[ARQUITECTURA DISTRIBUÏDA]

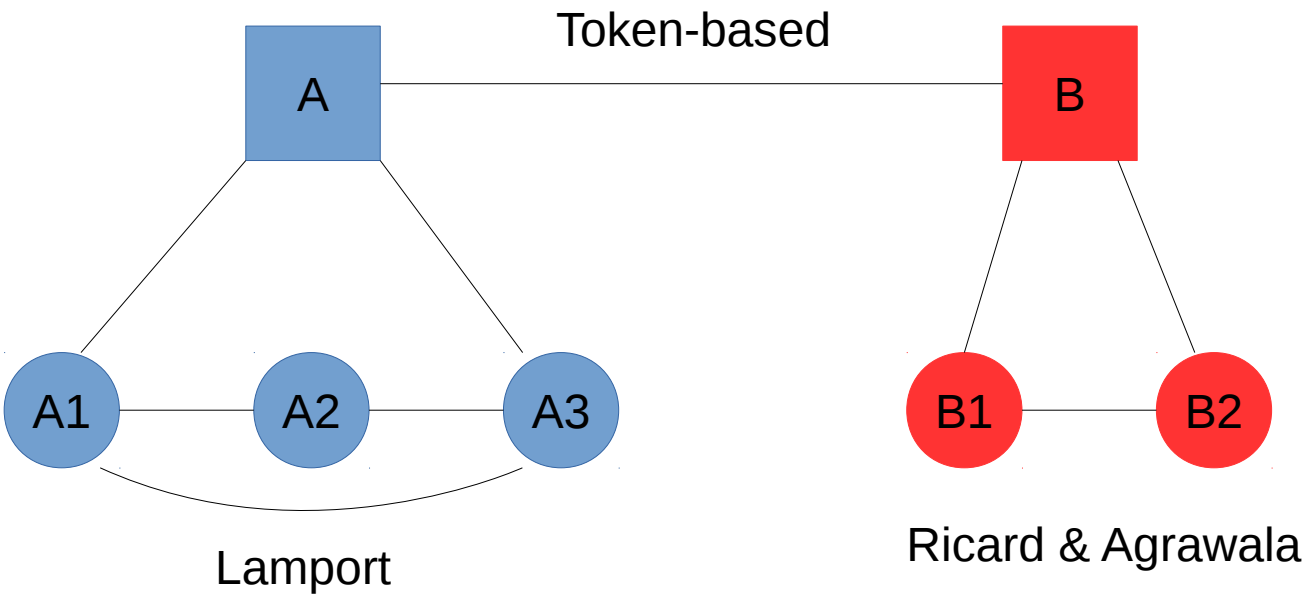
PRACTICA 2

Lamport, Ricard & Agrawala

Índex

Introducció

En aquesta pràctica s'implementen 3 mètodes de sincronització i exclusió mútua entre processos: token-based, Lamport Logical Clock i Ricard & Agrawala Mutual Exclusion. L'objectiu és protegir la pantalla com a regió crítica de tal manera que només un procés pugui accedir a ella i alhora sincronitzar l'execució dels processos per tal que aquests segueixin un ordre.



Funcionament

El funcionament de l'algorisme és el següent:

1. A té el token.
2. A1 mostra el seu missatge per pantalla.
3. A2 mostra el seu missatge per pantalla.
4. A3 mostra el seu missatge per pantalla.
5. A passa el token a B.
6. B1 mostra el seu missatge per pantalla.
7. B2 mostra el seu missatge per pantalla.
8. B passa el token a A.
9. Tornar al punt 2.

Primerament s'han de configurar tots els sockets del sistema, concretament 9, 1 per cada línia que s'aprecia a la imatge anterior. Els sockets cobreixen les funcionalitats següents:

- Entre A i B: Es el canal pel qual passarem el token.
- Entre A i A1: Per avisar a A1 que ja pot escriure per pantalla.
- Entre A i A3: Per avisar a A que tots 3 processos ja han escrit per pantalla.
- Entre A1, A2 i A3: Pel pas de missatges del protocol Lamport.
- Entre B i B1: Per avisar a B1 que ja pot escriure per pantalla.
- Entre B i B2: Per avisar a B que els 2 processos ja han escrit per pantalla.
- Entre B1 i B2: Pel pas de missatges del protocol Ricard & Agrawala.

