Two Models

Time Order, Distributed Algorithms
Instructor: S. Bhalla
Chapters: Ch 11 Coulouris, Ch 6, 18 Silberschatz
1. LAN based systems (in Distributed Systems)

2. Web based systems

1. uses synchronous communications

2. uses Asynchronous communications

- Time-Stamp order ?

Other Algorithms
- Transaction Recovery ?
- System Recovery ?
• 1. Global time and event order (in Distributed Systems)

• 2. Distributed Mutual Excusion

• Concept of Transactions

• Concurrency Control
  - Locking
  - Validation
  - Time-Stamp order

• Atomicity
  - Transaction Recovery
  - System Recovery
Models → Implementation

Models

- [A] Fail-stop model: Atomicity
- [B] Fail-stop channel
- [C] Transaction model
  - 4 properties: isolation, consistency, atomicity, durability
- [D] Time-event ordering
- [E] Distributed Mutual Excusion

Lower level Implementation

- [A] 2 phase commit
- [B] Parity check
- [C] Transaction model implementation -
  - Data Locking, 2 phase commit, ...
- [D] Events and Numbering protocols
- [E] Centralized/distributed: different protocols
## Event Order

<table>
<thead>
<tr>
<th>Time order</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• which event is before and which is after</td>
<td></td>
</tr>
<tr>
<td>• No global Clock</td>
<td></td>
</tr>
</tbody>
</table>

### Event

- [A] Happened-before relation (→)
- [B] If A and B are events in the same process, and A was executed before B, then $A \rightarrow B$
- [C] 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 $A \rightarrow B$
- [D] If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
Relative Time for 3 processes

Happened-before:
- p1 → q2
- r0 → q3
- q3 → r4
- p1 → q4 (transitive order p1 → q2 and q2 → q4)

Concurrent events:
- q0 and p2, r0 and q3
- r0 and p3, q3 and p3
• Associate a timestamp with each system event.
  - Require that for every pair of events A and B,
  - if, \( A \rightarrow B, \)
  - then time-stamp of A is less than time-stamp of B

• Within each process \( P_i \) a logical clock \( LC_i \) is associated
  - simple counter for each event inside a process

• A process advances its logical clock when it receives
  a message whose timestamp is greater than
  the current value of its logical clock.

• If the timestamps of two events A and B are the same,
  - then the events are concurrent.
  - use the process identity numbers to break ties and
  - to create a total ordering.
Example of logical clocks

Figure shows an example of how logical clocks are updated under Lamport's scheme.

- Clocks at P1 and P2 are assumed to be at zero initially. Clocks are incremented by 1 each time by a fixed increment value (increment value, \( d = 1 \)).

Rules for updating the clocks are:

Rule 1 - If \( a \) and \( b \) are two successive events at process P, and event \( a \) "happens before" event \( b \) \( (a \rightarrow b) \), then clock value \( C(b) = C(a) + d \).

Rule 2 - If the receiving process receives a message with time of message \( tm = C(a) \), then new local clock value is found by -
\[
C(b) = Max(C(local\text{clock}), tm) + d
\]
Class Exercise: Lamport’s Clock

- Find out the logical clock values for events in the Figure.
- Find out all ”happens before” relationships in Figure.
- Is there any difference between clock and event order at events e32 and e33.

Figure - Events in a Distributed System
## Distributed Mutual Exclusion (DME)

### Problem
- Share common resources - example, PRINTER

### System
- Co-ordination among processes at different sites

### Outline
- **[A] 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.

- **[B] Requirement** -
  - If \( P_i \) is executing in its critical section, then no other process \( P_j \) is executing in its critical section.

- **[C] Two approaches** -
  - Centralized approach
  - Distributed 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 → coordinator).

The coordinator decides which process can enter the critical section next, and it sends (reply message → that process).

When the process receives a reply message from the coordinator, it enters its critical section.
After exiting its critical section, the process sends (release message → coordinator) and proceeds with its execution.

This scheme requires three messages per critical-section entry:
request,
reply,
release

- When process Pi wants to enter its critical section, it generates a new timestamp, TS, and sends the message request \((\text{Pi}, \text{TS}) \rightarrow \) all other processes.
- When process Pj receives a request message, it may reply immediately or it may defer sending a reply back.
- When process Pi 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.
The decision whether process Pj replies immediately to a request (Pi, TS) message or defers its reply is based on three factors:

- If Pj is in its critical section, then it defers its reply to Pi.
- If Pj does not want to enter its critical section, then it sends a reply immediately to Pi.
- If Pj 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 Pi (Pi asked first).
  - Otherwise, the reply is deferred.
Good Points - Fully Distributed Approach [15]

- Freedom from Deadlock is ensured.
- Freedom from starvation is ensured, since entry to the critical section is scheduled according to the timestamp ordering. The timestamp ordering ensures that processes are served in a first-come, first served order.
- The number of messages per critical-section entry is \( 2 \times (n - 1) \).

This is the minimum number of required messages per critical-section entry when processes act independently and concurrently.
Three Undesirable Problems

- 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.

- Processes that have not entered their critical section must pause frequently to assure other processes that they intend to enter the critical section. This protocol is therefore suited for small, stable sets of cooperating processes.
OHP 8

- [A] Please see example in OHP 7

[B] Please find the event order and clock order for messages in OHP 8