Seminar Report: Mutty

Marc Ortiz
Omair Iqbal
Víctor González

March 22, 2017

1 Introduction

Aquesta practica consisteix en implementar diferents versions d'un lock d'exclusió mutua. El lock està basat en l'algorisme de Ricart Agrawala. De fet, la tercera versió és una implementació de l'algorisme (amb rellotges de Lamport i desempat amb identificador de procès.)

Un lock d'exclusió mútua serveix per coordinar l'accés a un recurs compartit, de manera que dos processos no pugin accedir al recurs alhora.

2 Work done

Els fitxers font es troben a la carpeta codes i les imatges la carpets figures per si no es veuen prou bé.

3 Experiments

3.1 The architecture

Per la primera part es dona el codi de la primera implementació de l'algorisme a partir del qual podem fer diversos experiments per veure com reacciona el programa amb diferents temps de sleep i work.

1. *Experiment amb sleep time molt més petit que work time*

   Hipòtesis: Si el temps de sleep es molt més petit que el temps de work, llavors dos o més workers poden tenir més probabilitats de demanar accés a la regió crítica en el mateix temps, fent que aquest workers estiguin esperant els missatges de OK del altres workers que també a la vegada demanen accés a aquesta regió, això provoca un deadlock que s'allibera 8 segons després amb posibilitat de repetir el procès.

Demostració:

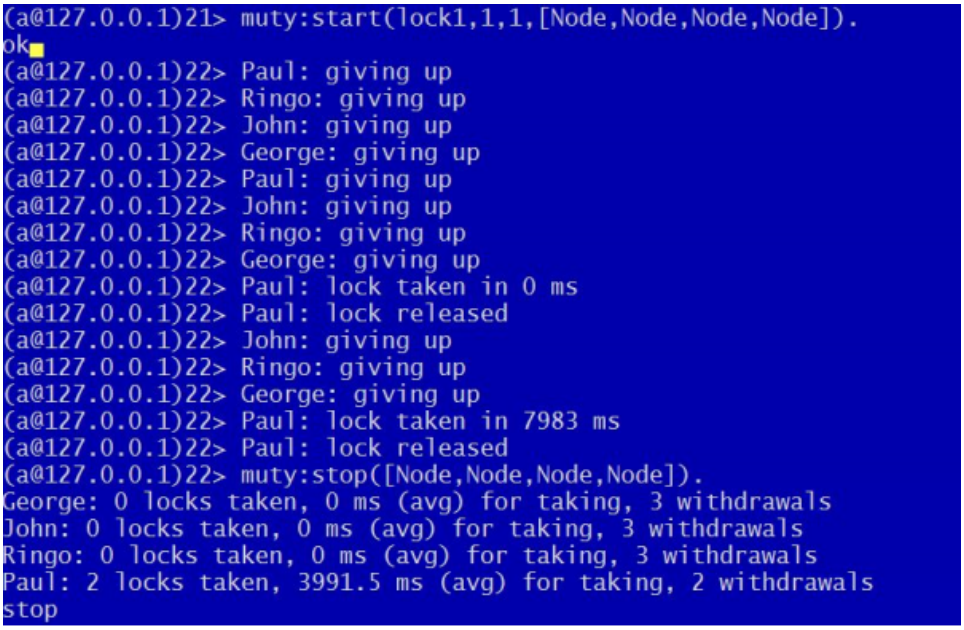

Figure 1: Es produeix un deadlock

2. *Experiment amb Sleep time major i mateix worktime*

   Hipòtesis: Si el temps de sleep es major que el de worktime, els workers demanaran acces a la regio critica amb intervals de temps més separats disminuint la probabilitat de deadlocks però això no significa que no desapareixin.

