

CSC 4103 - Operating Systems  
Spring 2008

LECTURE - XXII  
DISTRIBUTED SYSTEMS - I

Tevfik Koşar

Louisiana State University  
April 24<sup>th</sup> , 2008

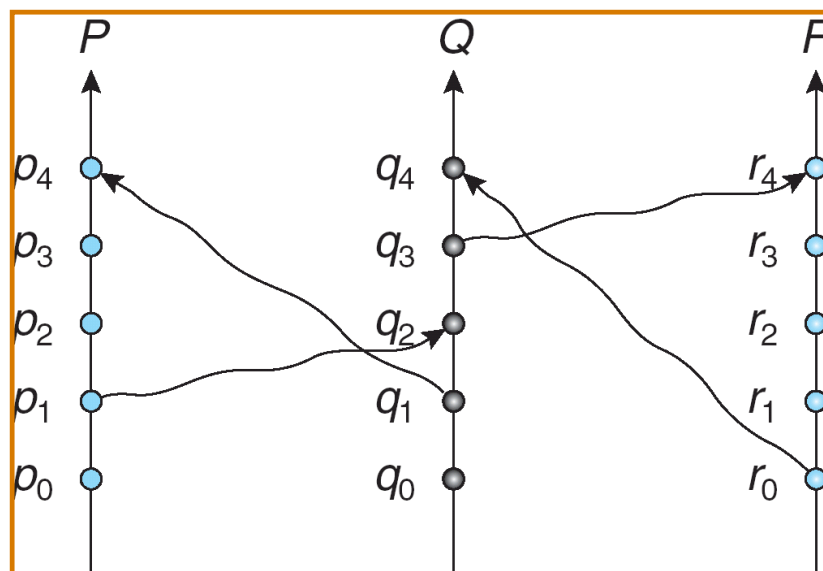
## Distributed Coordination

- Ordering events and achieving synchronization in centralized systems is easier.
  - We can use common clock and memory
- What about distributed systems?
  - No common clock or memory
  - *happened-before* relationship provides partial ordering
  - How to provide total ordering?

## Event Ordering

- **Happened-before** relation (denoted by  $\rightarrow$ )
  - If  $A$  and  $B$  are events in the same process (assuming sequential processes), and  $A$  was executed before  $B$ , then  $A \rightarrow B$
  - If  $A$  is the event of sending a message by one process and  $B$  is the event of receiving that message by another process, then  $A \rightarrow B$
  - If  $A \rightarrow B$  and  $B \rightarrow C$  then  $A \rightarrow C$
  - If two events  $A$  and  $B$  are not related by the  $\rightarrow$  relation, then these events are executed **concurrently**.

### Relative Time for Three Concurrent Processes



Which events are concurrent and which ones are ordered?

## Exercise

Which of the following event orderings are true?

- (a)  $p0 \rightarrow p3$  :
- (b)  $p1 \rightarrow q3$  :
- (c)  $q0 \rightarrow p3$  :
- (d)  $r0 \rightarrow p4$  :
- (e)  $p0 \rightarrow r4$  :

Which of the following statements are true?

- (a)  $p2$  and  $q2$  are concurrent processes.
- (b)  $q1$  and  $r1$  are concurrent processes.
- (c)  $p0$  and  $q3$  are concurrent processes.
- (d)  $r0$  and  $p0$  are concurrent processes.
- (e)  $r0$  and  $p4$  are concurrent processes.

## Implementation of $\rightarrow$

- Associate a timestamp with each system event
  - Require that for every pair of events A and B, if  $A \rightarrow B$ , then the timestamp of A is less than the timestamp of B
- Within each process  $P_i$ , define a **logical clock**
  - The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process
    - Logical clock is **monotonically increasing**
- A process advances its logical clock when it receives a message whose timestamp is greater than the current value of its logical clock
  - Assume A sends a message to B,  $LC_1(A)=200$ ,  $LC_2(B)=195$
- If the timestamps of two events A and B are the same, then the events are concurrent
  - We may use the process identity numbers to break ties and to create a total ordering

## Distributed Mutual Exclusion (DME)

- Assumptions
  - The system consists of  $n$  processes; each process  $P_i$  resides at a different processor
  - Each process has a critical section that requires mutual exclusion
- Requirement
  - If  $P_i$  is executing in its critical section, then no other process  $P_j$  is executing in its critical section
- We present two algorithms to ensure the mutual exclusion execution of processes in their critical sections

## DME: Centralized Approach

- One of the processes in the system is chosen to coordinate the entry to the critical section
- A process that wants to enter its critical section sends a request message to the coordinator
- The coordinator decides which process can enter the critical section next, and it sends that process a reply message
- When the process receives a reply message from the coordinator, it enters its critical section
- After exiting its critical section, the process sends a release message to the coordinator and proceeds with its execution
- This scheme requires three messages per critical-section entry:
  - request
  - reply
  - release

## DME: Fully Distributed Approach

- When process  $P_i$  wants to enter its critical section, it generates a new timestamp,  $TS$ , and sends the message *request* ( $P_i$ ,  $TS$ ) to all processes in the system
- When process  $P_j$  receives a *request* message, it may reply immediately or it may defer sending a reply back
- When process  $P_i$  receives a *reply* message from all other processes in the system, it can enter its critical section
- After exiting its critical section, the process sends *reply* messages to all its deferred requests

## DME: Fully Distributed Approach (Cont.)

- The decision whether process  $P_j$  replies immediately to a *request*( $P_i$ ,  $TS$ ) message or defers its reply is based on three factors:
  - If  $P_j$  is in its critical section, then it defers its reply to  $P_i$
  - If  $P_j$  does *not* want to enter its critical section, then it sends a *reply* immediately to  $P_i$
  - If  $P_j$  wants to enter its critical section but has not yet entered it, then it compares its own request timestamp with the timestamp  $TS$ 
    - If its own request timestamp is greater than  $TS$ , then it sends a *reply* immediately to  $P_i$  ( $P_i$  asked first)
    - Otherwise, the reply is deferred
  - Example: P1 sends a request to P2 and P3 (timestamp=10)  
P3 sends a request to P1 and P2 (timestamp=4)

## Undesirable Consequences

- The processes need to know the identity of all other processes in the system, which makes the dynamic addition and removal of processes more complex
- If one of the processes fails, then the entire scheme collapses
  - This can be dealt with by continuously monitoring the state of all the processes in the system, and notifying all processes if a process fails

## Token-Passing Approach

- Circulate a token among processes in system
  - **Token** is special type of message
  - Possession of token entitles holder to enter critical section
- Processes *logically* organized in a **ring structure**
- Unidirectional ring guarantees freedom from starvation
- Two types of failures
  - Lost token - election must be called
  - Failed processes - new logical ring established