


EEC-681/781

Distributed Computing Systems

Lecture 10


Wenbing Zhao
 Department of Electrical and Computer Engineering
 Cleveland State University
 wenbing@ieee.org



Outline

- Announcement:
 - Midterm #2 (lecture 6-10, lab 3) – Tuesday, 4-6pm, March 28, SH306
 - Move 04/06's lab session (lab4) to 04/04 6-7:50pm
- Global state
- Election
- Mutual exclusion


23 March 2006 EEC681/781 Wenbing Zhao



Vector Timestamps

- Clarification on causal delivery condition
 - To support causal delivery of messages. P_j postpones delivery of m until P_j is sure that all messages that m is causally dependent on have been delivered
 - $vt(m)[i] = V_j[i] + 1$
 - $vt(m)[k] \leq V_j[k]$ for $k \neq i$
- The condition is valid only under the following assumptions
 - The only type of events are message sending and receiving
 - All messages sent are multicast to all other processes, and
 - The multicast is reliable and FIFO

23 March 2006 EEC681/781 Wenbing Zhao



Global state

- Global state
 - A set of local states that are concurrent with each other, and
 - Channel state – reflect messages in transit
- Concurrent states: two states do not have a happens-before relation with each other

23 March 2006 EEC681/781 Wenbing Zhao

5

Mystery of the Missing Dollars

```

    graph LR
      A((A)) -- Send $100 --> B((B))
      A --- A_val[$400]
      B --- B_val[$300]
  
```

- Picture taken at A - \$400
- A sends \$100 to B
- Picture taken at B - \$400
- Total is \$800

23 March 2006 EEC681/781 Wenbing Zhao

6

Distributed Snapshot Problem

- Determine the global system state
 - e.g. the total amount of money
- Assumptions
 - Each process records its own state
 - No shared clock/memory
- Image that a group of photographers taking snapshots of different portions and trying to combine to get the overall picture

23 March 2006 EEC681/781 Wenbing Zhao

7

Distributed Snapshot

- A distributed snapshot reflects a state in which the distributed system *might* have been
- Reflects a consistent global state
 - If we have recorded that process P has received a message from another process Q, then we should also have recorded that process Q had actually *sent* the message
 - The reverse condition (Q has sent a message that P has not yet received) is allowed

23 March 2006 EEC681/781 Wenbing Zhao

8

Consistent Cut

- A **cut** represents the last event that has been recorded for each of several processes
 - All recorded message receptions have a corresponding recorded send event
- An inconsistent cut would have a receipt of a message but no corresponding send event

23 March 2006 EEC681/781 Wenbing Zhao

9

Consistent and inconsistent cuts

Question: which cut is a consistent cut?

23 March 2006 EEC681/781 Wenbing Zhao

10

Chandy and Lamport's Algorithm

- Assumes
 - FIFO and
 - Unidirectional, reliable channels
 - No process fails during the snapshot
- A bidirectional channel is modelled as two unidirectional channels
- Any process P can initiate taking a distributed snapshot

23 March 2006 EEC681/781 Wenbing Zhao

11

Chandy and Lamport's Algorithm

- P starts by recording its own local state
- P sends a marker along each of its outgoing channels
- When Q receives a marker through channel C , its action depends on whether it had already recorded its local state:
 - Not yet recorded:
 - ❖ It records its local state, and sends the marker along each of its outgoing channels
 - ❖ **It starts recording incoming messages on OTHER channels**
 - Already recorded: the marker on C indicates that the channel's state should be recorded:
 - ❖ All messages received before this marker and after Q recorded its own state

23 March 2006 EEC681/781 Wenbing Zhao

12

Chandy and Lamport's Algorithm

- Q is finished when it has received a marker along each of its incoming channels
- The recorded local state as well as the state it recorded for each incoming channel, can be collected and sent to the process that initiated the snapshot
- The global state can be subsequently constructed

23 March 2006 EEC681/781 Wenbing Zhao

13

Chandy and Lamport's Algorithm

Process Q receives a marker for the first time (from C1) and records its local state

Q records all incoming message on C2 (and other incoming channels except C1, if any)

Q receives a marker for its incoming channel C2 and finishes recording the state of the incoming channel C2

23 March 2006 EEC681/781 Wenbing Zhao

14

Applications

- Checkpointing of a distributed systems
 - Provide fault tolerance in distributed systems
 - Distributed debugging, e.g., detect deadlocks

23 March 2006 EEC681/781 Wenbing Zhao

15

Election Algorithms

- Many algorithms require that some process acts as a coordinator. The question is how to select this special process **dynamically**
- In many systems the coordinator is chosen by hand (e.g. file servers). This leads to centralized solutions with a single point of failure
- **Question:** If a coordinator is chosen dynamically, to what extent can we speak about a centralized or distributed solution?
- **Question:** Is a fully distributed solution, i.e. one without a coordinator, always more robust than any centralized/ coordinated solution?

