Times, Clocks, and the Ordering of Events in a Distributed System

Jacopo Notarstefano
jacopo.notarstefano [at] gmail.com

13 February 2013

Main objectives and outline

In this paper Lamport:

1. Discusses the partial ordering defined by the "happened before" relation,
2. Gives a distributed algorithm for extending it to a consistent total ordering of all the events,
3. Uses this algorithm to solve a mutual exclusion problem.

Distributed system

Definition (Distributed system)

A *distributed system* consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages.
A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

Examples of distributed systems

- A worldwide network of interconnected computers
- A cluster of workstation in a data center
- Processes on a single computer

Ordering relations

Definition (Partial order)

A *partial order* $\prec$ on a set X is a reflexive, antisymmetric, transitive relation.

Definition (Total order)

A *total order* $\geq$ on a set X is a partial order such that, if $a, b \in X$, then either $a \geq b$ or $b \geq a$.

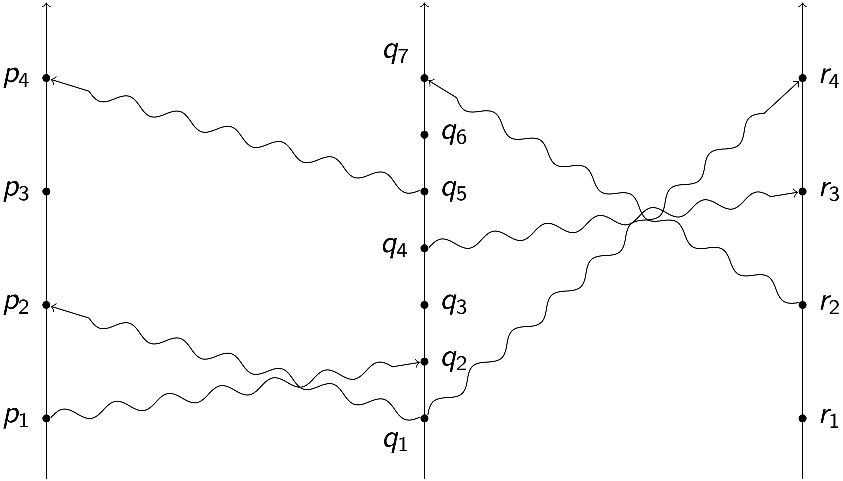
The "$\rightarrow$" relation

Definition (The "$\rightarrow$" relation)

The "$\rightarrow$" relation on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process, and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$, and $b \rightarrow c$, then $a \rightarrow c$.

The "space-time diagram"



Clocks

Definition (Clock)

For each process P_i we define a *clock* C_i to be a function that assigns a number $C_i\langle a\rangle$ to each event a in the process.

Definition (System of clocks)

A *system of clocks* is a function C that assigns to the event b in process P_j the time $C\langle b\rangle = C_j\langle b\rangle$.

The clock condition

Definition (The clock condition)

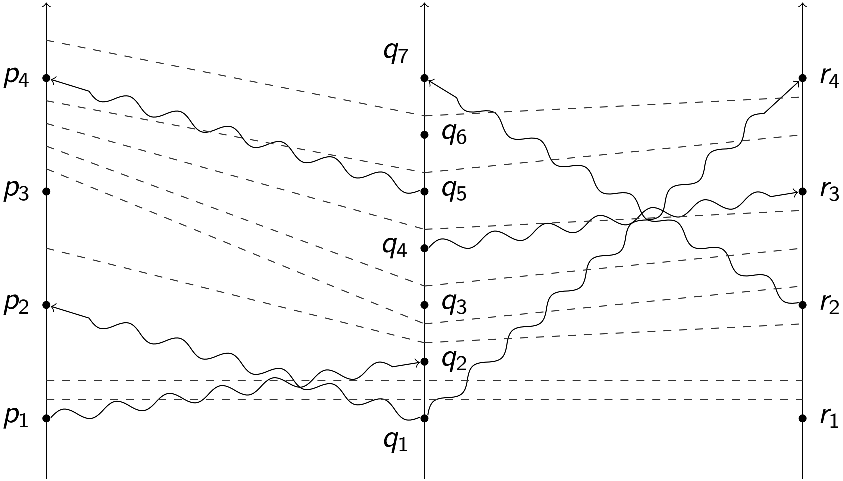
We say that a system of clocks satisfies the *clock condition* if, for any events a and b, we have: if $a \to b$ then $C\langle a\rangle < C\langle b\rangle$.

