Distributed Systems

So far:

- Communication - Networks
- Performance - Parallelism
- Applications
  - Programming models - RPC
  - Distributed File systems
  - Keeping Time - today
  - Synchronization - today
  - Global Knowlege
- Reliability
- Security

What time is it?

Problem: What does time mean in a distributed system.

Distributed Synchronization

Problem: Suppose we want to implement a semaphore mechanism for use in a distributed system.

Implementations:

- Centralized solution
- Distributed solution

Keeping Time

Problem: Define a total time ordering of all events in the system.

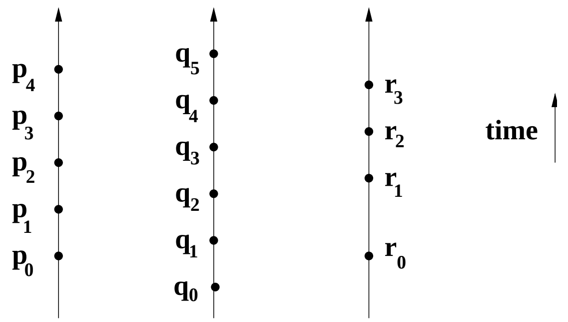
Stand-Alone System:

- Single clock, shared memory
- Total event ordering with a single processor is easy.

Distributed System:

- No global clock or memory, and the local clocks may be unsynchronized and run at different speeds.

$\Longrightarrow$ No order of events on different processors



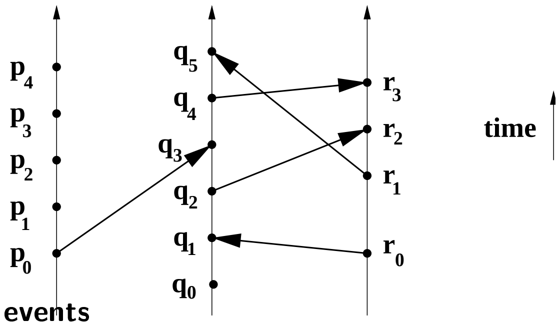
- How do we order events running on physically separated processors?

Key:

1. Through message communication.
2. A message must be sent before it can be received.
3. Send/Receives can thus "synchronize" their clocks.

Happened-Before Relation

1. If A and B are events in the same process, and A executed before B, then A $\rightarrow$ B.
2. If event A sends a message and event B receives that message, then A $\rightarrow$ B.
3. A $\rightarrow$ B and B $\rightarrow$ C, implies A $\rightarrow$ C.



Ordered events

- p_1 precedes
- q_4 precedes
- q_2 precedes
- p_0 precedes

Concurrent (unordered) events

- q_0 is concurrent with
- q_2 is concurrent with
- q_4 is concurrent with
- q_5 is concurrent with

Happened Before and Total Event Ordering

Goal: Define a notion of "time" of an event such that

1. if A $\rightarrow$ B, then "time" of A precedes "time" of B.
2. if A and B are concurrent events then "time" of A can precede B or vice versa.

Solution:

1. Each processor i maintains a logical clock lc_i.
2. Whenever an event occurs locally, $\mathrm{lc}_i = \mathrm{lc}_i + 1$.
3. When process X sends a message to Y, it also sends a *time stamp* lc_x on the message.
4. When Y receives this message from X:

   $\mathrm{lc}_y = \mathrm{lc}_y + 1$ (local event increment)
   if $\mathrm{lc}_y \leq \mathrm{lc}_x$
   then $\mathrm{lc}_y = \mathrm{lc}_x + 1$ (Y's clock was behind X's)
   else do nothing

Centralized Implementation of Distributed Semaphores

One known process in the system coordinates the *mutual exclusion*.

Worker processes:

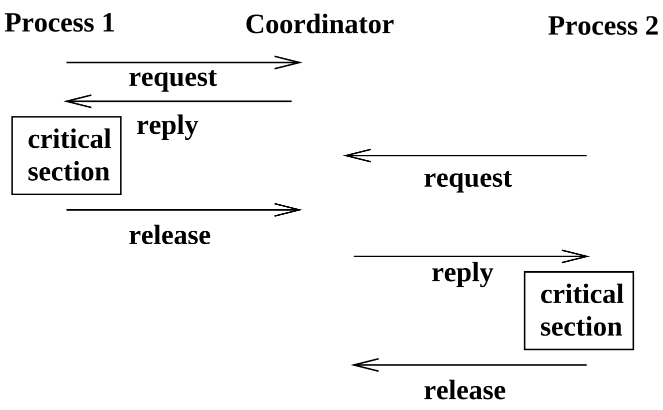
- To gain mutual exclusion, a process sends a *request* message to the controller, and waits for a *reply* message.
- When the processes gets a reply, it has the semaphore.
- When the process finishes, it sends a *release* message to the controller.

