Event Ordering

- Keeping the clock synchronized within 5,10 millisec is expensive task.
- Lamport observed, Its not necessary to keep the clocks synchronized in distributed system.
- It is sufficient to keep the all event be totally ordered.
- For Partial ordering of the events, Lamport defined a new relation called Happened-Before relation.

Happened Before Relation

Happened Before Relation(Denoted by →) on a set of events Satisfies Following Conditions.

- If a and b are two events in the same process and a occurs before b, then a→b.
- If a is event of sending a message by one process and b is event of receipt of same message by another process, then a→b. it follows law of causality because receiver can not receive message until sender sends it and propagation time from sender to receiver is always positive.
- If a→b and b→c, then a→c. happened before relation follows transitive relation
- a→a is not possible. Happened before relation is irreflexive partial ordering.

Happened Before relation cont..

Event types

- Causal events
- Concurrent Events:

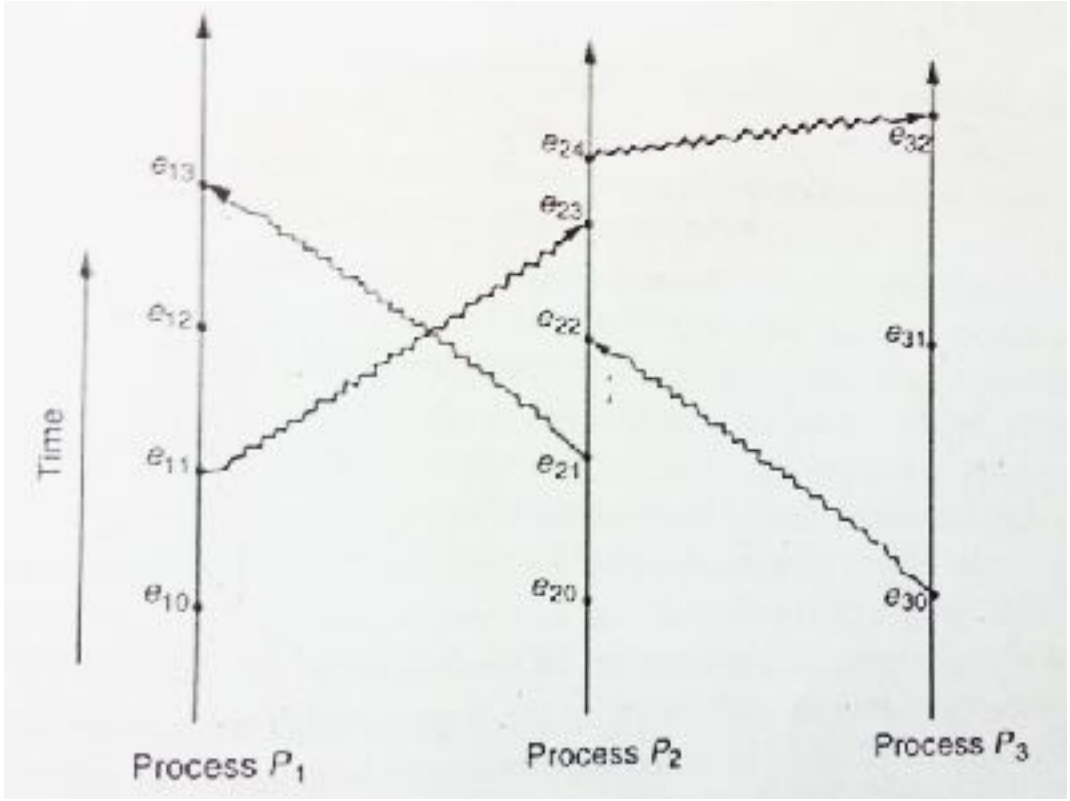
- Causal Events:

Two events a →b is true if path exists between a and b by moving forward in time along process and message lines in the direction of arrows. These events are related by happened before relation.

- Concurrent events:

Two events are concurrent if no path exists between them and are not related by happened before relation.

Diagram



- Causal Events:
- e1o → e11 e20 →e24 e11 → e23 e21 →e13
- e30 → e24
- e11 → e32
- Concurrent Events:
- e12 and e20 e21 and e30 e10 and e30 e11 and e31 e12 and e32 e13 and e22

Logical Clocks Concept

For determining the event a occurs before b, need common clock, or set of perfectly synchronized clocks.

