EXCLUSIÓ MÚTUA

Pràctica de projectes en arquitectura distribuïda

Curs 2015-2016

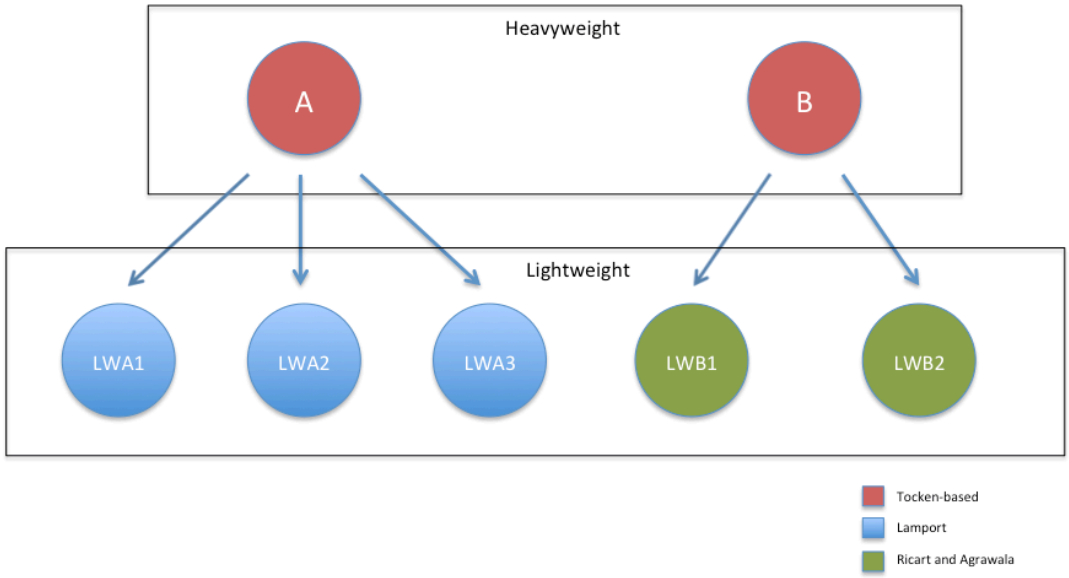
Balbina Virgili Rocosa
ls27476

Índex

1-. Introducció

L'objectiu principal d'aquesta pràctica és implementar una aplicació mútua distribuïda, en la qual tots els processos executats en la mateixa màquina lluitaran per un mateix recurs compartit com és la pantalla.

L'aplicació a implementar és la que es veu en la imatge de continuació:



L'aplicació, doncs, tindrà dos processos heavyweight A i B els quals invocaran els seus respectius processos lightweight, que mostraran infinitament el seu identificador per pantalla durant 10 vegades i esperaran un segon entre interval.

Entre els processos caldrà implementar diferents polítiques d'exclusió mútua, tal i com s'indica en la llegenda de la imatge mostrada anteriorment.

2-. Disseny general del sistema

Per tal d'implementar l'aplicació distribuïda demanada, primerament s'ha dividit el disseny en quatre grans blocs:

- Política centralized (tocken-based)

Hi ha un servidor central addicional que s'encarrega de rebre i gestionar les peticions dels nodes que volen accedir a la regió crítica. Aquest node s'encarrega de que només un dels nodes tingui el tocken en cada moment.

Quan un servidor ho desitja, envia una petició de request al servidor central, que gestiona la petició:

- Si té el tocken ell, el dóna al node que ha realitzat la petició retornant-li un acknowledge.

- Si no té el tocken, afegeix la petició en llista de pendents i, quan el servidor que té el tocken l'alliberi, li assignarà amb un acknowledge.

En rebre un acknowledge, el servidor sap que té el tocken i, per tant, que pot accedir a la zona crítica. Una vegada ha acabat la seva rutina, avisarà novament el servidor central amb un release, indicant que ja ha acabat d'utilitzar la regió crítica i, per tant, el tocken passa a estar novament en mans del servidor central que gestionarà el tocken amb futures peticions.
Cada node d'aquest algorisme ha estat implementat amb un únic fil d'execució.

- Política Lamport

En aquest segon cas, ja no hi ha un node addicional que “reparteix el joc”, sinó que són els propis nodes que es comuniquen entre ells i reconeixen qui ha de tenir la zona crítica de la següent manera:
Quan un servidor vol accedir a la zona crítica ho demana a la resta enviant una petició de request. La resta de nodes, al rebre la petició li contesten amb un acknowledge.
El node s’espera a entrar a la regió crítica fins que el seu valor lògic no sigui menor a tots els valors lògics de la resta de nodes. Una vegada ha acabat la utilització de la zona crítica, envia un release a cadascun de la resta de nodes.
Cada node d’aquest algorisme ha estat implementat amb dos threads, un per l’execució de la zona crítica i l’altra per gestionar els missatges rebuts per la resta de nodes.

- Política Ricart and Agrawala

Aquesta última política és semblant a la de Lamport, amb la diferència que es redueixen el número de missatges.

