Seminar Report: Muty

March 20, 2019

Carles Tornel
Jesus Alfaro
Ricard Abril

1 Introduction

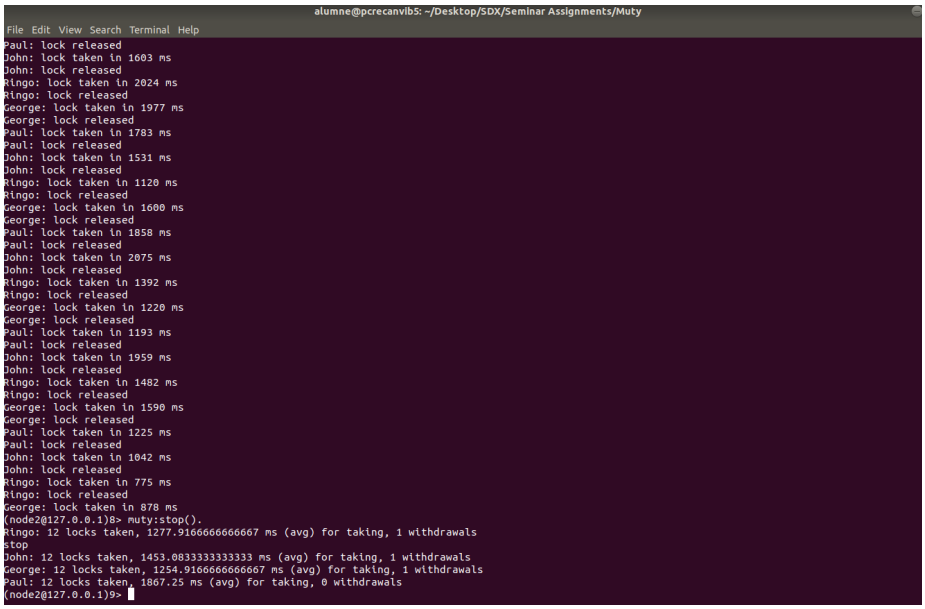
2 Experiments

Some Testing

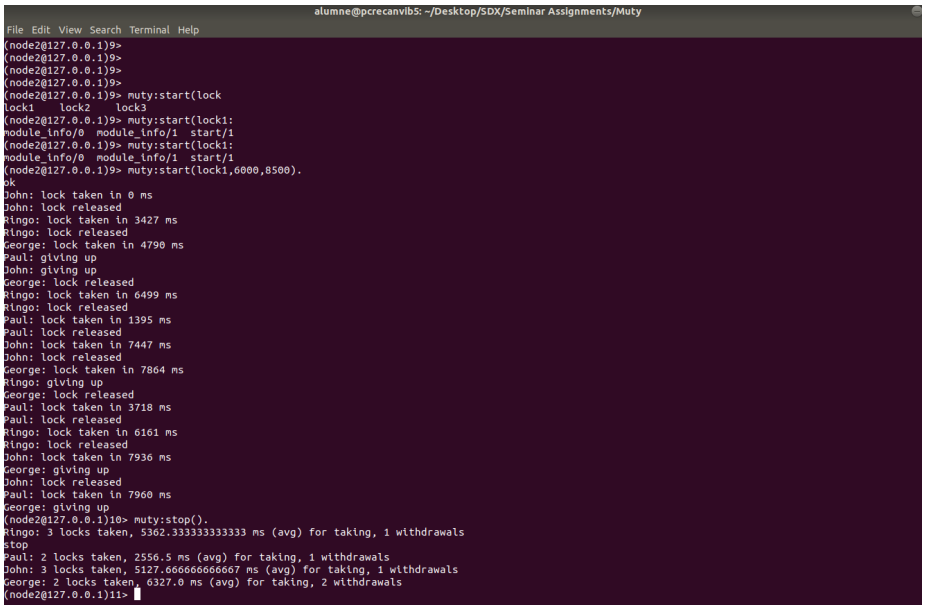
1. Make tests with different Sleep and Work parameters to analyze how this lock implementation responds to different contention degrees.

   #### Test 1: Sleep time inferior a work time

   Si el temps de sleep és considerablement més petit que el temps de work, la probabilitat de que diferents workers demanin accés a la regió critica serà més alta, quan això passa, aquets workers esperaran un missatge d' OK de la resta, però aquets també es estaran en la mateixa situació, de forma que ens trobarem davant d'un deadlock. Encara que al estar durant 8 segons en aquesta situació, el programa allibera els lock, per tal de poder seguir executant-se. Tal i com podem veure en la següent imatge:

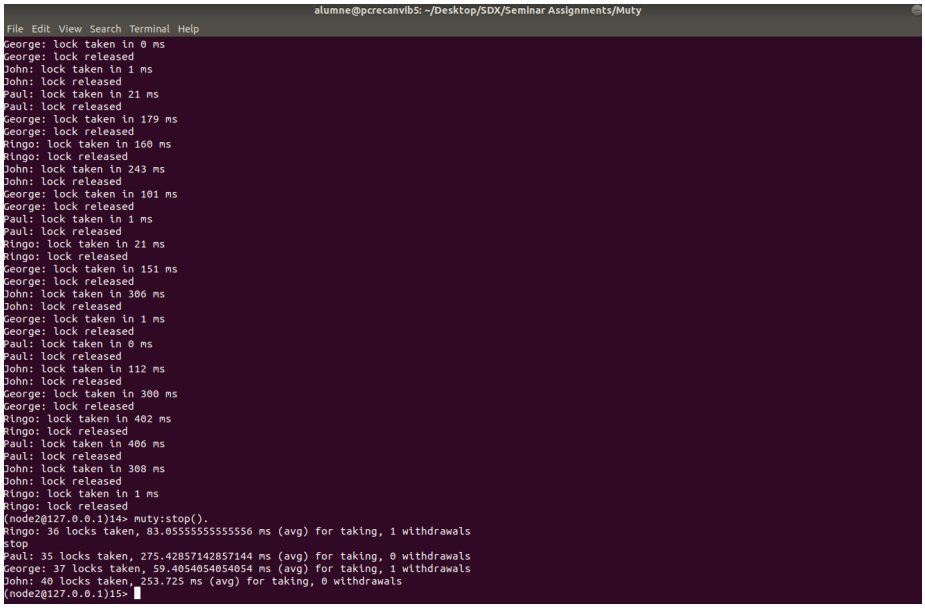


També ens podem trobar amb la situació de que el temps de work, sigui més gran que el withdrawal time, per tant els workers a la espera superarien aquet temps. Tal i com podem observar:



