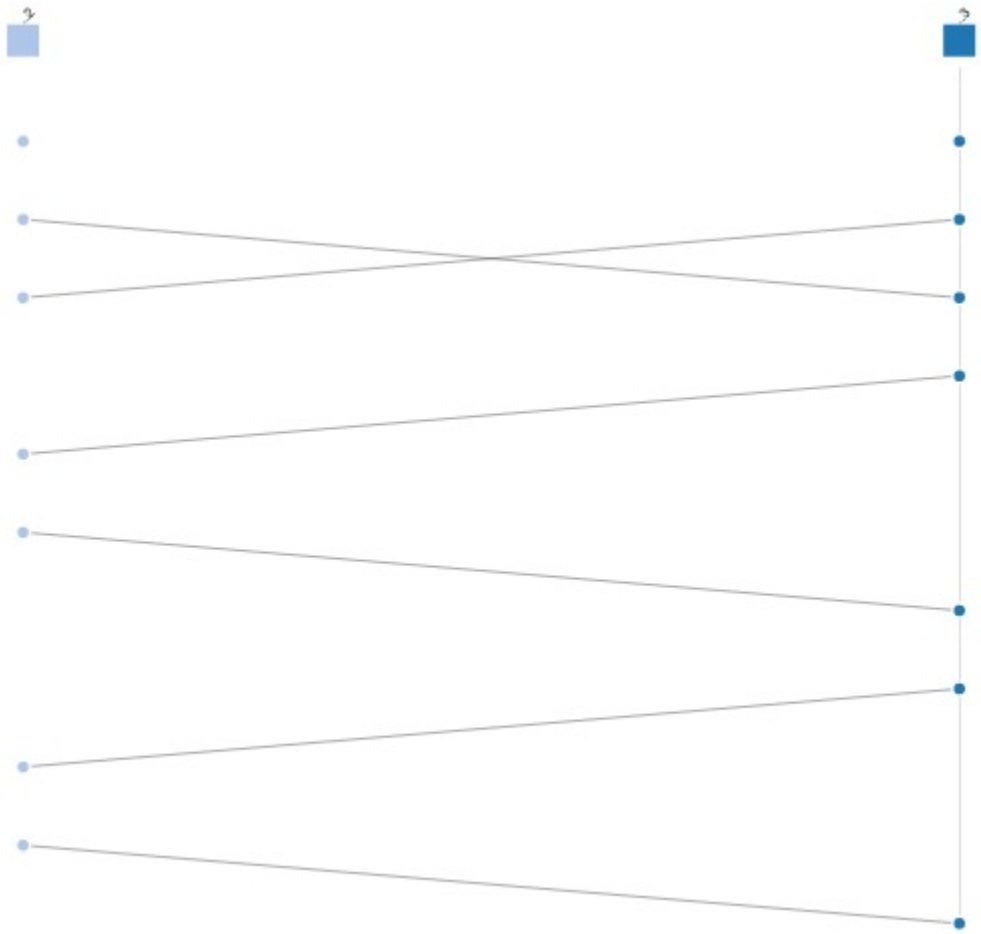


Un writer y un reader

Como vemos, cuando la operación se puede realizar concurrentemente, los procesos se responden de forma simétrica, es decir, uno envía request, otro le deja pasar.

Cuando las operaciones son excluyentes, el grafo cambia, en este caso, el proceso con PID menor envía request y recibe permiso mientras que el de PID mayor se queda apuntado, y cuando el proceso de PID menor ha acabado en SC, envía el permiso. La cuarta prueba era para demostrar que el tipo de operación no es lo que decide, sino el PID.

Estado



Aquí tenemos una ejecucción donde 2 ha terminado, pero 3 ha realizado dos peticiones a SC. Como dos está en el estado 3, es decir, out, en cuanto recibe el segundo request de 3, 2 le da permiso.

El log de operaciones sobre la base, que demuestra el correcto funcionamiento, es el siguiente:

Operation from 2: START_WRITE

Operation from 2: WRITE

Operation from 2: WRITE

Operation from 2: END_WRITE

Operation from 3: START_READ

Operation from 3: READ

Operation from 3: READ

Operation from 3: END_READ

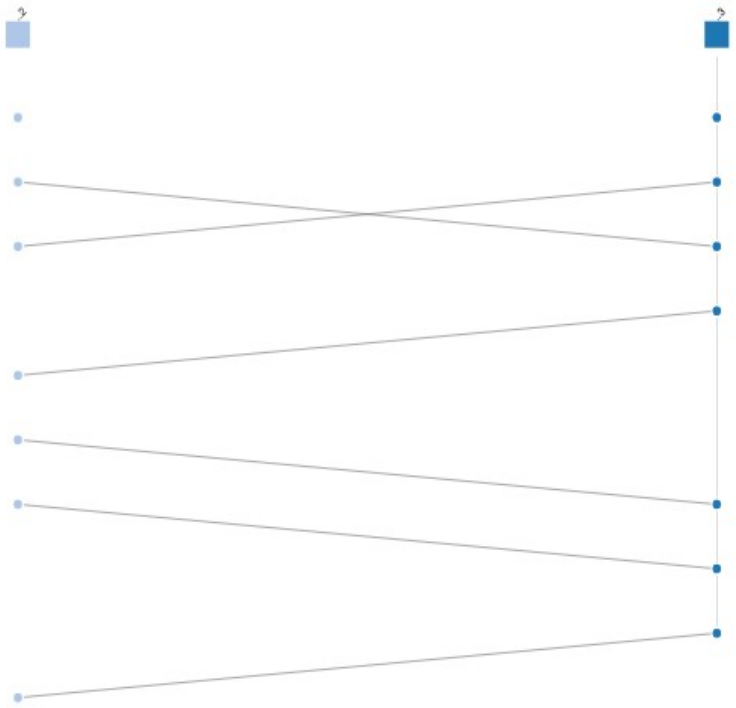
Operation from 3: START_READ

Operation from 3: READ

Operation from 3: READ

Operation from 3: END_READ

Clock



En esta última comprobación el proceso 2 (reader) intentará hacer dos operaciones, mientras 3 (writer) se duerme antes de entrar en seccion crítica. Lo que vemos al principio es la sucesión de mensajes que ya hemos visto, 2 quiere entrar, 3 quiere entrar, como 3 es un writer, entra sólo 2 y a la salida de SC le deja entrar a 3. 3 en este momento se duerme, y es por ello que 2 envía su request y no recibe respuesta, aquí no se dislumbra el tiempo, pero 2 se queda el tiempo que 3 está dormido esperando a la última respuesta de 3. En la salida estandar del programa 3 se muestra por pantalla como le deniega la entrada a 2.