- Neither of these available, Lamport provided a solution of this problem.
- The logical clock concept is to associate a timestamp with each system event.
- Each process Pi, has a clock Ci.
- And assigned a number Ci(a) to any event in that process.
- In fact, Logical clocks implemented by counters with no actual timing mechanism.
- Happened-before relation should be ordered, using these clocks.
- So that Logical clocks must satisfy the following condition:
- For any two events a and b, if $a \rightarrow b$, then $C(a) < C(b)$

Implementation of Logical Clocks

Happened Before algorithm with logical clocks must follows the following conditions:

- C1: If a and b are two events within the same process Pi and a occurs before b , then Ci(a) < Cj(b).
- C2: If a is the sending message by process Pi, and b is the receipt of that message by process Pj, then Ci(a) < Cj(b)
- C3: A clock Ci associated with a process Pi, must always go forward , never backward, That is, corrections to time of a logical clock must always be made by adding a positive value to the clock , never by subtracting the value.

Implementation of Logical Clocks

To meet conditions C1, C2, and C3 , Lamport used the following implementation rules:

IR1(implementation Rule): Each process Pi increments Ci between two successive events.

IR2(implementation Rule): If Pi sending a message m by an event a, and the message m contains a timestamp Tm= Ci(a) , and other process Pj receive this message Ci(a) . And if Ci(a) is greater than and equal to its present value then Ci(a) value updated on it with +1 increment, otherwise Cj present value continue.

Implementation of Logical Clocks by Using Counters

- Two processes P1 and P2 each have counters C1 and C2.
- C1 values increase C1=0,1,2,3,4,5…
- C2 values increase C2=0,1,2,3,4,5…
- Counters act like logical clocks.
- Message having the incremented value of counter, that is sent by P1.
- On the other side P2 receive this message, and instead of simple increment of its counter, a check is made .
- Check is , If the incremented counter value of P2 is less than or equal to the timestamp in the received message from P1 .
- if C2 <= C1 ,e.g 3 <= 4 , C2 value will be updated with +1.