   Demostració:

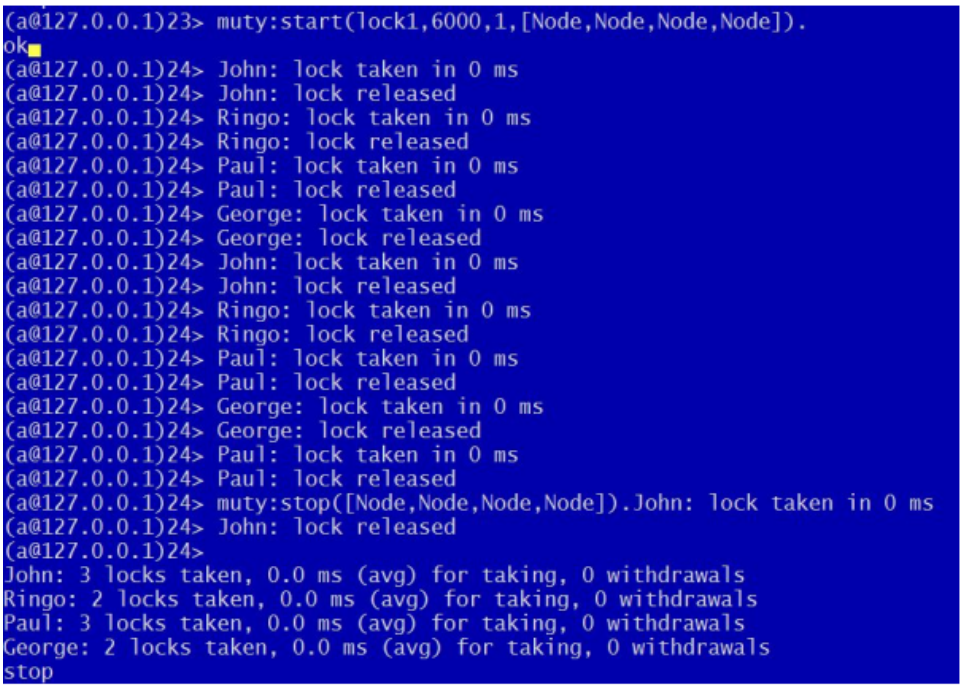

Figure 2: Els workers no tenen problemes per sincronitzar l'acces

3. *Experiment amb work time molt més gran que sleep time*

   Hipòtesis: Hi pot haver un punt en que degut a la implemetació del nostre worker, que amb un parametre de work prou elevat provocaria que la espera dels altres workers sigui més elevada que el parametre widthdrawal. Aquest parametre pot ser per exemple un work time igual o més gran al widthdrawal evidentment, o fins i tot amb un worktime més gran que widhtdrawal/4 pot provocar-ho en els pitjors dels casos.

Demostració:

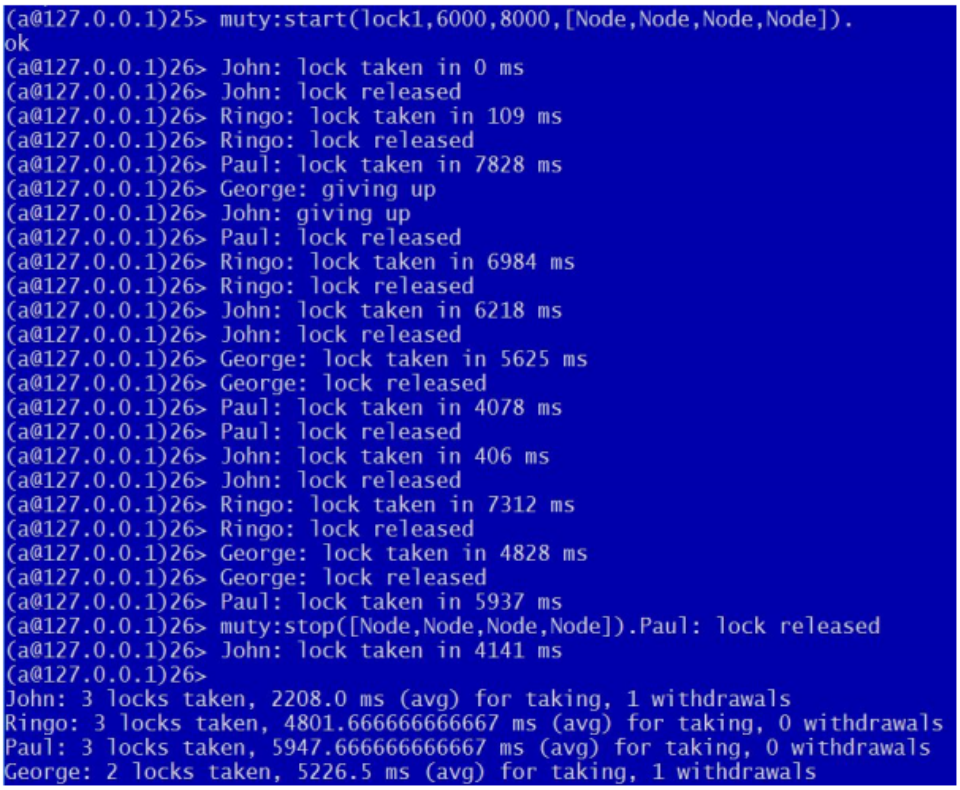

Figure 3: John i en George han fet el request, han esperat, i degut a que el temps d'espera ha superat els 8 segons, han alliberat el lock.

3.2 Resolving deadlock

En aquesta segona part es una fa millora per a que cada instancia de lock tingui un id per a que quan dos workers accedeixen a la regió critica, el que tingui el id més petit es que aconseguira l'acces.

1. *Experiment amb sleep time molt més petit que work time*

   Hipotesis: dos nodes demanen accés a la regio critica, el node amb l'identificador més petit (mes prioritari) rep el missatge de OK per part del meny prioritari evitant així deadlocks. Una conseqüencia d'això es que la reparticio del control de la regio critica no sigui equitatiu.

Demostració:

