SISTEMAS DISTRIBUIDOS

Memoria práctica 2

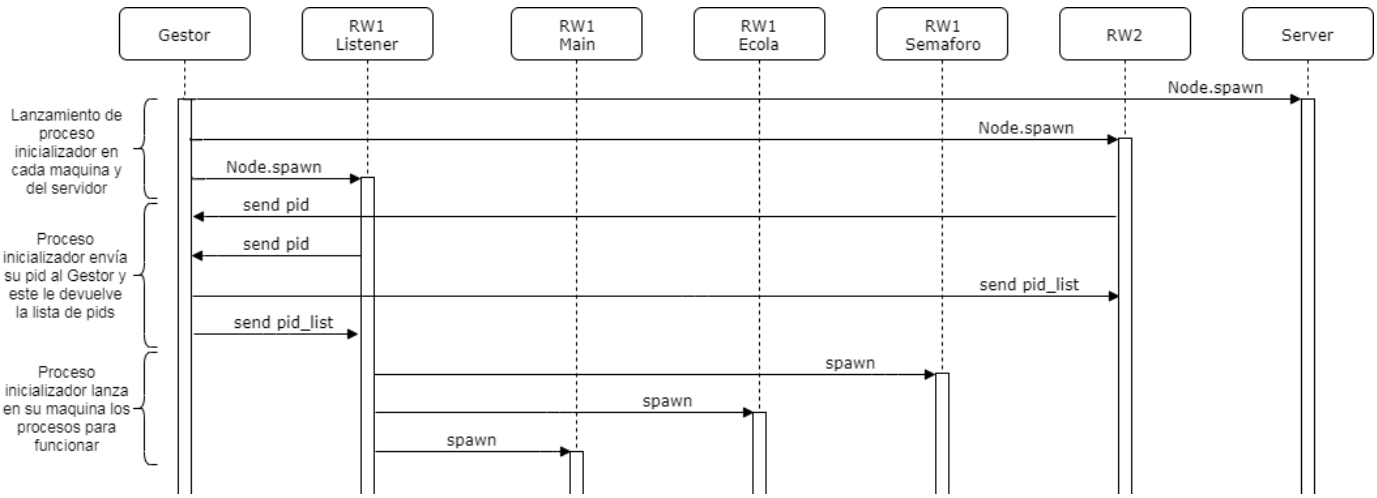
Sergio García Esteban 755844
Irene Fumanal Lacoma 758325

Introducción

En esta práctica hemos implementado un mutex distribuido no centralizado basado en Ricart-Agrawala para gestionar un repositorio (SC) al que acceden profesores (escritores) y alumnos (lectores) con unas restricciones establecidas (reglas de exclusión).

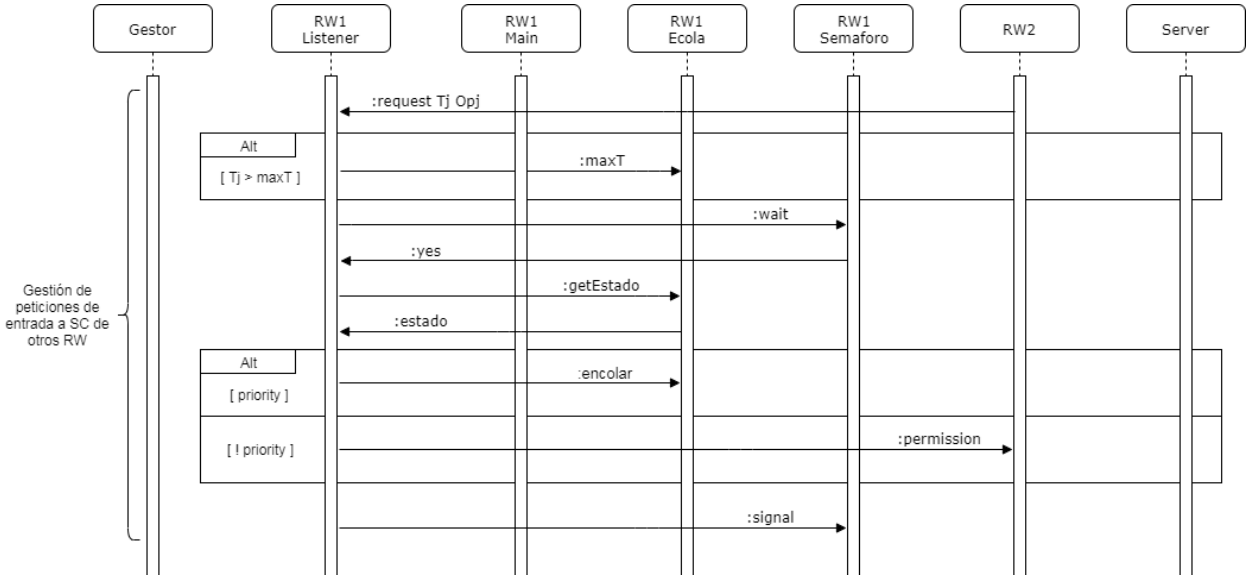
¿Como iniciamos el servicio?

