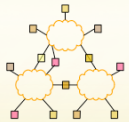
15-446 Distributed Systems Spring 2009



L-9 Logical Time

1

Announcements

- Project 1 update – Thursday
  - Due 2/26
- HW 1 – due Thursday

2

Last Lecture – Clock Sync Important Lessons

- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Two paths to solution: synchronize clocks or ensure consistent clocks
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source

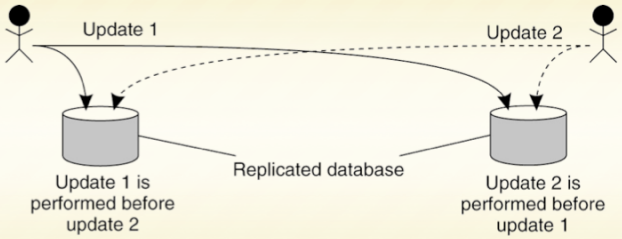
3

Today's Lecture

- Lamport Clocks
- Vector Clocks
- Mutual Exclusion
- Election

4

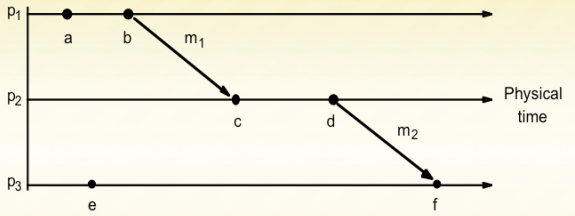
Example: Totally Ordered Multicasting



- Updating a replicated database and leaving it in an inconsistent state

5

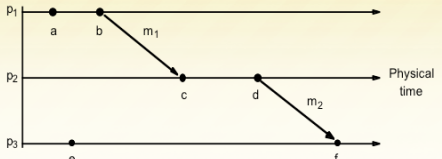
Logical time and logical clocks (Lamport 1978)



- Events at three processes

6

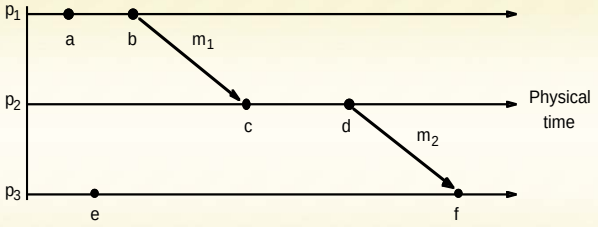
Logical time and logical clocks (Lamport 1978)



- Instead of synchronizing clocks, event ordering can be used
  1. If two events occurred at the same process pi (i = 1, 2, … N) then they occurred in the order observed by pi, that is $\rightarrow_i$
  2. when a message, m is sent between two processes, send(m) happened before receive(m)
  3. The happened before relation is transitive
- the happened before relation is the relation of causal ordering

7

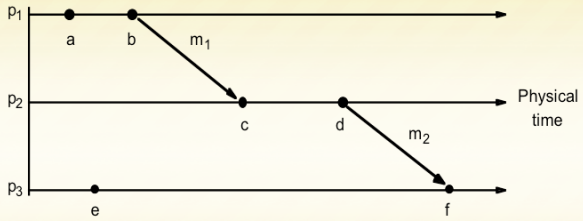
Logical time and logical clocks (Lamport 1978)



- $a \rightarrow b$ (at $p1$) $c \rightarrow d$ (at $p2$)
- $b \rightarrow c$ because of $m1$
- also $d \rightarrow f$ because of $m2$

8

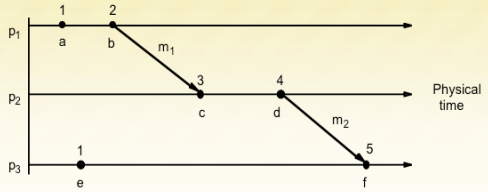
Logical time and logical clocks (Lamport 1978)



- Not all events are related by →
- Consider *a* and *e* (different processes and no chain of messages to relate them)
  - they are not related by → ; they are said to be concurrent
  - written as *a* || *e*

9

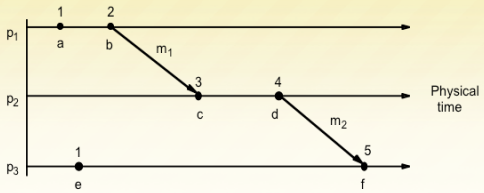
Lamport Clock (1)



- A logical clock is a monotonically increasing software counter
  - It need not relate to a physical clock.
- Each process p_i has a logical clock, L_i which can be used to apply logical timestamps to events
  - Rule 1: L_i is incremented by 1 before each event at process p_i
  - Rule 2:
    - (a) when process p_i sends message m, it piggybacks $t = L_i$
    - (b) when p_j receives (m,t) it sets $L_j := max(L_j, t)$ and applies LC1 before timestamping the event *receive* (m)

10

Lamport Clock (1)

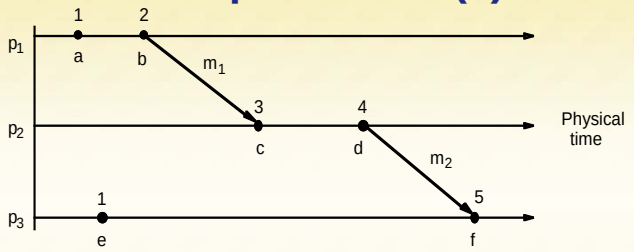


- each of p1, p2, p3 has its logical clock initialised to zero,
- the clock values are those immediately after the event.
- e.g. 1 for a, 2 for b.