Implementation of Logical Clocks by Using Counters cont…

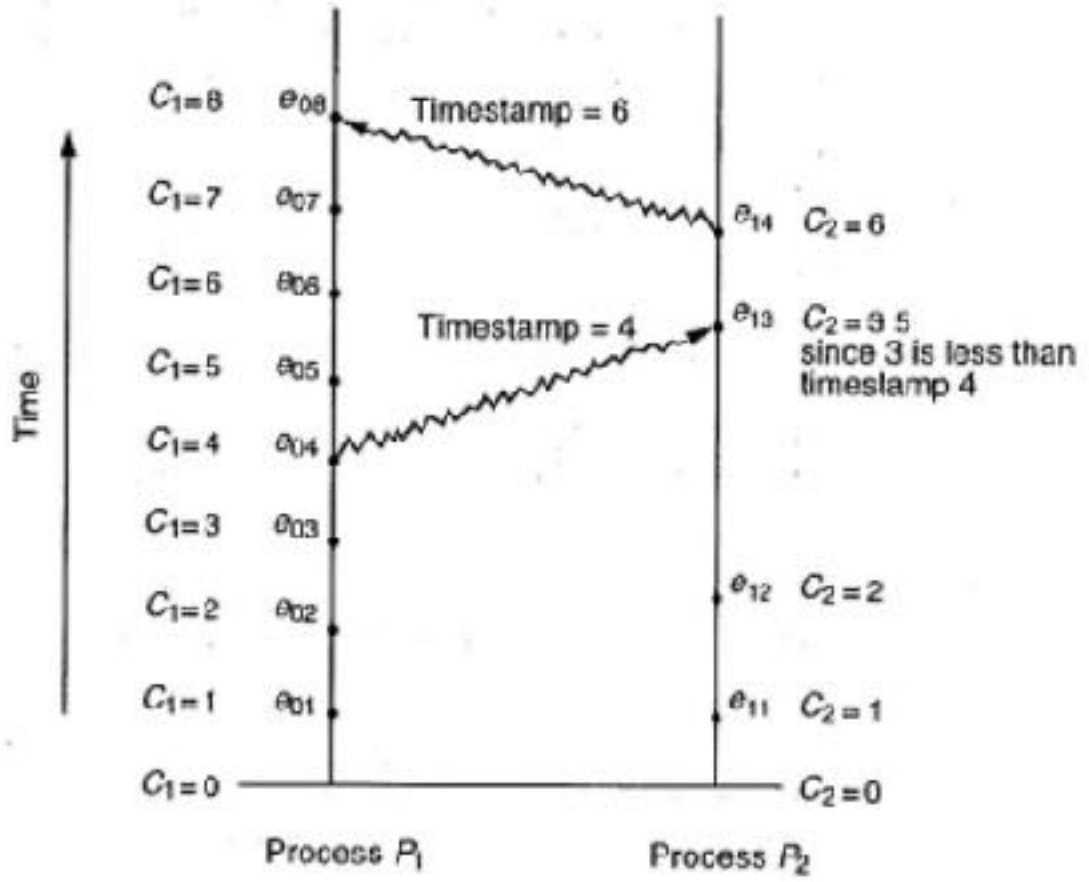
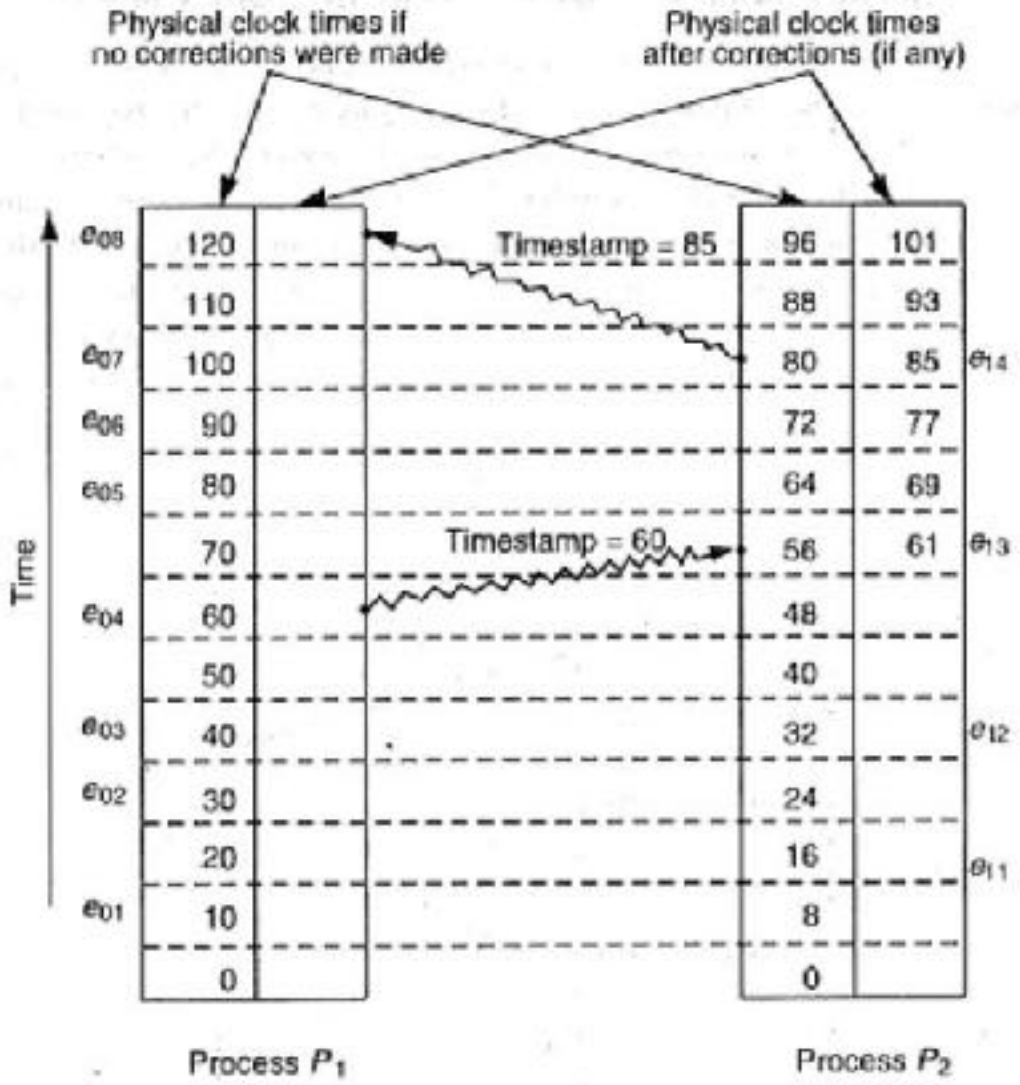


Fig. 6.4 Example illustrating the implementation of logical clocks by using counters.