1. Utilizamos un proceso Gestor que se encarga de lanzar en cada nodo su proceso principal.
2. El proceso principal de cada nodo envía su pid al Gestor, para que este genere una lista con ellos y se la reenvíe.
3. Al recibir la lista, el proceso principal de cada nodo lanzará los procesos necesarios para el correcto funcionamiento.



Gestión de entrada a SC

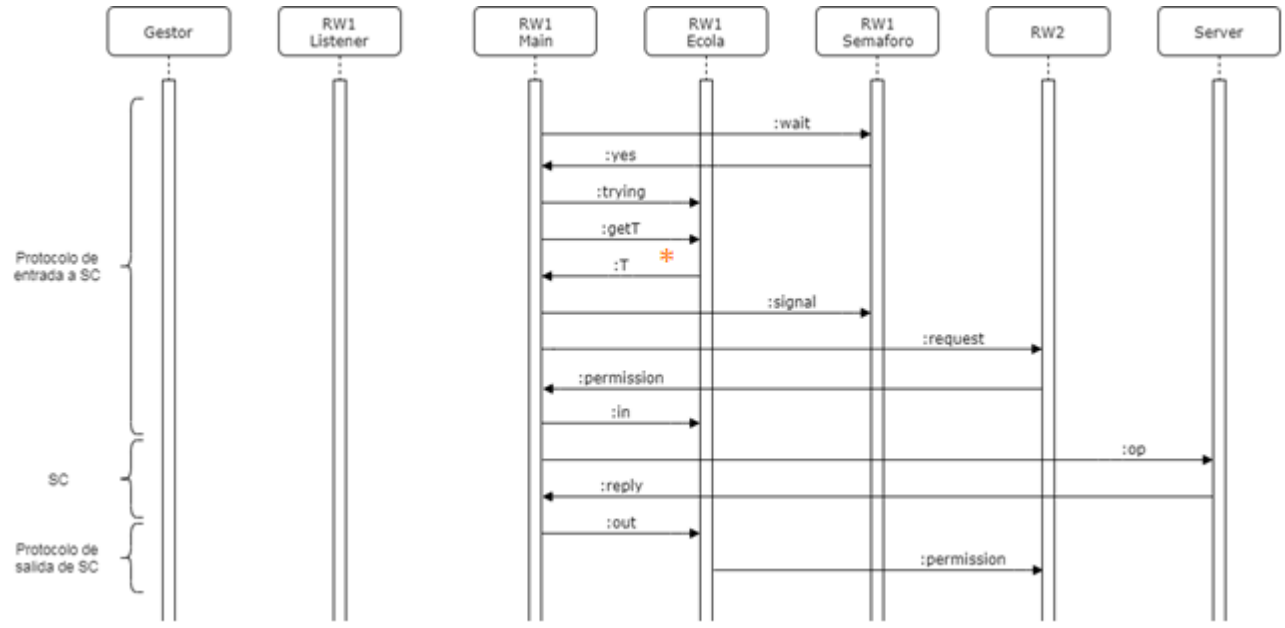
Para entrar a SC, debes obtener un reloj en exclusión mutua con la gestión de peticiones. Nos ayudamos de un proceso Semáforo para obtener exclusión mutua y de un proceso Ecola que guarda y gestiona: estado, maxReloj y pendingCola.



Gestión de peticiones de entrada a SC

Al recibir una petición, el algoritmo se encarga de actualizar el maxClock y determinar si darle permiso de entrada ahora o dárselo en cuanto Pi salga de SC.

*: Sincronización del clock (clock + 1)



Escenarios de prueba

Para comprobar el correcto funcionamiento del algoritmo hemos realizado diferentes pruebas, aquí mostramos las que demuestran las principales cualidades de sincronización.