- for *m*1, 2 is piggybacked and *c* gets *max*(0,2)+1 = 3

11

Lamport Clock (1)



- $e \rightarrow e'$ implies $L(e)<L(e')$

- The converse is not true, that is $L(e)<L(e')$ does not imply $e \rightarrow e'$
  - e.g. $L(b) > L(e)$ but b || e

12

Today's Lecture

- Lamport Clocks
- Vector Clocks
- Mutual Exclusion
- Election

13

Vector Clocks

- Vector clocks overcome the shortcoming of Lamport logical clocks
  - $L(e) < L(e')$ does not imply e happened before e'
- Vector timestamps are used to timestamp local events
- They are applied in schemes for replication of data

14

Vector Clocks

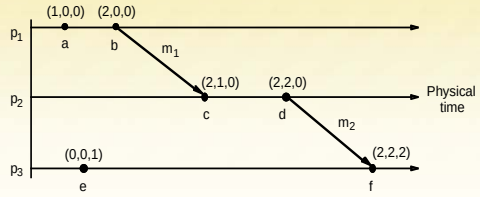
- $V_i[i]$ is the number of events that p_i has timestamped
- $V_i[j]$ $(j \neq i)$ is the number of events at p_j that p_i has been affected by

Vector clock V_i at process p_i is an array of N integers

1. initially $V_i[j] = 0$ for $i, j = 1, 2, \ldots N$
2. before p_i timestamps an event it sets $V_i[i] := V_i[i] + 1$
3. p_i piggybacks $t = V_i$ on every message it sends
4. when p_i receives (m,t) it sets $V_i[j] := \max(V_i[j], t[j])$ $j = 1, 2, \ldots N$ *(then before next event adds 1 to own element using rule 2)*

15

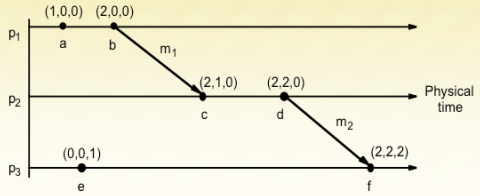
Vector Clocks



- At p_1
  - *a occurs at* (1,0,0); *b* occurs at (2,0,0)
  - piggyback (2,0,0) on m_1
- At p_2 on receipt of m_1 use *max* ((0,0,0), (2,0,0)) = (2, 0, 0) and add 1 to own element = (2,1,0)
- Meaning of =, <=, max etc for vector timestamps
  - compare elements pairwise

16

Vector Clocks



- Note that e →e' implies L(e)<L(e'). The converse is also true
- Can you see a pair of parallel events?
  - *c* || *e(* parallel) because neither *V*(c) <= *V*(*e*) nor *V*(*e*) <= *V*(*c*)

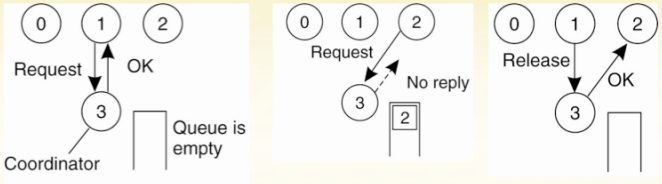
17

Today's Lecture

- Lamport Clocks
- Vector Clocks
- Mutual Exclusion
- Election

18

Mutual Exclusion
A Centralized Algorithm



1. Process 1 asks the coordinator for permission to access a shared resource → Permission is granted
2. Process 2 then asks permission to access the same resource → The coordinator does not reply
3. When process 1 releases the resource, it tells the coordinator, which then replies to 2

19

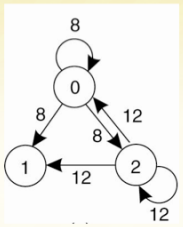
A Distributed Algorithm (1)

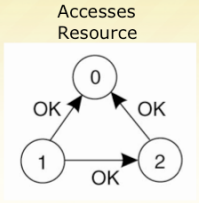
Three different cases:

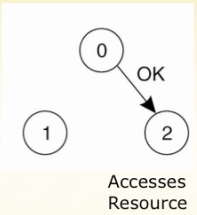
1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

20

A Distributed Algorithm (2)



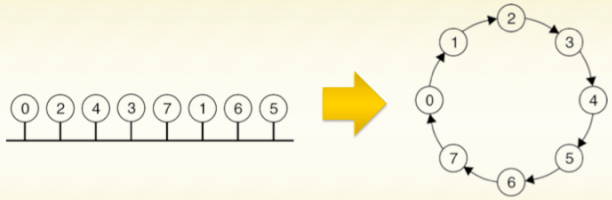




- Two processes want to access a shared resource at the same moment.
- Process 0 has the lowest timestamp, so it wins
- When process 0 is done, it sends an OK also, so 2 can now go ahead.