Controller:

Receives a request: if the semaphore is available and the queue is empty, then it sends a reply message. Otherwise, it queues the request message.

Receives a release: it removes a request from queue (FCFS priority) and sends the reply message.

Example: Centralized Implementation



- **Advantages:**

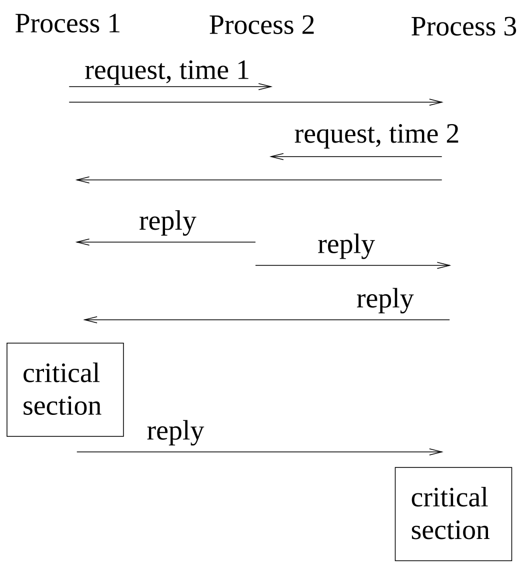
- **Disadvantages:**

Decentralized Implementation of Distributed Semaphores

Ricart & Agrawala 1981

- To achieve mutual exclusion, process k
  1. Generates a new time stamp ts_k.
  2. Sends request (k, ts_k) to all n-1 other processes.
  3. Waits for reply(j) from all n-1 processes.
  4. Enters critical section.
- On receipt of request (l, ts_l) from process l, process k
  1. Defers sending a reply if k is in the critical section,
  2. Else if k is not waiting to enter the critical section, k immediately sends reply(k) to l.
  3. Else if k is waiting and $ts_l < ts_k$, then k sends reply(k) to l.
  4. Else k defers the reply.
- On leaving a critical section, process k sends reply(k) to all deferred requests.

Example: Decentralized Implementation

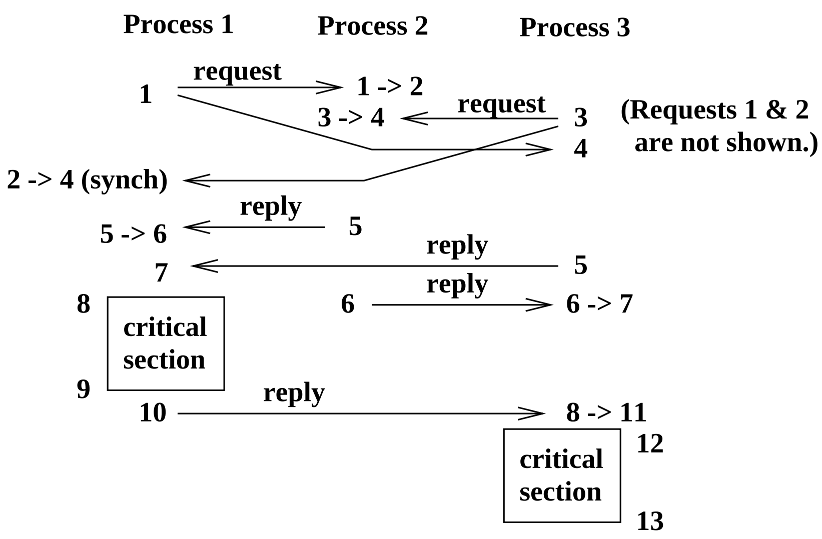


(Assume time2 happens between Process 1's request is sent and when it is received by Process 3)

Advantages:

Disadvantages:

Decentralized Semaphores with Total Event Ordering



Claim: Processes can agree on a total order for all events in the system.

- If $clock_i < clock_j$, then i $\rightarrow$ j.
- Define a total order on processes. If $clock_i = clock_j$, then i $\rightarrow$ j iff process i precedes process j.

What is the total order for our example assuming the processes are ordered 1, 2, 3?

Decentralized Semaphores with Token Passing

- A special token message is circulated among the processes in the system.
- Possession of the token implies the right to enter the critical section.
- After leaving the critical section, pass token to next process in a virtual ring of the participating processes.
- If a process gets the token and does not want to enter the critical section, it can
  1. immediately pass the token, or
  2. hold on to the token

Advantages:

Disadvantages:

Summary

- Global knowledge requires a lot of communication
- Partial knowledge in some cases is sufficient
- Centralization is problematic due to a single point of failure