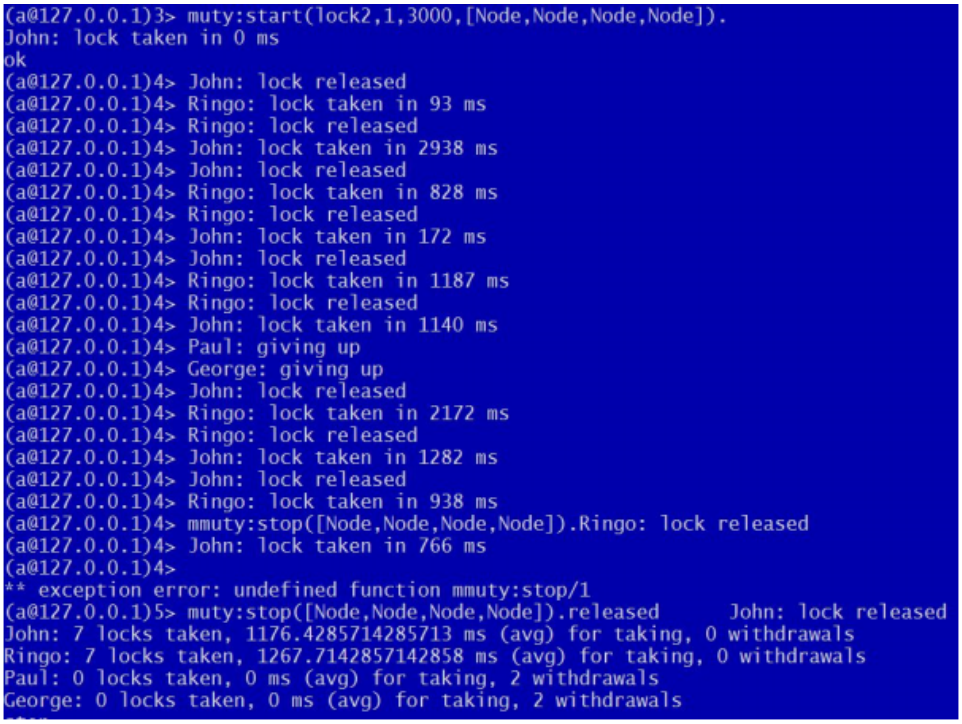

Figure 4: Versio millorada per evitar deadlocks

Com poden veure a la figura 4, aquest cop no hi ha hagut deadlocks, però el problema que té aquesta implementacio es que els dos locks amb id més petit (mes prioritaris) es reparteixen l'accés a la regió critica. En aquest cas el sleep time que es 1 provoca que tots els workers demanin el lock al mateix temps. El lock 1 guanya (John) mentres que el lock 2 te les confirmacions de tots menys del lock 1. Quan el lock 1 acaba confirma a el lock 2 (Ringo) que entra a la regio critica, el lock 1 que just despres de sortir de la regio critica torna a fer un request per accedir-hi i al ser el més prioritari de tots torna a accedir deixant als altres a la espera.

2. *Experiment amb Sleep time major i mateix work time*

   Hipotesis: Quan el temps de espera es més gran, augmenten les posibilitat de que un node amb el id menys prioritari agafi el lock i s'eviti la situació de l'experiment anterior.

