Informe del seminari: Muty

Alejandro Carol
Héctor Mañosas
17 d'Octubre, 2013

1 Introducció

L'objectiu de la pràctica és implementar un sistema basat en el bloqueig distribuït d'exclusió mútua (mutex). Es farà amb estratègia de multicasting en una xarxa asíncrona.
La pràctica es divideix en tres parts, una per cada tipus d'implementació del lock:
- **Lock1**: Sistema propens al deadlock que l'únic que comprova és que només accedeixi un procés a la zona crítica a la vegada.
- **Lock2**: Millora respecte el lock anterior basada en establir un sistema de prioritats.
- **Lock3**: Millora respecte els locks anteriors basada en la utilització de rellotges de Lamport.

2 Feina feta

Hem implementat diferents versions del lock (una incompleta, una amb prioritats i una altra amb el rellotge de Lamport) i hem realitzat unes proves per cadascun d'ells.

Primera part
En aquesta part s'ha revisat, compilat i executat el codi proporcionat utilitzant el lock1.

Segona part
En aquesta part hem desenvolupat el sistema lock basat en prioritats. Hem partit del codi proporcionat realitzant les següents modificacions:

- Hem assignat a cada procés lock un identificador que s'ha utilitzat com a prioritat de 1 a 4 (on 1 és la màxima prioritat).
- Per a que els processos poguessin saber quina prioritat se'ls havia assignat hem propagat la prioritat per les diferents capçaleres:

```
init(Id, Nodes)
open(Id, Nodes)
requests(Id, Nodes)
wait(Id, Nodes, Master, [], Waiting)
wait(Id, Nodes, Master, Refs, Waiting)
held(Id, Nodes, Waiting).
```

- Per establir el sistema de prioritats hem modificat la funció *wait()* donant prioritat als processos amb identificador més baix. Per tant, si un procés rep un *request* amb més prioritat aquest li garanteix l'accés responent *ok* i continuant en estat de *wait* esperant rebre el seu *ok* i els que falten. Així garantim que aquest no entri a la zona crítica mentre hi hagi un altre procés amb més prioritat.

Tercera Part
En aquesta part hem desenvolupat el sistema de lock utilitzant rellotges de Lamport.
Amb aquesta implementació un procés cedeix el pas (envia *ok*) quan el que ha fet un *request* la fet abans que ell (en temps lògic).
Utilitzem un rellotge lògic que es troba inicialitzat a 0 per a tots el processos. S'incrementa en 1 i es propaga el seu valor cada cop que es rep un *request* o un *ok*.

3 Experiments

Hem fet diferents proves amb cadascun dels locks amb paràmetres d'Sleep i Work diferents.

i) Fes tests amb diferents paràmetres de Sleep i Work per analitzar com la implementació del lock respon als diferents graus de contenció.

Els resultats dels tests es troben a l'annex 1.

ii) Divideix el mòdul del muty i fes els canvis necessaris per cada parell de worker-lock i executa'ls en diferents màquines (això és, John i l1 hauria d'executar-se a una màquina, Ringo i l2 a una altra i així.)

Si executem els workers en diferents nodes podem observar com el temps mig per aconseguir el lock augmenta lleugerament, degut a l'overhead que comporta la comunicació entre els diferents nodes.

4 Preguntes obertes

Lock1

Què passa quan incrementem el risc de conflicte de lock?

Quan augmentem el risc de conflictes de locks augmenten els deadlocks, el que provoca que pugi el nombre de withdrawals.

Lock2

Justifica com garanteixes que només un procés estigui a la secció crítica a la vegada.

Quan els locks estan inactius (en mode open) donen l'ok a tothom. Quan volen accedir a la secció crítica el lock es posarà en estat wait fins que rebi un senyal ok dels altres nodes. En cas que algun altre li "deixés passar" per tenir més prioritat, aquest lock també demanaria ser notificat quan alliberés la zona crítica, de tal manera que fins que no s'allibera la zona crítica no hi pot entrar ningú.

Què es el principal drawback de la implementació del lock2?

Que els nodes que tinguin prioritat baixa tindran molts withdrawals.

Lock3

Seria possible que un worker no tingués accés al lock inclús si va emetre una solicitud a la instància abans (assumim temps global absolut) que el worker que va pendre el lock?

Sí podria passar, però el temps seria d'una diferència mínima (microsegons). Això ocorre perquè no contemplem el temps lògic en els workers, només en les instàncies del lock.

Annex 1:

2000, 1000	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken
John	630	0	24	464	0	6	558	0	15
Ringo	596	0	22	505	0	7	485	0	15
Paul	655	0	21	552	0	6	606	0	13
George	625	0	24	303	0	8	524	0	16
1000, 2000	lock1			lock2			lock3		
	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken
John	2547	0	16	843	0	89	2380	0	22
Ringo	2260	0	16	999	0	79	2260	0	11
Paul	2173	0	16	3489	5	32	2172	0	10
George	2149	0	17	3972	22	4	1739	0	11
500, 500	lock1			lock2			lock3		
	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken
John	496	0	52	177	0	62	492	0	74
Ringo	497	0	49	288	0	51	520	0	70
Paul	512	0	47	667	0	34	517	0	71
George	505	0	51	1749	1	15	518	0	73
100, 100	lock1			lock2			lock3		
	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken	Avg TIme (ms)	Withdrawal	Locks taken
John	574	1	84	36	0	436	102	0	114
Ringo	308	4	79	56	0	381	98	0	110
Paul	391	3	83	122	0	271	97	0	110
George	309	4	80	459	0	109	96	0	114