Seminar Report: Muty

April 3, 2019

Carles Tornel
Jesus Alfaro
Ricard Abril

1 Introduction

En aquesta pràctica el que volem és implementar un servei multicast que respecti diferents ordres, com FIFO, causal o total.

2 Experiments

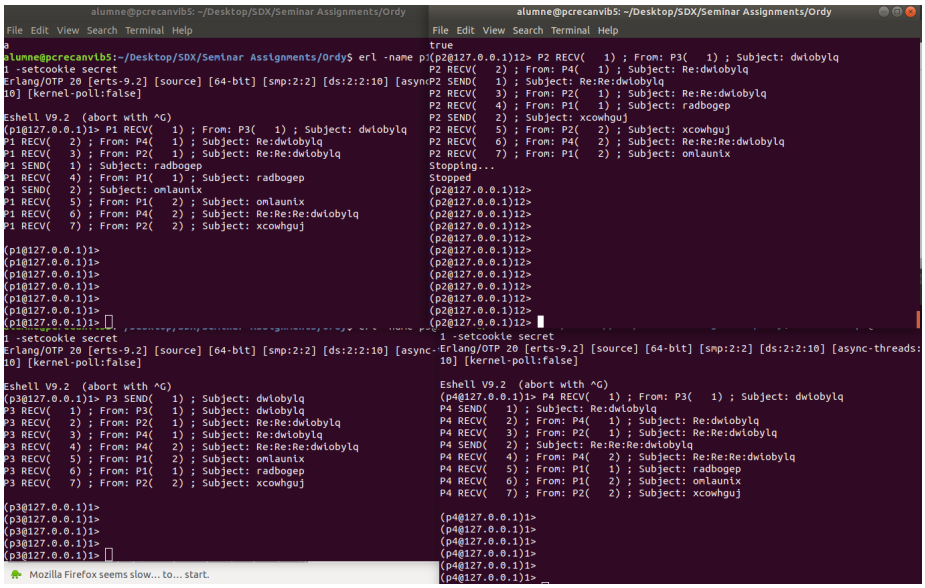
Basic Multicast

1. Set up the basic multicast system, and use the following test program to experiment with different values for Sleep and Jitter. Sleep stands for up to how many milliseconds the workers should wait until the next message is sent. Jitter stands for up to how many milliseconds the messages are delayed in the network. Duration stands for the duration of the experiment in milliseconds. Note that we are using the name of the multicast module (i.e. basic) as a parameter to the start procedure. We will easily be able to test different multicast implementations. Note also that processes are spawned in different nodes, hence you need to start an Erlang runtime for each process and name it accordingly.

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

Test 1: Sleep time major que Jitter time

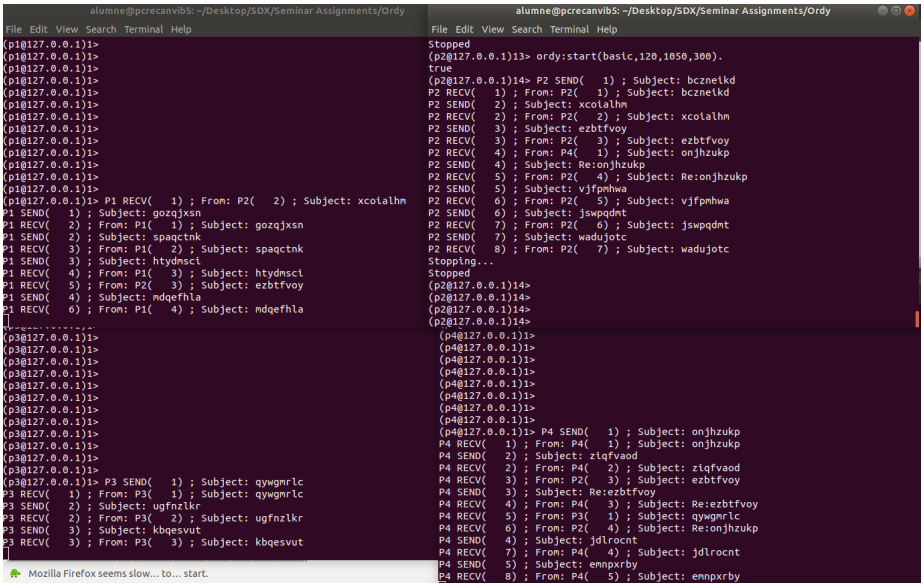
La probabilitat de que un missatge sigui enviat metres estem fent multicast, serà més baixa, ja que és més probable que la resta de nodes es trobin en Sleep time.