Protocol Lamport

El funcionament d'aquest protocol és senzill, cada node pot enviar 3 tipus de missatge:

- REQUEST: Demanar accés a la regió compartida.
- REPLY: Contestar en rebre un missatge del tipus REQUEST.
- RELEASE: Alliberar la regió compartida.

En començar tots els nodes envien en broadcast una petició REQUEST i s'encuen a la llista. En rebre una petició REQUEST els nodes afegiran aquesta petició a la llista, l'ordenaran de menor a major timestamp i seguidament enviaran un missatge REPLY al node en qüestió. En rebre un REPLY s'augmentarà un comptador, si aquest comptador és 2, voldrà dir que ja ha rebut REPLY d'ambdós nodes, per tant pot comprovar si ell és el primer a la llista, si no és el primer, haurà d'esperar a rebre un RELEASE, desencuar al primer de la llista i tornar-ho a comprovar. Si és el primer de la llista, podrà accedir a la regió compartida, escriure 10 cops per pantalla el seu identificador, esperar 1 segon,seguidament enviar en broadcast un missatge RELEASE, desencuar el primer node de la llista (ell mateix) i enviar en broadcast un missatge REQUEST. Sempre que un node envia un missatge augmenta el seu timestamp en 1 unitat. Sempre que un node rep un missatge actualitza el seu timestamp al màxim entre el timestamp rebut i el propi, seguidament l'augmenta una unitat.

Protocol Ricard & Agrawala

El funcionament d'aquest protocol és molt semblant al de Lamport, però més optimitzat, ja que fusiona els missatges REPLY i RELEASE

- REQUEST: Demanar accés a la regió compartida.
- REPLY: Donar permís d'accés a la regió compartida al receptor de part de l'emissor.

En començar els 2 nodes envien en broadcast una petició REQUEST. En rebre una petició REQUEST cada node comprova si el seu timestamp és menor que el rebut, si ho és llavors encua al node per tal de saber a qui enviar REPLY un cop ha sortit de la regió compartida, si no ho és, envia REPLY al node que ha enviat el REQUEST. En rebre un REPLY el node entrarà en la regió compartida, mostrarà el seu identificador 10 cops per pantalla, enviarà un REPLY a tots els nodes de la cua mentre els desencua i finalment enviarà un REQUEST en broadcast.

Classes

- LamportClock.java: Classe que implementa un rellotge lògic de Lamport.
- LamportHWProcess.java: Procés A, crea els processos A1, A2 i A3 i els inicia.
- LamportNode.java: Processos A1, A2 i A3.
- LamportQueueNode.java: Classe per fer la llista de peticions pendents a ambdós protocols. Conté timestamp i id.
- Mainjava: Crea els 2 processos Heavyweight A i B i els inicia.
- RAHWProcess.java: Procés B, crea els processos B1 i B2 i els inicia.
- RANode.java: Processos B1 i B2.
- StdListener.java: Crea un procés per poder llegir de 2 sockets alhora sense bloquejar el Thread principal, s'utilitza en els processos A1, A2 i A3.

Conclusions i problemes observats

El principal problema que he tingut amb aquesta pràctica ha estat el desconeixement general d'ambdós protocols, he hagut de cercar per internet, ja que tot i la informació de l'eStudy m'ha estat útil, documentar-me externament m'ha ajudat a fer-me una idea del concepte i així poder implementar-los.

Un altre problema amb el qual em vaig topar va ser a l'hora de dissenyar el protocol Lamport, perquè com podia rebre missatges d'ambdós buffers d'entrada per cada procés, no podia bloquejar el thread principal esperant a rebre un missatge per un socket, ja que podia rebre'l per l'altre. Per solucionar-ho vaig crear 2 threads per cada procés de manera que cadascun llegia d'un socket mentre el thread principal s'esperava en un semàfor a què un dels 2 threads revés un missatge i el desbloqueges.

Aquesta pràctica m'ha ajudat a entendre com funciona realment ambdós protocols, però també a entendre la idea d'un algorisme i aprendre com implementar-lo seguint un disseny propi.