   Demostració: En aquest cas George i en Paul han aconseguit més locks

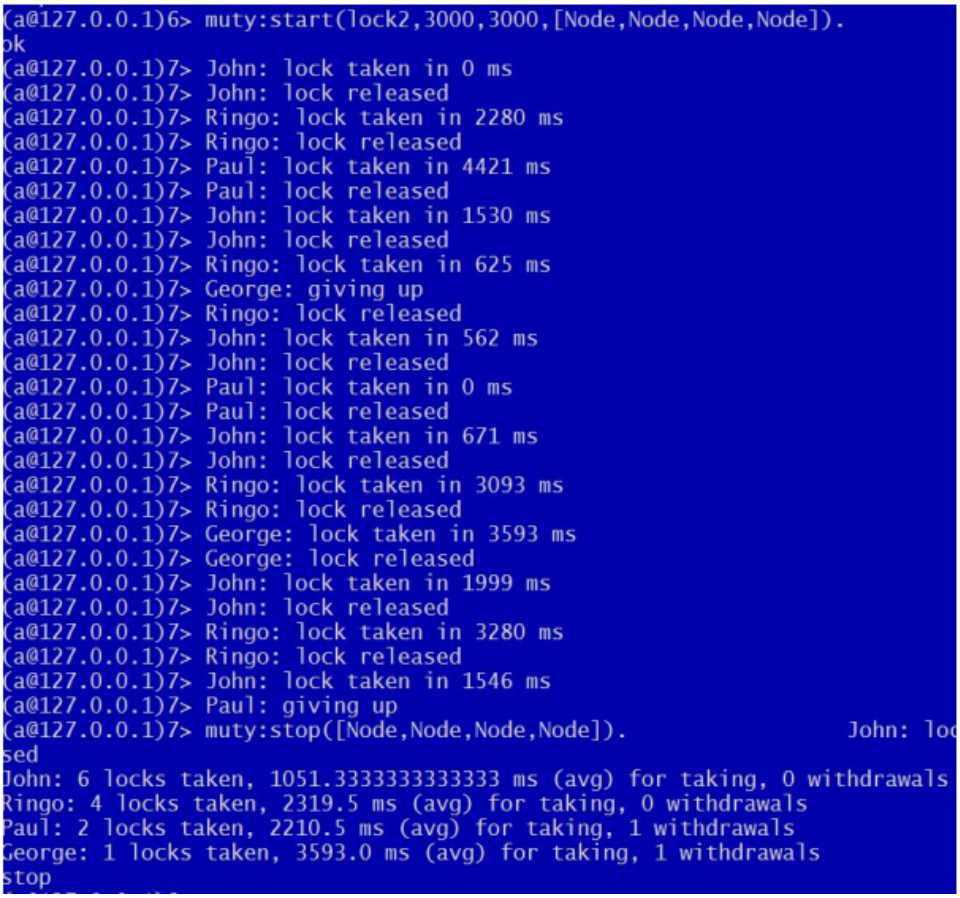

Figure 5: Els nodes amb menys prioritat aconseguir accedir a la regió critica

   degut a l'increment del sleep time. Aquest increment ha permes que en George sigui l'unic demanat el lock en un moment concret (que es l'unica manera que el pot aconseguir). Tambe veiem que el temps en agafar els locks esta relacionat amb el identificador del node, de manera que John al ser el més prioritari ha pogut agafar més locks.

3. *Experiment amb Work time molt més elevat que sleeptime*

   En aquest cas el resultat es el mateix que el de la implementacio anterior, pero es pot donar el cas on si el work time es molt elevat els

locks rebin un missatge de release del worker.

3.3 Lamport time

En aquesta ultima implementacio hem afegit als locks Lamport logical clocks, per tant tenin una variable inicialitzada a 0 a cada instancia de lock i que es incrementada cada vegada que el lock accedeix a la regió critica, d'aquesta manera els locks s'agafen per prioritat de temps, el lock amb el clock més petit te més prioritat.

1. *Experiment amb sleep time molt més petit que work time*

   Hipotesis: quan dos nodes demanen accés a la regio critica, el node amb el clock més petit (mes prioritari) rep el missatge de OK per part del menys prioritari i accedeix a la regió critica. Pero si el temps de sleep es molt més petit que el de work (10 i 10000) llavors workers poden enviar missatges de release.

   Demostració:

Figure 6: Els workers no tenen problemes en sincronitzar l'acces amb sleep time 100 i work time 1000

   Com es pot observar en la figura 6, els nodes menys prioritaris poden accedir a la regió critica sense cap problema, cosa que no pasaba en la implementacio anterior.

2. *Experiment amb Sleep time major que work time*

   Hipotesis: Els nodes accedeixen per ordre de temps per tant la distribucio dels locks es més uniforme.

   Demostració:

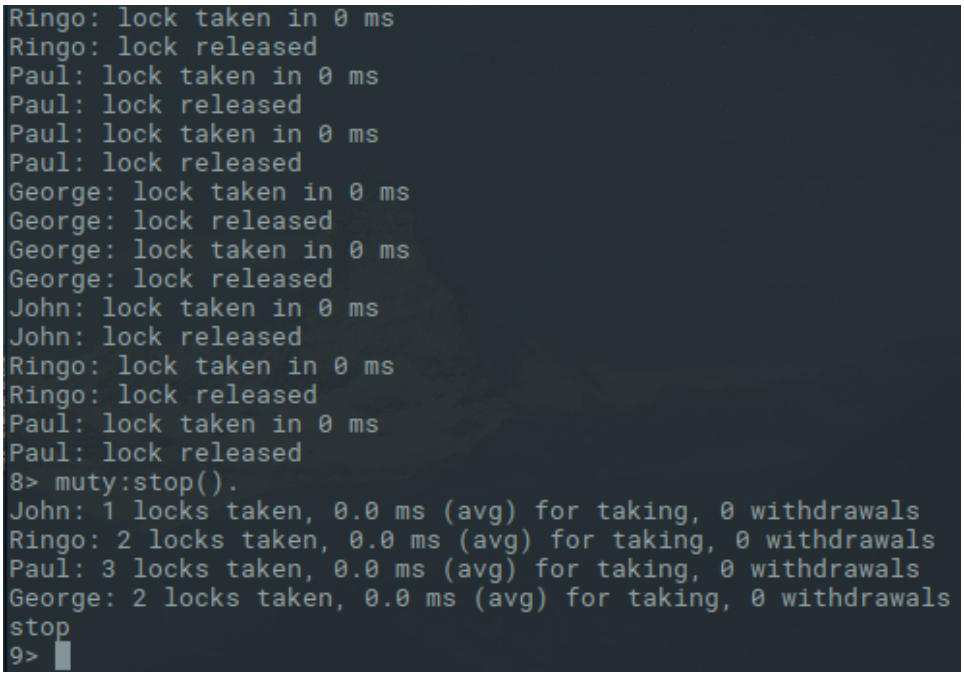

Figure 7: Els workers no tenen problemes en sincronitzar l'acces amb sleep time 1000 i work time 10