Com podem observar a la figura, hi ha diversos missatges, en el que si que es respecta l'ordre FIFI, però també es pot observar com en alguna ocasió els missatges no arriben en ordre.

Test 2: Sleep time inferior a Jitter time

En aquet cas, és més probable que l'ordre dels missatges es vegi alterat, ja que en el moment en que estem fent multicast, es més probable que hi hagin missatges a ser enviats, de forma que la probabilitat de que un missatge arribi en desordre és major



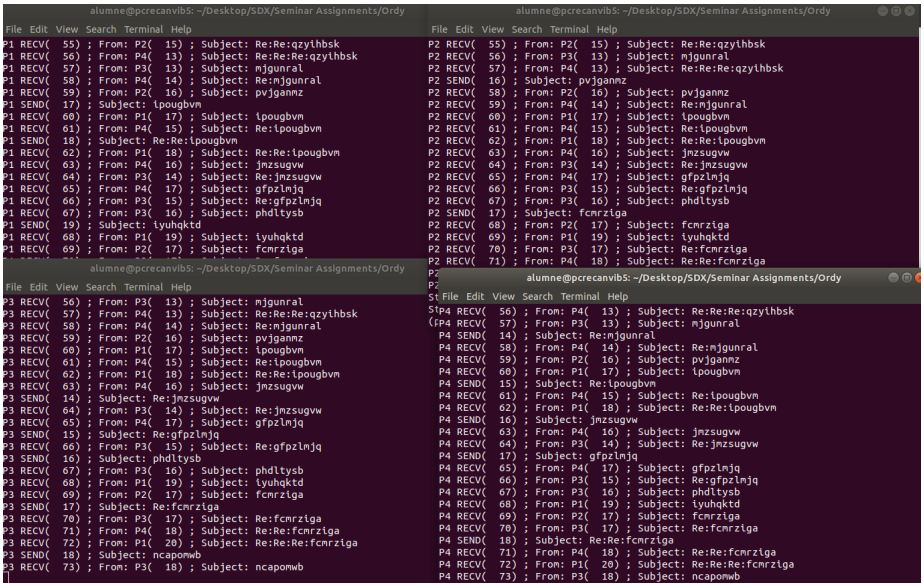
Com podem veure, ara és més fàcil observar desordres en els missatges, degut a que en el període en el que fem multicast, s'estan enviant més missatges.