23 March 2006 EEC681/781 Wenbing Zhao

16

Election by Bullying

- **Principle:** Each process has an associated priority (weight). The process with the highest priority should always be elected as the coordinator
- **Issue:** How do we find the heaviest process?
 - Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others)
 - If a process P_{heavy} receives an election message from a lighter process P_{light} , it sends a take-over message to P_{light} . P_{light} is out of the race
 - If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes

23 March 2006 EEC681/781 Wenbing Zhao

17

The Bully Algorithm

(a) Process 4 holds an election

(b) Previous coordinator has crashed
Process 5 and 6 respond, telling 4 to stop

(c) Now 5 and 6 each hold an election

23 March 2006 EEC681/781 Wenbing Zhao

18

The Bully Algorithm

(d) Process 6 tells 5 to stop

(e) Process 6 wins and tells everyone

23 March 2006 EEC681/781 Wenbing Zhao

19

Election in a Ring

- **Principle:** Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator
- **Ring Algorithm**
 - Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor
 - If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known
 - The initiator sends a coordinator message around the ring containing a list of all living processes
 - The one with the highest priority is elected as coordinator

23 March 2006 EEC681/781 Wenbing Zhao

20

A Ring Algorithm

Previous coordinator has crashed

No response

Election message

23 March 2006 EEC681/781 Wenbing Zhao

21

Mutual Exclusion

- **Problem:** A number of processes in a distributed system want exclusive access to some resource
- **Basic solutions:**
 - Via a centralized server
 - Completely distributed, with no topology imposed
 - Completely distributed, making use of a (logical) ring

23 March 2006 EEC681/781 Wenbing Zhao

22

Mutual Exclusion: A Centralized Algorithm

- Assumption
 - Messages are received reliably and in FIFO order
 - There exist a coordinator and it does not fail
 - ◆ The coordinator could be elected dynamically
- Algorithm
 - When a process wants to enter a critical region (CR), it sends a request to the coordinator
 - If no other process is in CR, the coordinator grants the request and sends back a reply
 - When the reply arrives, the requesting process enters CR
 - When a process leaves the CR, it notify the coordinator. If there is any queued request, the coordinator will reply to the oldest request

23 March 2006 EEC681/781 Wenbing Zhao

23

Mutual Exclusion: A Centralized Algorithm

(a)

Process 1 asks the coordinator for permission to enter a critical region. Permission is granted

(b)

Process 2 then asks permission to enter the same critical region. The coordinator does not reply

(c)

When process 1 exits the critical region, it tells the coordinator, when then replies to 2

23 March 2006 EEC681/781 Wenbing Zhao

24

Mutual Exclusion: A Centralized Algorithm

- Critique
 - **Single point of failure** - If the coordinator fails, no one will be able to enter the CR => a process cannot distinguish a dead coordinator from “permission denied” scenario
- How to fix the problem?
 - Any ideas?

23 March 2006 EEC681/781 Wenbing Zhao

Mutual Exclusion: A Distributed Algorithm

25

Assumption

- All messages are broadcast to every process reliably
- All messages are timestamped and there is a total order on them
- No process failure

Algorithm

- When a process wants to enter a critical region, it broadcasts a request
- When a process receives a request, it sends a reply only when
 - ❖ The receiving process has no interest in the shared resource; or
 - ❖ The receiving process is waiting for the resource, but has lower priority (known through comparison of timestamps).
- When a process gets reply from every other process, it enters the CR
- When a process leaves the CR, it sends the deferred replies to the queued requests

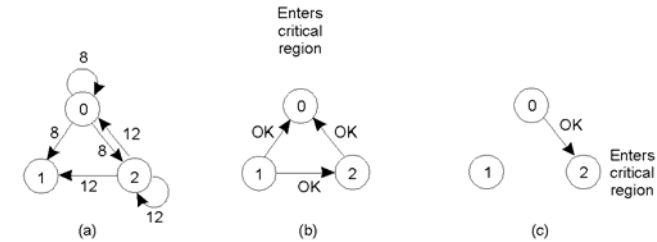
23 March 2006

EEEC681/781

Wenbing Zhao

Mutual Exclusion: A Distributed Algorithm

26



Two processes want to enter the same critical region 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 enter the critical region

23 March 2006

EEEC681/781

Wenbing Zhao

Mutual Exclusion: A Distributed Algorithm

27

Critique

- **N-point failure** - The algorithm fails if any of the processes fails
- Very inefficient – all processes are involved in all decisions
 - ❖ To enter CR, there are n requests and n replies, where n is the number of processes in the system
- Every process must maintain a correct membership
 - ❖ Who is in the system, who is not

Improvement?

23 March 2006

EEEC681/781

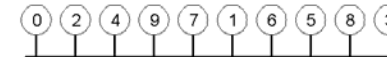
Wenbing Zhao

Mutual Exclusion: A Token Ring Algorithm

28

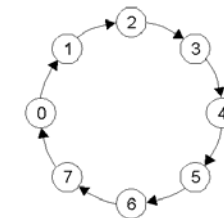
Assumption: no process failure and no message loss

- Organize processes in a *logical ring*, and let a token be passed between them.
- The one that holds the token is allowed to enter the critical region (if it wants to)



(a)

An unordered group of processes on a network



(b)

A logical ring constructed in software

23 March 2006

EEEC681/781

Wenbing Zhao



Mutual Exclusion: A Token Ring Algorithm

29

- Critique
 - If token is lost, the algorithm stops working
 - If a process fails, the algorithm also stops working
- Improvement
 - The token must be regenerated if lost – very difficult to do if processes might fail; otherwise using TCP would fix the problem
 - Process failure must be detected promptly
 - ❖ A process must acknowledge the receipt of a token
 - ❖ Every process must maintain a correct membership