

# Muty: a distributed mutual-exclusion lock

Fèlix Andrés Navarro  
Jordi Calatayud Àlvarez

# Index

|                   |          |
|-------------------|----------|
| <b>Index</b>      | <b>2</b> |
| <b>Lock1</b>      | <b>3</b> |
| Experiments       | 3        |
| Preguntas obertes | 6        |
| <b>Lock2</b>      | <b>6</b> |
| Experiments       | 6        |
| Preguntas obertes | 7        |
| <b>Lock3</b>      | <b>8</b> |
| Experiments       | 8        |
| Preguntas obertes | 10       |

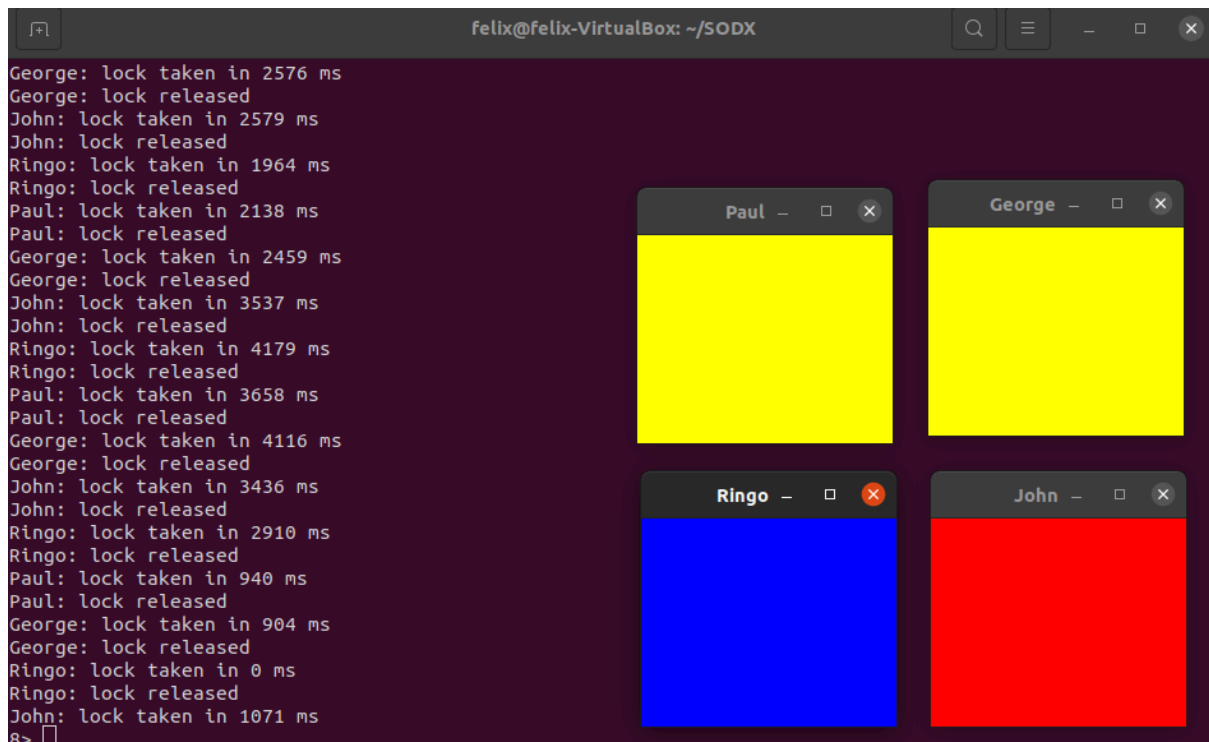
# Lock1

## Experiments

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

Primer provem amb els paràmetres 1000 i 2000

`muty:start(lock1, 1000, 2000).`



```
felix@felix-VirtualBox: ~/SODX
George: lock taken in 2576 ms
George: lock released
John: lock taken in 2579 ms
John: lock released
Ringo: lock taken in 1964 ms
Ringo: lock released
Paul: lock taken in 2138 ms
Paul: lock released
George: lock taken in 2459 ms
George: lock released
John: lock taken in 3537 ms
John: lock released
Ringo: lock taken in 4179 ms
Ringo: lock released
Paul: lock taken in 3658 ms
Paul: lock released
George: lock taken in 4116 ms
George: lock released
John: lock taken in 3436 ms
John: lock released
Ringo: lock taken in 2910 ms
Ringo: lock released
Paul: lock taken in 940 ms
Paul: lock released
George: lock taken in 904 ms
George: lock released
Ringo: lock taken in 0 ms
Ringo: lock released
John: lock taken in 1071 ms
8>
```

Observem els resultats:

```
8> muty:stop().
John: 28 locks taken, 2753.464285714286 ms (avg) for taking, 0 withdrawals
Ringo: 28 locks taken, 2652.8928571428573 ms (avg) for taking, 0 withdrawals
Paul: 28 locks taken, 2868.285714285714 ms (avg) for taking, 0 withdrawals
George: 27 locks taken, 2629.074074074074 ms (avg) for taking, 0 withdrawals
stop
9>
```

Després provem amb els paràmetres 3000 i 3000:

```
13> muty:start(lock1, 3000, 3000).
ok
```

Observem els resultats:

```
14> muty:stop().
John: 21 locks taken, 2171.714285714286 ms (avg) for taking, 0 withdrawals
Ringo: 21 locks taken, 2345.7619047619046 ms (avg) for taking, 0 withdrawals
Paul: 19 locks taken, 2836.5789473684213 ms (avg) for taking, 0 withdrawals
George: 20 locks taken, 2889.0 ms (avg) for taking, 0 withdrawals
stop
15> █
```

I finalment provem amb els paràmetres 5000 i 5000:

```
15> muty:start(lock1, 5000, 5000).
ok
```

Observem els resultats:

```
16> muty:stop().