Escenario 1

```
iex(node7@127.0.0.1)1> Gestor.escenario4
"REPOSITORIO on"
"-CLK- node1@127.0.0.1 adquiere reloj 1"
"-CLK- node2@127.0.0.1 adquiere reloj 1"
"+++++ node1@127.0.0.1 LECTOR in SC"
"+++++ node2@127.0.0.1 LECTOR in SC"
"----- node2@127.0.0.1 LECTOR out SC"
"----- node1@127.0.0.1 LECTOR out SC"
"-CLK- node3@127.0.0.1 adquiere reloj 2"
"-CLK- node4@127.0.0.1 adquiere reloj 2"
"+++++ node3@127.0.0.1 ESCRITOR in SC"
"----- node3@127.0.0.1 ESCRITOR out SC"
"+++++ node4@127.0.0.1 LECTOR in SC"
"----- node4@127.0.0.1 LECTOR out SC"
"-CLK- node5@127.0.0.1 adquiere reloj 3"
"-CLK- node6@127.0.0.1 adquiere reloj 3"
"+++++ node5@127.0.0.1 ESCRITOR in SC"
"----- node5@127.0.0.1 ESCRITOR out SC"
"+++++ node6@127.0.0.1 ESCRITOR in SC"
"----- node6@127.0.0.1 ESCRITOR out SC"
"bye"
```

Con esta prueba comprobamos que se cumple la cualidad de Exclusión mutua (con reglas de exclusión). Lanzamos 2 lectores a la vez (reloj 1) y ambos coinciden dentro de SC. Tras una espera, se lanza 1 escritor y 1 lector (reloj 2), entra el proceso prioritario (menor pid) y el otro entra cuando sale el primero. Finalmente, se lanzan 2 escritores (reloj 3) para comprobar que no coinciden dentro de SC.

Escenario 2

```
iex(node5@127.0.0.1)1> Gestor.escenario3
"REPOSITORIO on"
"..... node1@127.0.0.1 LECTOR trying SC"
"-CLK- node1@127.0.0.1 adquiere reloj 1"
"..... node2@127.0.0.1 ESCRITOR trying SC"
"-CLK- node2@127.0.0.1 adquiere reloj 1"
"-RCV- node1@127.0.0.1 recive request con reloj 1"
"-RCV- node4@127.0.0.1 recive request con reloj 1"
"..... node4@127.0.0.1 LECTOR trying SC"
"-RCV- node2@127.0.0.1 recive request con reloj 1"
"-RCV- node3@127.0.0.1 recive request con reloj 1"
"..... node3@127.0.0.1 LECTOR trying SC"
"-CLK- node3@127.0.0.1 adquiere reloj 2"
"-RCV- node3@127.0.0.1 recive request con reloj 1"
"-RCV- node2@127.0.0.1 recive request con reloj 2"
"-RCV- node1@127.0.0.1 recive request con reloj 2"
"+++++ node1@127.0.0.1 LECTOR in SC"
"-RCV- node4@127.0.0.1 recive request con reloj 1"
"-CLK- node4@127.0.0.1 adquiere reloj 2"
"-RCV- node1@127.0.0.1 recive request con reloj 2"
"-RCV- node2@127.0.0.1 recive request con reloj 2"
"-RCV- node3@127.0.0.1 recive request con reloj 2"
"-RCV- node4@127.0.0.1 recive request con reloj 2"
"----- node1@127.0.0.1 LECTOR out SC"
"+++++ node2@127.0.0.1 ESCRITOR in SC"
"----- node2@127.0.0.1 ESCRITOR out SC"
"+++++ node4@127.0.0.1 LECTOR in SC"
"+++++ node3@127.0.0.1 LECTOR in SC"
"----- node4@127.0.0.1 LECTOR out SC"
"----- node3@127.0.0.1 LECTOR out SC"
"bye"
```

Una segunda prueba nos permite comprobar, mostrando más información, que no se producen errores de inanición y bloqueo. Además, observamos que el algoritmo cumple la cualidad de ordenación. En la prueba lanzamos 3 procesos lectores y 1 escritor, pero el escritor pide el acceso a SC antes de que lo hagan 2 de los lectores. La prueba muestra que el escritor accede después del primer lector y antes de los otros 2 lectores.