21

A Token Ring Algorithm



- An unordered group of processes on a network → A logical ring constructed in software
  - Use ring to pass right to access resource

22

A Comparison of the Four Algorithms

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	3mk, k = 1,2,...	2 m	Starvation, low efficiency
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

23

Today's Lecture

- Lamport Clocks
- Vector Clocks
- Mutual Exclusion
- Election

24

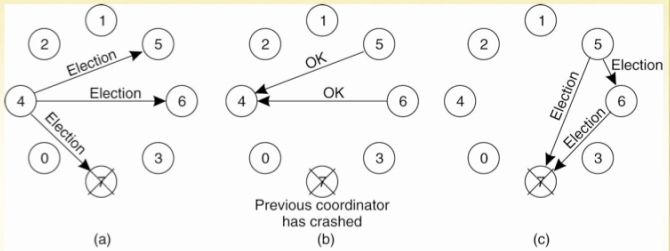
Election Algorithms

- The Bully Algorithm:

1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. P's job is done.

25

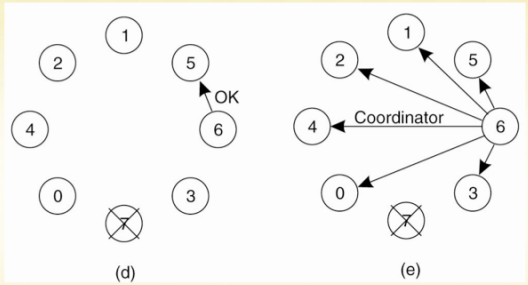
The Bully Algorithm (1)



- (a) Process 4 holds an election
- (b) Processes 5 and 6 respond, telling 4 to stop.
- (c) Now 5 and 6 each hold an election.

26

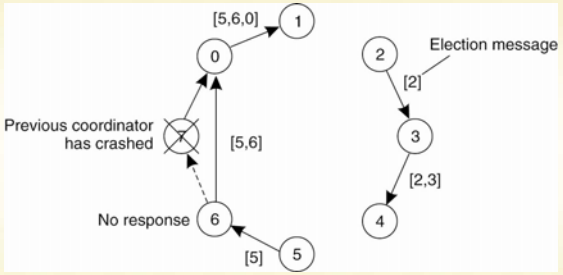
The Bully Algorithm (2)



- (d) Process 6 tells 5 to stop
- (e) Process 6 wins and tells everyone.

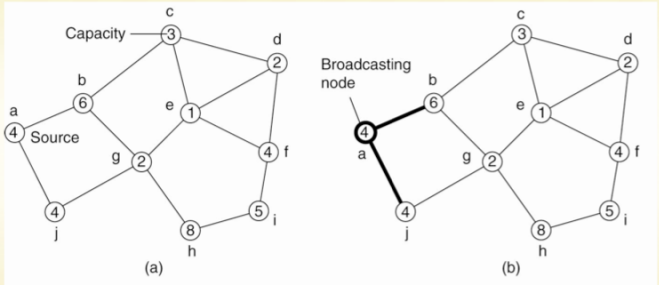
27

A Ring Algorithm



28

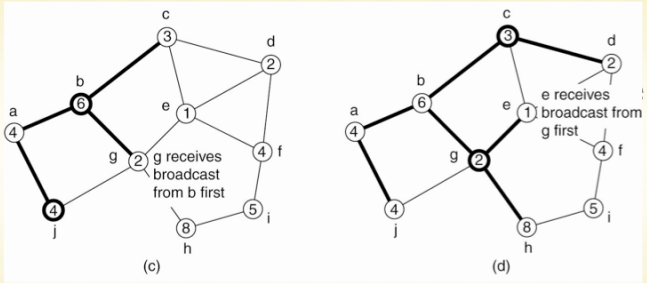
Elections in Wireless Environments (1)



- node a as the source
- The build-tree phase

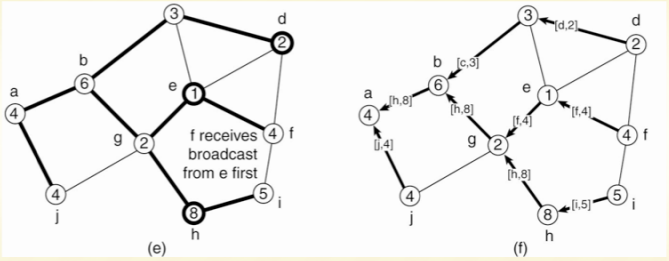
29

Elections in Wireless Environments (2)



30

Elections in Wireless Environments (3)



- Reporting of best node to source.

31

Important Lessons

- Lamport & vector clocks both give a logical timestamps
  - Total ordering vs. causal ordering
- Other issues in coordinating node activities
  - Exclusive access to resources
  - Choosing a single leader

32