3. *Experiment amb work time molt elevar comparat amb sleep time*

   Hipotesis: Hi pot haber un punt en que un parametre del work time provocaria que la espera dels altres workers sigui més elevada que el parametre widthdrawal.

Demostració:

Figure 8: Hi ha locks no aconsegueixen accedir a la regió critica, sleep time 10 i work time 10000

Hi ha locks que no poden accedir a la regió critica per que el parametre del work es més elevat que el de widthdrawal, per tant el worker corresponent envia un missatge de release.

4 Open questions

4.1 The architecture

1. *What is the behavior of the lock when you increase the risk of a conflict?*

Si considerem que es pot produir un deadlock, llavors dos o més locks poden enviar requests a la vegada entrant en la funcio o estat del lock wait, encuant les requests dels altres. Els locks es quedarien infinitament esperant les confirmacions. Per seguretat, els locks quen es produeix un deadlock reben un missatge de release del worker si al cap d'un temps no aconsegueixen el lock. En aquesta versio del lock es preserva la safetyness degut a que no hi ha manera de que dos locks accedeixin a la regió crítica a la vegada, però no asegura la liveness degut a que com hem vist i explicat es poden produir deadlocks.

4.2 The architecture

1. *Justify how your code guarantees that only one process is in the critical section at any time.*

   Per aconseguir que dos locks aconsegueixein l'acces a la regio critica al mateix temps cal que els dos rebin el missatge de OK per part de tots els altres. Hi ha un escenari que esdeve quan un node de menys prioritat demana l'OK als altres, un dels nodes amb més prioritat li respon amb un OK i acte seguit li fa una request per el control de la regió crítica. El node menys prioritari rebrà l'OK i, si encara no ha rebut l'OK dels altres nodes, rebrà la request del node prioritari, cosa que li respondrà amb un OK. Per evitar aquest cas quan un node rep una request d'un node més prioritari li envia l'OK i seguidament li envia una request al més prioritari d'aquesta manera evitem l'escenari anterior i assegurem que el node prioritari s'executarà després. Degut a això aquesta versió de lock no respecta el happened-before order. Tampoc es respecta el liveness degut a que com hem vist per a valors molt petits de sleep no s'assegura que els nodes menys prioritaris aconsegueixin el control de la regió crítica en algun moment. Si es respecta el safetyness.

2. *What is the main drawback of lock2 implementation*

   El principal problema que pot sortir es que degut a la prioritat d'alguns nodes el control de la regió critica no es ben repartit, i que quan el sleep time es molt petit llavors els nodes amb id més petits i que demanin acces sempre tindran més prioritat. Això com s'ha comentat en l'experiment anterior es pot donar que alguns nodes no aconsegueixin mai la regió crítica o també que el temps d'accés de nodes menys prioritari sigui molt elevat.

4.3 Lamport Time

1. *Note that the workers are not involved in the Lamport clock. According to this, would it be possible that a worker is given access to a critical section prior to another worker that issued a request to its lock instance before (assuming real-time order)?*

   No, ja que l'acces es donat en funcio de la variable clock que es incrementada cada vegada que el lock accedeix a la regió critica i que els locks amb clock més petit tenen més prioritat.

5 Personal opinion

Aquest lab ha sigut molt util per tenir una idea general d'un mecanisme de sincronitzacio aixi com poder veure els problemes associats i haber pogut posat en practica els coneixements teorics adquirits.