John: 11 locks taken, 4336.272727272727 ms (avg) for taking, 1 withdrawals
Ringo: 11 locks taken, 3693.0 ms (avg) for taking, 1 withdrawals
Paul: 10 locks taken, 5405.4 ms (avg) for taking, 1 withdrawals
George: 10 locks taken, 4278.5 ms (avg) for taking, 2 withdrawals
stop
17> █
```

2. Split the muty module and make the needed adaptations to enable each worker-lock pair to run in different machines (that is, john and l1 should run in one machine, ringo and l2 in another, and so on). Remember how names registered in remote nodes are referred and how Erlang runtime should be started to run distributed programs.

Separem el codi de mutty en 4 codis: muty\_distr1, muty\_distr2, muty\_distr3, muty\_distr4. Cadascun per a un worker diferent.

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

% We use the name of the module (i.e. lock3) as a parameter to the start

start(Lock, Sleep, Work) ->
    register(l1, apply(Lock, start, [1])),
    l1 ! {peers, [l2, l3, l4]},
    register(w1, worker:start("John", l1, Sleep, Work)),
    ok.

stop() ->
    w1 ! stop,
```

muty\_distr1

```

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

% We use the name of the module (i.e. lock3) as a parameter to the start function

start(Lock, Sleep, Work) ->
    register(l2, apply(Lock, start, [2])),
    l2 ! {peers, [l1, l3, l4]},
    register(w2, worker:start("Ringo", l2, Sleep, Work)),
    ok.

stop() ->
    w2 ! stop,

```

muty\_distr2

```

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

% We use the name of the module (i.e. lock3) as a parameter to the start function

start(Lock, Sleep, Work) ->
    register(l3, apply(Lock, start, [3])),
    l3 ! {peers, [l1, l2, l4]},
    register(w3, worker:start("Paul", l3, Sleep, Work)),
    ok.

stop() ->
    w3 ! stop,

```

muty\_distr3

```

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

% We use the name of the module (i.e. lock3) as a parameter to the start function

start(Lock, Sleep, Work) ->
    register(l4, apply(Lock, start, [4])),
    l4 ! {peers, [l1, l2, l3]},
    register(w4, worker:start("George", l4, Sleep, Work)),
    ok.

stop() ->
    w4 ! stop.

```

muty\_distr4

Ara per a executar el codi de manera distribuïda arrancarem Erlang de la següent forma:

```
erl -name worker1@ip-adress1 -setcookie worker
```

I d'aquesta forma amb la resta de màquines. Després hem de connectar les i aquesta part la farem de d'una sola màquina.