Causal order multicast

1. Set up the causal order multicast system, and repeat the previous tests.

Test 1: Sleep time inferior a Jitter time

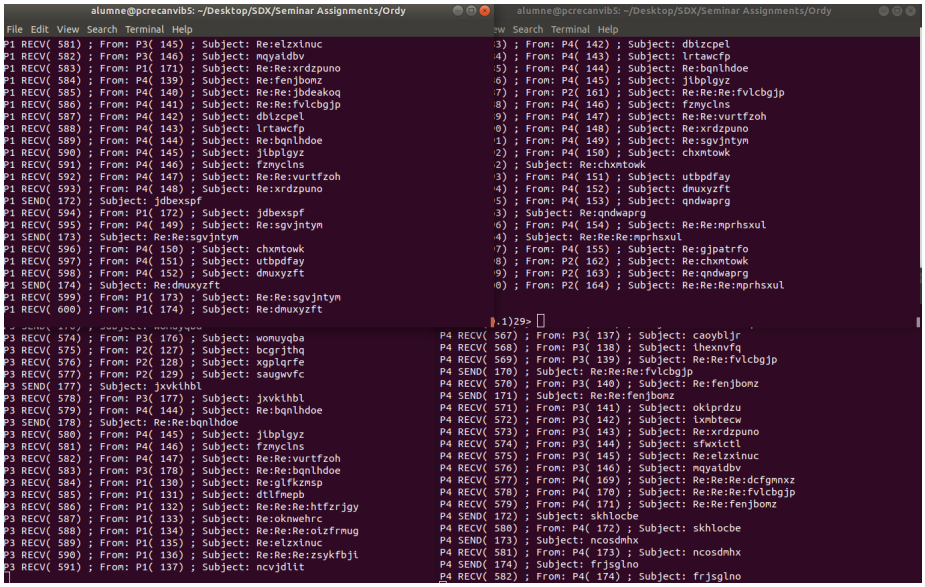
Amb el Causal order, haurien de arribar els missatges en l'ordre correcte, ja que comprova que el missatge que s'està rebent és el esperat i que s'han rebut tots els anteriors del procés que l'ha enviat.



Com podem veure, el ordre s'ha mantes, es a dir no hem rebut ningun missatge no esperat, es a dir que el missatges del emissor arriben en ordre i que no hi ha cap missatge de resposta enviat abans del primer, però el ordre total no s'ha mantes.

Test 2: Sleep time superior a Jitter time

El cas de tenir un sleep time menor al Jitter time, no hauria de causar ninguna mena de desordre, en tot cas el que provocarà, és que els missatges triguin més a ser rebuts, ja que els processos estaran en sleep time més estona, com podem veure en la figura.