Implementation of Logical Clocks by Using Physical Clocks

- Each process has a physical clock associated with it.
- Each clock run at a constant rate.
- However, The rates at which different clocks run are different.
- When clock of process P1 has ticked 10 times, The clock of process P2 has ticked only 8 times.
- , If the incremented counter value of P2 is less than or equal to the timestamp in the received message.
- if C2 <= C1 ,e.g 56<= 60 , C2 value will be updated with +1.

Implementation of Logical Clocks by Using Physical Clocks cont..



Total Ordering of Events

Happened-before has only partial ordering.

- Two events a or b not related by Happened-before , if same timestamps are associated with them.
- P1 and P2 clocks are same for example 100 .
- Nothing can be said about the order.
- Additional information required.
- Lamport purposed Total ordering of processes.
- He used process identity numbers.
- Timestamp associated with them will be 100.001 and 100.002.
- Here , process identity numbers of processes P1 and P2 are 001 and 002.

Mutual Exclusion

- A file must not be simultaneously updated by multiple processes.
- Such as printer and tape drives must be restricted to a single process at a time.
- Therefore , Exclusive access to such a shared resource by a process must b ensured.
- This exclusiveness of access is called mutual exclusion between processes.

Critical Section

- The sections of a program that need exclusive access to shared resources are referred to as critical sections.
- For mutual exclusion , Ways are introduced, to prevent processes from executing concurrently with their associated critical section.

- Following requirement must satisfy for mutual exclusion:
- 1. Mutual exclusion: At anytime only one process should access the resource. A process can get another resource first releasing its own.
- 2. No starvation: if every process that is granted the resource eventually release it, every request must be eventually granted.

Starvation of process

- Indefinite postponement of a process because it requires some resource. but the resource is never allocated to this process. It is sometimes called livelock.

  [https://www.cs.auckland.ac.nz/~alan/courses/os/book/6.Mana.13.starvation.pdf]

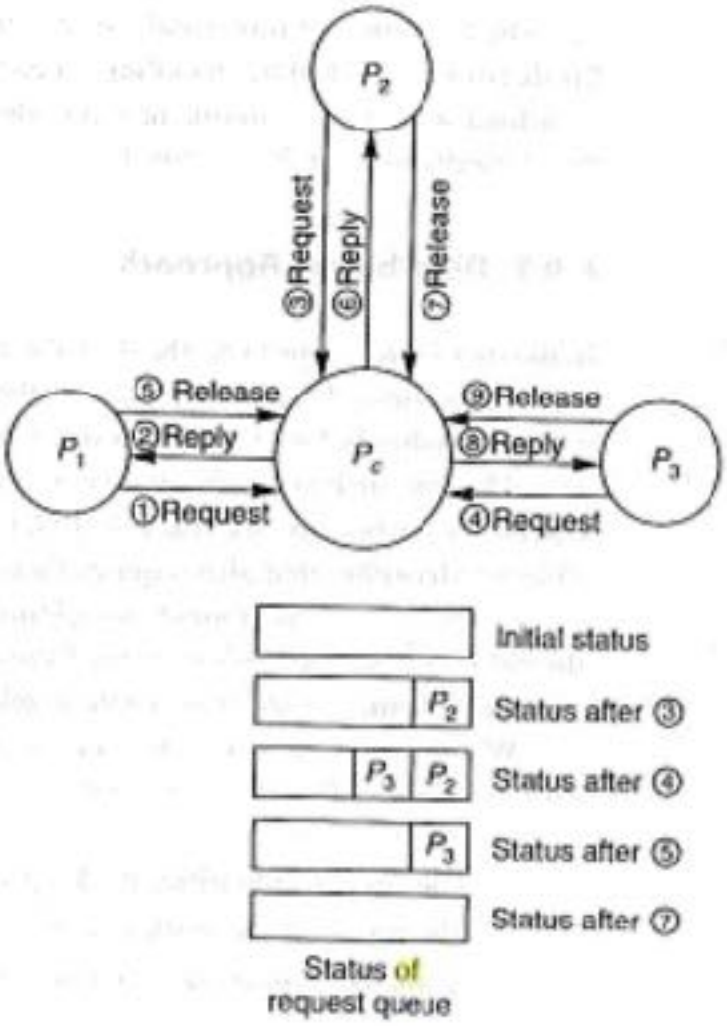
Three approaches to implement mutual exclusion

