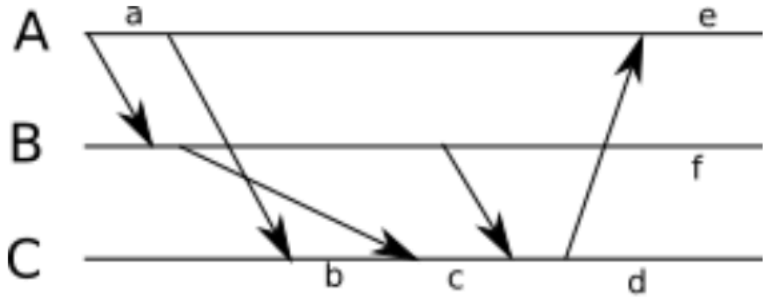
COMP 360 — Homework 4
Jeff Epstein October 8, 2016

Fabien Bessez

Questions

1. The diagram shows messaging events between three machines. Give the state of the given machine's vector clock at points a through e. Each point corresponds to time between message events. Assume that all vector clocks starts at (0, 0, 0).



The vector clocks:
a: (1,0,0); b: (2, 0, 1); c: (2, 1, 2); d: (2, 2, 4) ; e: (3, 2, 4); f: (1, 2, 0)

2. Consider the following two threads, executed in parallel. Assume that all variables are initialized to 0.

Listing 1: Thread 1
x = 1
y = x + 1
Listing 2: Thread 2
y = 4
x = y * 2

(a) Enumerate all the possible final values of x and y, given arbitrary interleavings of these instructions.
Let's assume that "x = 1" is on line 1, "y = x + 1" is on line 2, "y = 4" is on line 3 and "x = y * 2" is on line 4. In that case, there are 6 different combinations of orderings: 1234, 1324, 1342, 3412, 3124, and 3142. In the 1234 ordering, the final values are y = 4 and x = 8. In the 1324 ordering, the final values are y = 2, x = 4. In the 1342 ordering, the final values are x = 8, y = 9. In the 3412 ordering, the final values are x = 1, y = 2. In the 3124 ordering, the final values are x = 4, y = 2. In the 3142 ordering, the final values are x = 8, y = 9.

(b) Now assume that the above threads are bracketing by locking instructions. That is, thread 1 acquires a lock before its first instruction and releases it after its last instruction; and thread 2 acquires the same lock before its first instruction and releases it after its last instruction. Which possible final results of the execution are possible, if any?
Assuming that thread 1 and thread 2 have agreed on a protocol for requesting locks in a particular order to avoid deadlocks, then there are two possible final results. Depending on which thread gets access to the locks first, then the final values are x = 1 and y = 2 or x = 8 and y = 4. However, if they happen to request locks in a disorganized way, there is a possibility that thread 1 requests a lock on x while thread 2 requests a lock on y. In this case, both threads will be waiting for the other thread to release their desired lock – resulting in deadlock.

3. Assume that a read/write lock (also known as a readers/writer lock) always gives access to waiting readers, if the lock is already held by at least one reader. Describe how this policy can lead to starvation.

In a world where a read/write lock always gives access to waiting readers, if the lock is already held by at least one reader, one could imagine a scenario when there is an un-ending stream of readers for the entirety of the lock's existence. Since a writer would never be granted ownership of the lock in that scenario, it can be said that that implementation can lead to starvation.

4. Discuss the factors that affect the performance of a system when considering fine-grained (smaller elements) versus coarse-grained (larger elements) locking.

When considering whether to implement fine-grained versus coarse-grained locking, it is important to understand the trade-offs in choosing one over the other. In choosing coarse-grained locking, you decrease the availability of the system, because whenever a lock is owned, other requests will have to wait. As a result, deadlocks are less likely. If fine-grained locking is chosen, then availability is increased but so is the likelihood of deadlock. It is important to know how many requesters there might be. If speed of a transaction is a concern, then it seems like a fine-grained locking system would be preferred since there will be less waiting time.