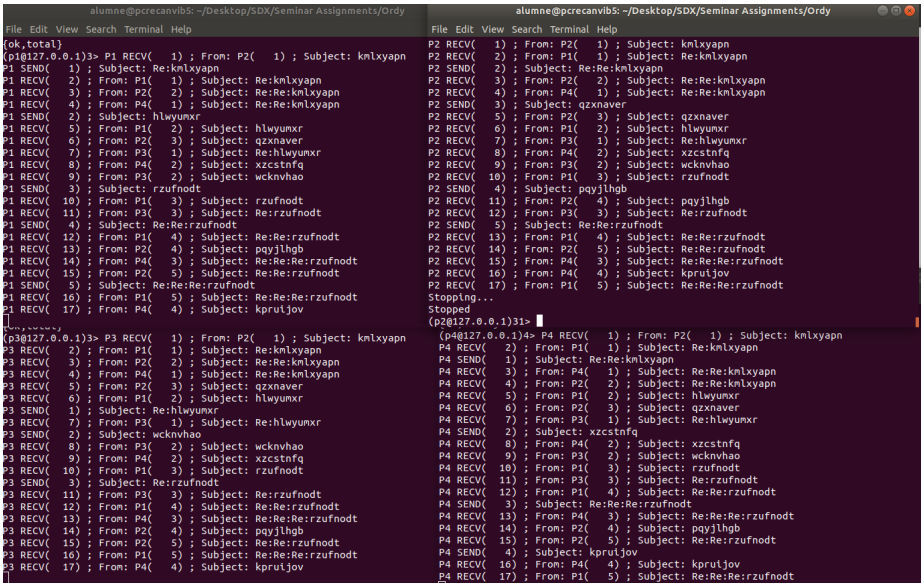


Total order multicast

1. Set up the total order multicast system, and repeat the previous tests.

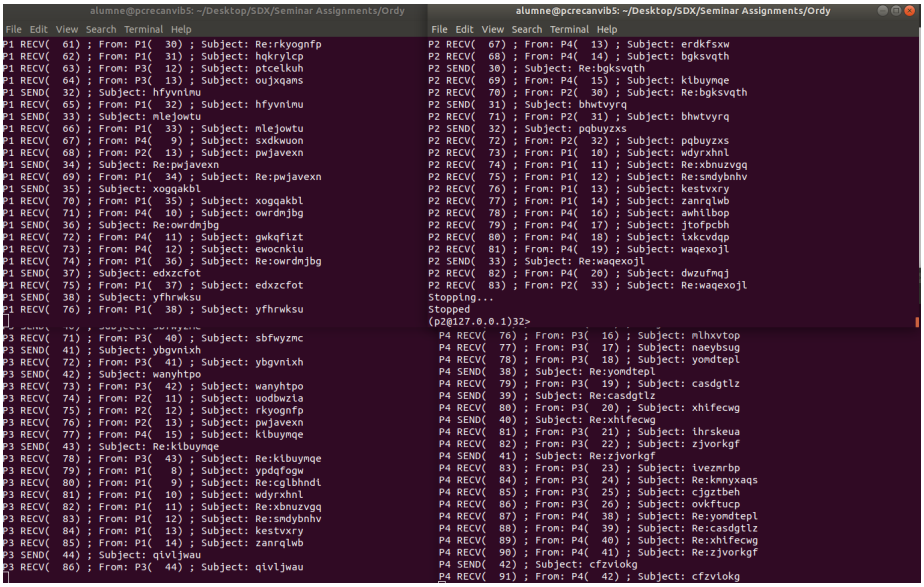
Test 1: Sleep time inferior a Jitter time

Ni el Sleep time ni el Jitter time, haurien de provocar cap desordre, ja que els missatges s'entregaran en ordre de mínim nSeq de la queue. El que si que notarem, es que degut al alt temps de Jitter, s'enviaran més missatges.



Test 2: Sleep time superior a Jitter time

Igual que en el cas anterior, tots els processos rebran els missatges en el mateix ordre, però degut a que el Sleep time, és més alt el numero de missatges rebuts serà menor, tal i com podem veure:



2. We have a lot of messages in the system. Derive a theoretical quantification of the number of messages needed to deliver a multicast message as a function of the number of workers and check exper- imentally that your formulation is correct

   Basic = 1 * send + n * multicast + n * deliver = 2 * n + 1.

   Total = 1 * send + n * request + n * proposal + n * agreed + n * deliver = 4 * n + 1.

   N = numero de workers.

3 Open questions

1. Basic Multicast: Are the posts displayed in FIFO, causal, and totalorder? Justify why.
   Cap de les opcions. En aquesta versió bàsica utilitzem la funció send-after que fa servir el Jitter per treure un temps de retard random, per tant, si tenim el cas d'un worker que té temps de resposta (o d'enviar un altre missatge multicast) prou petit, llavors aquest segon worker enviarà abans que el primer worker, els missatges multicast.

2. Casual Order Multicast: Are the posts displayed in FIFO, causal, and totalorder? Justify why.
   El vector clock fa que hi hagi una relació happened-before entre els missatges, per tant, es respectarà l'ordre causa. FIFO també ja que 2 missatges enviats pel mateix procés s'hauran de lliurar segons ho indiqui el vector clock. Però pel contrari no es respecta l'ordre total, ja que no hi ha una relació entre missatges de diferents procesos enviats al mateix temps. En aquesta versió el paràmetre Jitter perd importancia a causa de l'ordre que implica els vectors clock i la cua.

3. Total Order Multicast: Are the posts displayed in FIFO, causal, and totalorder? Justify why.
   L'Ordre FIFO no es compleix ja que la posició de cada missatge depèn del jitter amb que arriba als processos. Aquests no tenen un ordre intern dels missatges que emeten a diferència de la versió causal.
   L'ordre causal tampoc es compleix. No s'ha establert relació de happened-before entre missatges.
   L'ordre total si que es respecta perquè tots els processos reben la mateixa sequència de missatges.

4 Personal opinion

El lab practicat ens ha ajudat a clarificar els conceptes explicats a teoria i ha exemplificat de manera clara un dels protocols en més èxit que s'utilitza actualment.