- Centralized Approach
- Distributed Approach
- Token Ring

Centralized Approach

- One process in the system is selected as coordinator.
- That coordinates the entry to the critical sections.
- Each process that wants to enter a critical section must first seek permission from the coordinator.
- If two processes simultaneously ask for permission to enter the critical section , Coordinator grants permission according with some scheduling algorithms.
- When process exits, it notify the coordinator so that the coordinator can grant permission to another process.
- This algorithm (in centralized) ensures mutual exclusion, because at a time , the coordinator allows only one process to enter in critical section.
- This algorithm (in centralized) ensures no starvation, because use of FCFS (First come First served) .

Centralized Approach cont..



Centralized Approach cont..

- Benefits:
  - easy to implement
  - Requires only three messages per critical section (request, reply, release).
- Drawbacks:
  - Single coordinator
  - Single point of failure
  - Performance down due to bottleneck in large systems.

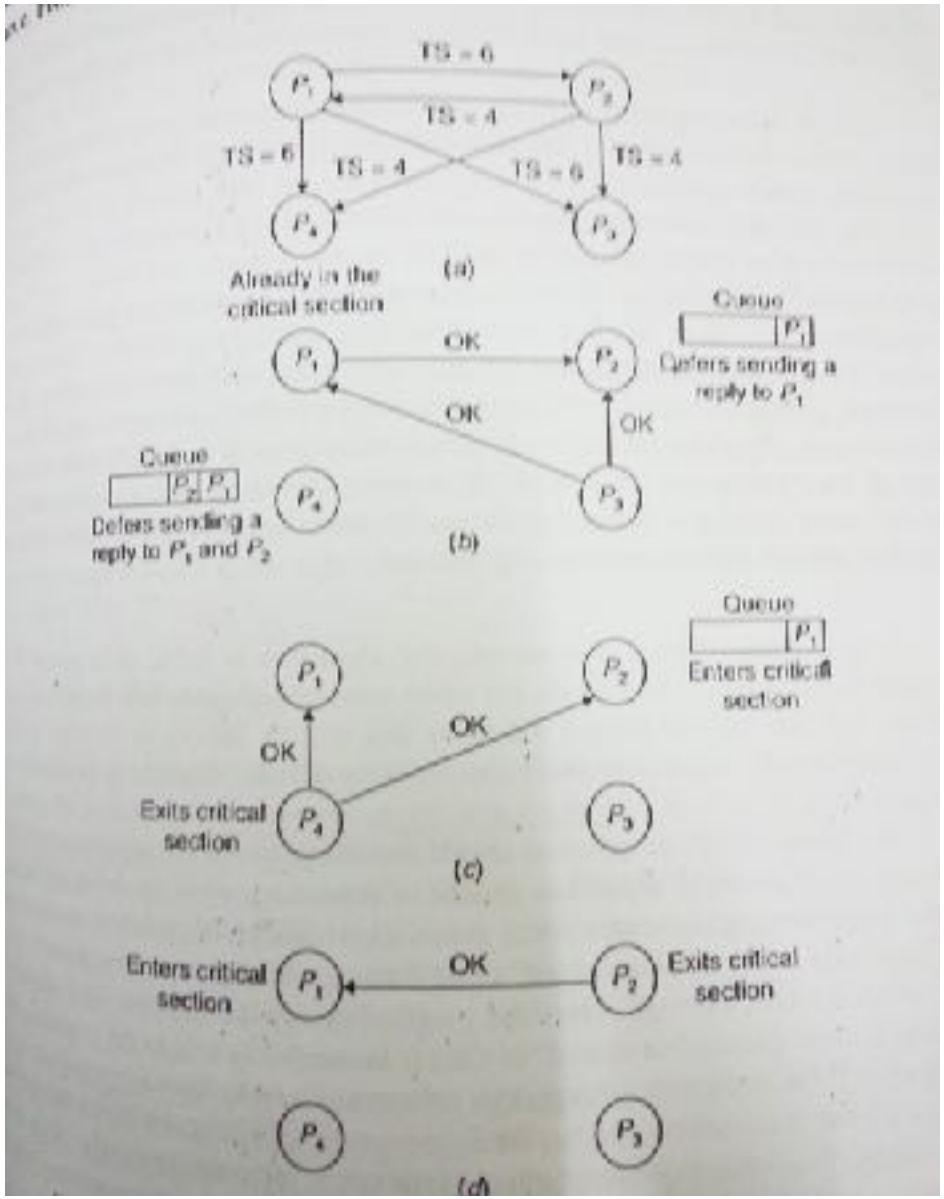
Distributed Approach

- Decision making for mutual exclusion is distributed across entire system.
- All process that want to enter critical section co-operate with each other before taking decision.
- Ricart and Agarwala's algorithm is used, based upon Lamport's event ordering scheme to assign unique time stamp to each event.
- When a process wants to enter critical section it sends request message to all process. Message contains
- Process identifier of process
- Name of critical section that process wants to enter.
- Unique time stamp generated by process for request message.
- On receiving request message a process immediately sends reply message or defers reply based upon following rules.

Distributed Approach Cont..

- If receiver process is in critical section, it places request message in queue and defers sending a reply.
- If receiver process is not in critical section and waiting to enter, it compares it's own time stamp with received time stamp. If received time stamp is lower than it' own, it means that process made request earlier, than it sends reply to that process otherwise places it in queue and defers sending reply.
- If receiver process is neither in critical section nor waiting, it simply sends reply message.
- A process enters critical section if it receives message from all process, executes in critical section and than sends reply message to all processes in queue.

Distributed Approach Cont..



Drawbacks and solutions

- 1)System having n processes is liable to n points of failure and blocks in case of single process failure.