En aquest cas doncs, ja no hi ha tampoc un node addicional que “reparteix el joc” i, per tant, la comunicació es realitza entre els diferents nodes.
Quan un servidor vol accedir a la zona crítica ho demana a la resta enviant, també una petició de request. En aquest cas, però, la resta de nodes no li respondran amb un acknowledge amb el seu valor lògic, sinó que li respondran amb un release, només en el cas en que ells no estiguin interessats en utilitzar la regió crítica (no tinguin doncs cap request pendent) o, quan el valor lògic actual sigui més gran que el del request demanat pel node. En cas contrari, afegiran el request en una llista de pendent i no respondran fins que ells rebin la confirmació de la zona crítica i la deixin d’utilitzar.
Així doncs, un node només podrà utilitzar la zona crítica, en el moment en que hagi rebut un missatge de release de cadascun dels altres nodes.
En el moment de finalitzar la utilització de la zona crítica, enviarà un release a tots els altres nodes que s’han emmagatzemat a la llista de pendents (han sol·licitat un request posterior al actual).
Cada node d’aquest algorisme ha estat implementat amb dos threads, un per l’execució de la zona crítica i l’altra per gestionar els missatges rebuts per la resta de nodes.

- Comunicació entre nodes

	0	1	2
0	Stack<message>	Stack<message>	Stack<message>
1	Stack<message>	Stack<message>	Stack<message>
2	Stack<message>	Stack<message>	Stack<message>

A més a més, per tal de poder enviar els missatges propis de cada política d’exclusió mútua, s’ha creat una comunicació entre servidors mitjançant matrius de comunicació.

D'aquesta manera s'ha aconseguit que els processos heavyweight poguessin comunicar-se amb els processos lightweight que ha invocat i, al mateix temps, que aquests processos lightweight també es puguin comunicar entre ells.

Aquesta comunicació s'ha pogut realitzar gràcies a una matriu, on a cada posició de la matriu hi ha una cua de missatges. Cada node, en funció del seu Id escriurà sempre als altres nodes en una sèrie de posicions de l'array i llegirà els missatges per ell en unes posicions diferents de l'array.

- Les posicions d'escriptura són: *(IdActual, IdDesti)*
- Les posicions de lectura són: *(IdDesti, IdActual)*

La comunicació entre nodes es fa a partir de Messages, els quals estan formats per l'id d'origen, l'id de destí, el tipus i el valor.

Així doncs, la implementació final de la pràctica ha sigut la següent:

El main de l'aplicació crea la capa heavyweight amb tres processos centralized (central, A i B) i una matriu de comunicació entre ells. Al mateix temps, cadascun dels processos crearà els seus processos ligthweight respectius; A tres Lamport i B dos RAM. De la mateixa manera, cadascun també crearà una altra matriu de comunicació per comunicar-se amb ells, la qual els processos lightweight també utilitzaran per comunicar-se entre ells.

Així doncs, al iniciar l'execució, primerament seran els processos A i B els que lluitaran per aconseguir el tocken i, seguidament ho comunicaran als seus processos invocats. Aquests, al rebre que el pare té el tocken començaran a comunicar-se per tal d'aconseguir la regió crítica i, al aconseguir-ho, mostraran el seu identificador per pantalla. Una vegada acabada la rutina de la regió crítica, avisarà al seu pare i, una vegada que el procés heavyweight detecti que tots els processos invocats ja han alliberat la regió crítica, es desfarà també del tocken. Serà llavors, una vegada més que el servidor central centralitzat escollirà quin es el següent procés que li toca el tocken per tal de poder entrar

a la regió crítica com és la pantalla. Aquest procés es realitzarà indefinidament en l'aplicació.

3-. Proves realitzades

Per tal de comprovar el funcionament de l'aplicació implementada, s'han creat per cada node tantes pantalles com threads té implementat el procés. Així doncs, es pot comprovar en aquestes l'intercanvi dels missatges entre els nodes i quan cadascun entra dins la regió crítica i, una vegada ha acabat l'execució i ja no la necessita, l'allibera i l'obté el següent node que l'ha sol·licitat.

D'aquesta mateixa manera, el desenvolupament de la pràctica ha estat incremental de manera que es va començar a desenvolupar primer l'intercanvi entre els nodes heavylight, afegint noves polítiques ligthweight quan les anteriors ja funcionaven correctament.

Per tal de comprovar el bon funcionament de l'aplicació, també s'ha provat que els nodes mostressin per pantalla el seu identificador a diferents velocitats. Tant en el cas ràpid com en el lent, el resultat ha sigut satisfactori.

4-. Problemes observats i conclusions

El principal problema trobat al implementar l'aplicació va ser entendre des d'un primer moment les diferents polítiques d'exclusió mútua a fons, per tal de poder-les implementar després. Sobretot la diferencia entre l'algorisme de Lamport i de Ricart and Agrawala. Tot i això una vegada entès, seguint els pseudocodis facilitats a classe es va aconseguir la seva implementació.
L'algorisme que va causar més confusió va ser el de Lamport.

A més a més, pel gran nombre de threads que es creen al executar l'aplicació era difícil que l'ordinador no es pengés a l'hora de provar-ho i, també, que es mostressin tots els logs necessaris per comprovar el seu funcionament.
És per això que es va decidir crear un JFrames diferent per cada thread de l'aplicació, ja que d'aquesta manera es veia clarament el log de cada una, sense perdre cap informació, tot i que feia que el rendiment de l'aplicació fos més baix.

També va costar la comunicació a partir de la matriu de connexió, ja que al principi alguns missatges s'escrivien al lloc d'escriptura i viceversa.

Crec que ha sigut una pràctica molt interessant a realitzar, ja que en ella s'han conegut nous algorismes d'exclusió mútua i són una manera alternativa de mantenir informació actualitzada entre diferents servidors, molt diferent a lo conegut fins al moment. En general crec que són algorismes més efectius a altres mètodes que ja coneixíem fins ara, com el de tenir un servidor central que gestiona ell totes les peticions dels diferents servidors. Tot i això, crec que també té les seves limitacions, ja que de moment no té informació de si un node ha caigut o no tindrà mai resposta i, per tant, estarà esperant una resposta que mai arribarà.