Outline

- Clock
  - Logical Clocks & Ordering of Events
  - Vector Clocks
- Q&A
Partial Ordering of Events

**Assumptions**

- System Consists of N Processes: 
  \[ p_i \quad (i = 1, \ldots, N) \]
- System Is Composed of a Collection of Events
  - Examples of events
    - Execution of a subprogram
    - Execution of a single machine instruction
    - Sending or receiving a message

\[
\text{history}(p_i) = h_i = \left\{ e_i^k \mid k = 1, \ldots \right\} \quad (i = 1, \ldots, N)
\]
Partial Ordering of Events

[{

“Happened Before” Relation ($\rightarrow$)

1. $k < l \Rightarrow (if) \ e_i^k \rightarrow e_i^l$ (i = 1, ..., N)
2. $e_i = \text{send}(m), e_j = \text{receive}(m)$ for Message $m$
   \[ \Rightarrow \ e_i \rightarrow e_j \]
3. $e ightarrow e' \land e' \rightarrow e'' \Rightarrow e \rightarrow e''$

   * $e \not\rightarrow e$

Concurrent Relation ($\parallel$)

\[ e \not\rightarrow e* \land e* \not\rightarrow e \iff e \parallel e* \]
Logical Clocks [Lamport78]

- Ways of Assigning Numbers to Events
  \[ LC_i(e), \text{where } e \text{ is an event in process } p_i \]

- Clock (Correctness) Condition
  \[ e \rightarrow e^* \implies LC(e) < LC(e^*) \]

- Satisfied if the following two conditions hold:
  1. \[ k < l \implies LC(e^k_i) < LC(e^l_i) \quad (i = 1, ..., N) \]
  2. \[ e_i = \text{send}(m), e_j = \text{receive}(m) \text{ for Message } m \]
     \[ \implies LC(e_i) < LC(e_j) \]
**Logical Clocks (Cont’d) [Lamport 78]**

**Implementation Rules**

1. $LC(e_i^{k+1}) = LC(e_i^k) + 1$ for Successive Events $e_i^k, e_i^{k+1}$
   
   $(i = 1, ..., N)$

2. $e_i = send(m), e_j^l = receive(m)$ for Message $m$

   $\Rightarrow \quad t = LC(e_i) \in m, LC(e_j^l) = \max\{LC(e_j^{l-1}), t\} + 1$
Ordering the Events Totally

[Model T78]

Total Ordering ($\Rightarrow_{t.o.}$)

\[ e \Rightarrow_{t.o.} e^* \iff LC_i(e) < LC_j(e^*) \vee \{LC_i(e) = LC_j(e^*) \land i < j\} \]

\[ e \rightarrow e^* \Rightarrow e \Rightarrow_{t.o.} e^* \]

- **Use Case: Mutual Exclusion Problem**
  - Processes sharing a single resource
  1. A process using it must release it before the following use
  2. The processes must use it in their request order
  3. If it is eventually released, every request is eventually granted
Ordering the Events Totally (Cont’d) [Lamport78]

- Nontrivial Scheduling Problem
  - The Requests Cannot Be Granted in the Order They Are Received

![Diagram showing scheduler and physical time with P0, P1, P2 and an arrow indicating an earlier request received later.](image-url)
Ordering the Events Totally (Cont’d)

Nontrivial Scheduling Problem

Assumptions for an Algorithm

- In-order message delivery
- Each process’ request queue
  - Initially containing a request with T0:P0

Algorithm

- Pi sends a Tm:Pi Req to all others and puts it in the queue
  - Each receiver puts it in the queue and sends a timestamped Ack to the requestor
- When releasing the resource, Pi removes the Req and sends a timestamped Rel to all others
  - Each receiver removes Pi’s Req from its queue
- The Tm:Pi Req is granted if it is the first in ⇒, and Pi has received a message from all others timestamped later than Tm

Smallest Time Value: Scheduler ID
Vector Clocks

**Shortcomings of Lamport’s Logical Clock**

\[ LC_i(e) < LC_j(e^*) \iff e \rightarrow e^* \]
Vector Clocks (Cont’d)

Implementation Rules

(1) $VC_i[j] = 0 \ (i, j = 1, ..., N)$

(2) $VC_i(e_i^{k+1})[i] = VC_i(e_i^k)[i] + 1 \ for\ Successive\ Events\ e_i^k, e_i^{k+1} \ (i = 1, ..., N)$

(3) $e_i = send(m), e_j^l = receive(m) \ for\ Message\ m$

\[ \Rightarrow \ t = VC_i(e_i) \in m, VC_j(e_j^l) = \max\{VC_j(e_j^{l-1}), t\} + j \ 1 \]

Increment the j-th Elt
Vector Clocks (Cont’d)

- Vector-Timestamp Comparisons

\[ VC = VC^* \text{ iff } VC[j] = VC^*[j] (j = 1, \ldots, N) \]
\[ VC \leq VC^* \text{ iff } VC[j] \leq VC^*[j] (j = 1, \ldots, N) \]
\[ VC < VC^* \text{ iff } VC \leq VC^* \land VC \neq VC^* \]

- Properties

\[ e \rightarrow e^* \text{ iff } VC(e) < VC(e^*) \]

- Hints for the Proof

\[ e \parallel e^* \Rightarrow \neg \{ VC(e) \leq VC(e^*) \lor VC(e) \geq VC(e^*) \} \]
\[ (\text{Note That } VC_i[j] \leq VC_j[j]) \]
Vector Clocks

Properties

\[ e_i \Rightarrow e_j \text{ iff } VC_i[i] \leq VC_j[i], \text{ where } i \neq j \]

Hint for the Proof

(Note That \( VC_i[j] \leq VC_j[j] \))

\[ \exists k (\neq j) \text{ s.t. } VC_i[k] < VC_j[k] \]

\[ \Rightarrow \exists e_k \text{ s.t. } (e_k \leftrightarrow e_i) \land (e_k \rightarrow e_j) \]
Vector Clocks (Cont’d)

Properties

\[ \exists k (\neq j) \text{ s.t. } VC_i(e_i)[k] < VC_j(e_j)[k] \]
\[ \Rightarrow \; \exists e_k (\neq e_i) \text{ s.t. } (e_k \leftrightarrow e_i) \land (e_k \rightarrow e_j) \]

Suppose That \( i = k \),
\[ \exists k (\neq j) \text{ s.t. } VC_k(e_k^*)[k] < VC_j(e_j)[k] \]
\[ \Rightarrow \; \exists e_k (\neq e_k^*) \text{ s.t. } (e_k \leftrightarrow e_k^*) \land (e_k \rightarrow e_j) \]
iff \[ \exists e_k (\neq e_k^*) \text{ s.t. } e_k^* \rightarrow e_k \rightarrow e_j \]
\[ \exists e_k (\neq e_k^* \land k \neq j) \text{ s.t. } e_k^* \rightarrow e_k \rightarrow e_j \]
\[ \Rightarrow \; VC_k(e_k^*)[k] < VC_j(e_j)[k] \]
Vector Clocks (Cont’d)

Properties

∀(k ≠ j), \( VC_k(e_k^*)[k] ≥ VC_j(e_j)[k] \)

⇒ ¬{∃e_k(≠ e_k^*) } s.t. e_k^* → e_k → e_j

If the Monitoring Process Keeps the Value of the k-th Elt of VC from P_k, It Can Decide Whether There Is No Event That Happened before a Given Event

- Wait