```
net_adm_ping('worker2@ip-adress2').
```

Una vegada estan connectades les màquines a la principal ja podem fer cridades de manera remota a la resta de la següent manera.

```
rpc:call('worker2@ip-adress2',muty,start,[Lock, Sleep, Work]).
```

## Preguntes obertes

Quino es el comportament del lock quan incrementa el risc de conflicte?

En lock1, quan el risc de conflicte és alt (és a dir, amb una contenció elevada), és probable que es produeixin interbloqueos, ja que no hi ha un mecanisme per prioritzar les sol·licituds. Això provoca un augment en la quantitat de retirades i uns temps d'espera llargs, perquè els treballadors queden atrapats esperant respostes que no es resolen. Així, el sistema es torna poc eficient i alguns treballadors poden bloquejar-se indefinidament si no s'implementen mecanismes de sortida d'aquest interbloqueig.

## Lock2

## Experiments

Primer provem amb els paràmetres 1000 i 2000

```
muty:start(lock2, 1000, 2000).
```



Observem els resultats:

```
10> muty:stop().
John: 0 locks taken, 0 ms (avg) for taking, 11 withdrawals
Ringo: 0 locks taken, 0 ms (avg) for taking, 11 withdrawals
Paul: 0 locks taken, 0 ms (avg) for taking, 11 withdrawals
George: 0 locks taken, 0 ms (avg) for taking, 11 withdrawals
stop
11> 
```

Després provem amb els paràmetres 3000 i 3000:

```
17> muty:start(lock2, 3000, 3000).
ok
```

Observem els resultats:

```
18> muty:stop().
John: 0 locks taken, 0 ms (avg) for taking, 9 withdrawals
Ringo: 0 locks taken, 0 ms (avg) for taking, 9 withdrawals
Paul: 0 locks taken, 0 ms (avg) for taking, 9 withdrawals
George: 0 locks taken, 0 ms (avg) for taking, 9 withdrawals
stop
19> 
```

I finalment provem amb els paràmetres 5000 i 5000:

```
3> muty:start(lock2, 5000, 5000).
ok
```

Observem els resultats:

```
20> muty:stop().
John: 0 locks taken, 0 ms (avg) for taking, 5 withdrawals
Ringo: 0 locks taken, 0 ms (avg) for taking, 5 withdrawals
Paul: 0 locks taken, 0 ms (avg) for taking, 5 withdrawals
George: 0 locks taken, 0 ms (avg) for taking, 4 withdrawals
stop
21> 
```

## Preguntes obertes

1. Justifica com el teu codi garanteix que només un procés és a la secció crítica a la mateixa vegada

En lock2, el codi garanteix l'exclusió mútua usant prioritats basades en l'ID de cada instància de bloqueig. Quan una instància està esperant respostes però rep una sol·licitud

d'una altra instància amb un ID més alt, aquesta li envia immediatament un missatge ok, evitant així l'interbloqueig. D'aquesta manera, cada treballador rep accés exclusiu a la secció crítica, garantint que només un treballador pot entrar-hi a cada moment. Aquest sistema permet evitar els interbloquejos, però ho fa a costa d'una jerarquia fixa que pot afectar l'equitat entre els treballadors.

## 2. Quin és el major desavantatge de la implementació del lock2?

La principal desavantatge de lock2 és que la prioritat està rígida i exclusivament basada en l'ID de cada instància. Això pot generar situacions d'“injustícia”, on treballadors amb IDs més baixos tinguin menys probabilitats d'accedir a la secció crítica en comparació amb aquells amb IDs més alts, independentment de l'ordre en què s'hagin fet les sol·licituds. Això pot portar a una distribució desigual dels accessos a la secció crítica, especialment en sistemes amb moltes instàncies competint per recursos limitats.

# Lock3

## Experiments

Repeteix els test previs per a comparar el comportament d'aquest lock en comparació al anterior.

Primer provem amb els paràmetres 1000 i 2000