Test 2: Sleep time superior a work time

Si el temps de sleep, és més gran que el temps de work, estarem provocant, que les peticions d'accés a la regió critica dels diferents workers, estigui més espatllada, i per tant la possibilitat de trobar-nos en una situació de deadlock serà més baixa. Encara que no es una solució definitiva, ja que encara que essent menys probable també ens podem trobar amb una situació de deadlock, tal i com podem observar en la següent imatge:



Test 3: Sleep time i work time iguals

En aquesta situació, a la llarga també ens podriem trobar amb deadlocks, encara que seran improbables com quan el sleep time es menor al work time. Ho podem observar en la següent imatge:

```
alumne@pcrecanvib5: ~/Desktop/SDX/Seminar Assignments/Muty
File  Edit  View  Search  Terminal  Help
George: lock released
John: lock taken in 658 ms
John: lock released
Paul: lock taken in 673 ms
Paul: lock released
Ringo: lock taken in 434 ms
Ringo: lock released
George: lock taken in 300 ms
George: lock released
John: lock taken in 39 ms
John: lock released
Ringo: lock taken in 432 ms
Ringo: lock released
Paul: lock taken in 467 ms
Paul: lock released
George: lock taken in 740 ms
George: lock released
Ringo: lock taken in 582 ms
Ringo: lock released
John: lock taken in 676 ms
John: lock released
Paul: lock taken in 587 ms
Paul: lock released
George: lock taken in 846 ms
George: lock released
Ringo: lock taken in 508 ms
Ringo: lock released
John: lock taken in 614 ms
John: lock released
Paul: lock taken in 605 ms
Paul: lock released
George: lock taken in 654 ms
George: lock released
Ringo: lock taken in 675 ms
Ringo: lock released
John: lock taken in 349 ms
John: lock released
Paul: lock taken in 240 ms
(node2@127.0.0.1)22> muty:stop().
Ringo: 29 locks taken, 527.4827586206897 ms (avg) for taking, 0 withdrawals
stop
John: 30 locks taken, 462.2 ms (avg) for taking, 0 withdrawals
Paul: 28 locks taken, 493.5 ms (avg) for taking, 0 withdrawals
George: 27 locks taken, 558.0370370370371 ms (avg) for taking, 0 withdrawals
(node2@127.0.0.1)23> canberra-gtk-module
```

2. Adapt the muty module to create each worker-lock pair in a different Erlang instance (that is, john and l1 should run in a node, ringo and l2 in another, and so on). Remember how processes are created remotely, how names registered in remote nodes are referred, and how Erlang runtime should be started to run distributed programs.

```
-module(muty).
-export([start/3, stop/0]).

% We use the name of the module (i.e. lock3) as a parameter to the start procedure. We also provide the average time (in millisecond

start(Lock, Sleep, Work) ->
    spawn('node1@127.0.0.1',fun()->register(l1, apply(Lock, start, [1])),register(w1, worker:start("John", l1, Sleep, Work))end),
    spawn('node2@127.0.0.1',fun()->register(l2, apply(Lock, start, [2])),register(w2, worker:start("Ringo", l2, Sleep, Work))end),
    spawn('node3@127.0.0.1',fun()->register(l3, apply(Lock, start, [3])),register(w3, worker:start("Paul", l3, Sleep, Work))end),
    spawn('node4@127.0.0.1',fun()->register(l4, apply(Lock, start, [4])),register(w4, worker:start("George", l4, Sleep, Work))end),
    {l1,'node1@127.0.0.1'} ! {peers, [{l2,'node2@127.0.0.1'}, {l3,'node3@127.0.0.1'}, {l4,'node4@127.0.0.1'}]},
    {l2,'node2@127.0.0.1'} ! {peers, [{l1,'node1@127.0.0.1'}, {l3,'node3@127.0.0.1'}, {l4,'node4@127.0.0.1'}]},
    {l3,'node3@127.0.0.1'} ! {peers, [{l1,'node1@127.0.0.1'}, {l2,'node2@127.0.0.1'}, {l4,'node4@127.0.0.1'}]},
    {l4,'node4@127.0.0.1'} ! {peers, [{l1,'node1@127.0.0.1'}, {l2,'node2@127.0.0.1'}, {l3,'node3@127.0.0.1'}]},
    ok.

stop() ->
    {w1,'node1@127.0.0.1'} ! stop,
    {w2,'node2@127.0.0.1'} ! stop,
    {w3,'node3@127.0.0.1'} ! stop,
    {w4,'node4@127.0.0.1'} ! stop.
```

Resolving deadlock

1. Repeat the previous tests to compare the behavior of this lock with respect to the previous one.

   Test 1: Sleep time inferior a work time

   Test 2: Sleep time superior a work time

Lamport's time

1. Repeat the previous tests to compare this version with the former ones.

   Test 1: Sleep time inferior a work time

   Test 2: Sleep time superior a work time

3 Open questions

4 Personal opinion