- Solution: receiver sends permission denied message when it is in critical section, else it sends OK. if receiver did not receive reply it keeps on trying after time out,until it gets reply, else it considers that process has crashed.
- 2) Each process must know identity of process participating in group.
- When a process joins it must receive names of all other processes and name of new process must be distributes to all members of group.
- suitable only for small group due to large no of updates of creation and deletion.
- 3) To much wait if there are large no of nodes. If there are n nodes and single message travels through network then each process will wait for 2(n-1) messages. Suitable for small group.
- **Best solution: Majority** consensus rather than consensus of all processes is proposed in literature.

Token passing Approach

- Mutual Exclusion is achieved by circulating single token among processes.
- Token is special type of message that entitles its holder to enter critical section. Process are arranged in logical ring structure.
- Token can be moved clockwise or anti-clockwise.
- When a process gets token it can enters critical section and after finishing work, exits from critical section and passes token to neighbor process.
- Process can enter critical section one time in each turn with token.
- If process is not interested to enter in critical section, it simply passes token to neighbor, if no one is interested token keeps on circulating in ring.
- Waiting time varies from 0 to n-1 messages depending upon receipt of token and interested to enter in critical section or interested just after the token passed to neighbor.

Drawbacks and Solutions:

- 1) **Process failure**: Single process failure causes logical ring to break.
- Solution: exchange of Ack(acknowledgement) after message Receipt with neighbor. If no Ack comes, means problem with process.
- Current ring information with each process, using this each process skips its faulty neighbor process and when that process recovers, it informs it's previous neighbor or process.
- 2**) Lost Token**: Token is lost, new token message must be generated.
- Solution: Designate one process as monitor process.
- Monitor periodically circulates "who has token" message. All the process passes message to neighbor, except the process which has token. It send its process identifier in special field with token to neighbor.
- When message returns to monitor after one circulation, if there is no entry in that field means token has been lost, generates new token and circulates in ring
- 3) **Monitor process and "who has token" message is lost.**
- Solution: more than one monitor process. And if one fails election has to be done for selection of next monitor who will generate token.

Diagram