```
muty:start(lock3, 1000, 2000).
```



```

11> muty:start(lock3, 1000, 2000).
ok
John: lock taken in 0 ms
John: lock released
Ringo: lock taken in 1233 ms
Ringo: lock released
Paul: lock taken in 2025 ms
Paul: lock released
George: lock taken in 2718 ms
George: lock released
John: lock taken in 1828 ms
John: lock released
Ringo: lock taken in 2324 ms
Ringo: lock released
Paul: lock taken in 2791 ms
Paul: lock released
George: lock taken in 2911 ms
George: lock released
John: lock taken in 1432 ms
John: lock released
Ringo: lock taken in 1481 ms
Ringo: lock released
Paul: lock taken in 2269 ms
Paul: lock released
George: lock taken in 2459 ms
George: lock released
John: lock taken in 2252 ms
John: lock released
Ringo: lock taken in 2928 ms
Ringo: lock released

```



Observem els resultats:

```

12> muty:stop().
John: 21 locks taken, 2445.190476190476 ms (avg) for taking, 0 withdrawals
Ringo: 21 locks taken, 2455.285714285714 ms (avg) for taking, 0 withdrawals
Paul: 21 locks taken, 2690.2380952380954 ms (avg) for taking, 0 withdrawals
George: 21 locks taken, 2592.8571428571427 ms (avg) for taking, 0 withdrawals
stop
13>

```

Després provem amb els paràmetres 3000 i 3000

```

21> muty:start(lock3, 3000, 3000).
ok

```

Observem els resultats:

```

22> muty:stop().
stop
Ringo: 14 locks taken, 2286.0 ms (avg) for taking, 0 withdrawals
John: 13 locks taken, 2113.6153846153848 ms (avg) for taking, 0 withdrawals
Paul: 12 locks taken, 2364.25 ms (avg) for taking, 0 withdrawals
George: 12 locks taken, 2785.0833333333335 ms (avg) for taking, 0 withdrawals
23>

```

I finalment provem amb els paràmetres 5000 i 5000

```

23> muty:start(lock3, 5000, 5000).
ok

```

Observem els resultats:

```
24> muty:stop().  
John: 7 locks taken, 4706.285714285715 ms (avg) for taking, 2 withdrawals  
Ringo: 9 locks taken, 4547.555555555556 ms (avg) for taking, 0 withdrawals  
Paul: 8 locks taken, 3985.0 ms (avg) for taking, 1 withdrawals  
George: 5 locks taken, 4544.6 ms (avg) for taking, 3 withdrawals  
stop  
25> |
```

## Preguntes obertes

Fixa't que els worker no estan involucrats en el Lamport clock. Tenint en compte esto, seria possible que un worker tinga access a una zona crítica prèviament a que altre worker hagi enviat una request a la instancia del lock abans(assumint real-time order)?

Sí, és possible que un treballador accedeixi a la secció crítica abans que un altre, fins i tot si aquest altre va fer la sol·licitud abans en temps real. Això passa perquè l'ordre d'accés en lock3 depèn del rellotge lògic de Lamport, no del temps real. Els rellotges de Lamport mantenen un ordre consistent dins del sistema distribuït, però aquest ordre no sempre coincideix amb l'ordre cronològic real de les sol·licituds. Així, es pot donar el cas que, dins d'un sistema distribuït, els esdeveniments es processin en una seqüència lògica que prioritzi algunes sol·licituds en funció de l'estat del rellotge lògic en lloc de l'ordre temporal real, cosa que pot afectar la percepció d'equitat en l'accés a la secció crítica.