Lemma

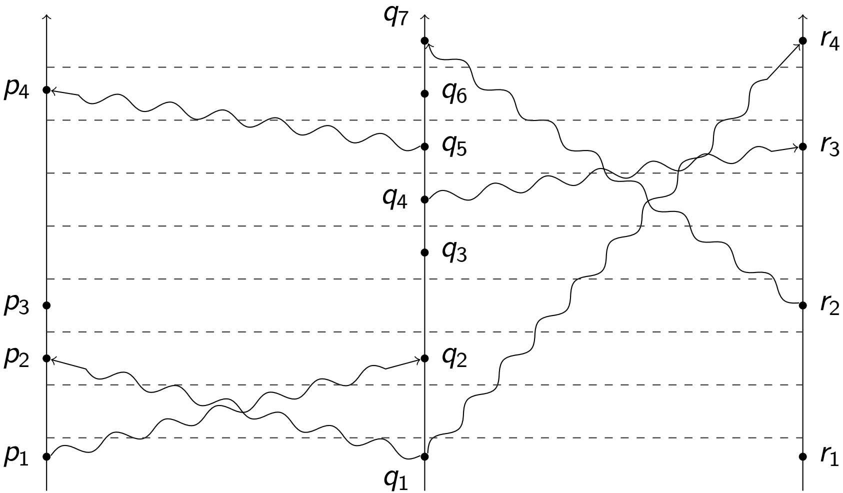
The clock condition is satisfied if the following conditions hold:

1. *If a and b are events in process P_i and a comes before b, then $C_i\langle a\rangle < C_i\langle b\rangle$.*
2. *If a is the sending of a message by process P_i and b is the receipt of that message by process P_j, then $C_i\langle a\rangle < C_i\langle b\rangle$.*

The “space-time diagram”, revisited



The "space-time diagram", rearranged



Implementation of the clock condition

Lemma

To guarantee that the system of clocks satisfies the clock condition we need to obey the following implementation rules:

1. *Each process P_i increments C_i between any two successive events.*
2. *If event a is the sending of a message m by process P_i, then the message m contains a timestamp $T_m = C_i\langle a \rangle$.*
3. *Upon receiving a message m, process P_j sets C_j greater than or equal to its present value and greater than T_m.*

The "$\Rightarrow$" relation

Definition (The "$\Rightarrow$" relation)

Let $\prec$ be a total ordering on the processes. If a is an event in process P_i and b is an event in process P_j, then $a \Rightarrow b$ if and only if either

1. $C_i\langle a\rangle < C_j\langle b\rangle$ or
2. $C_i\langle a\rangle = C_j\langle b\rangle$ and $P_i \prec P_j$.

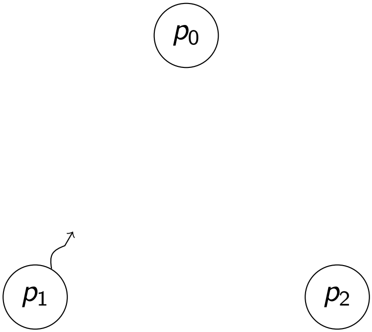
A mutual exclusion problem

A fixed collection of processes share a single resource, which can be used by one process at a time. We want to find an algorithm that satisfies the following conditions:

1. A process which has been granted the resource must release it before it can be granted to another process.
2. Different requests for the resource must be granted in the order in which they are made.
3. If every process which is granted the resource eventually releases it, then every request is eventually granted.

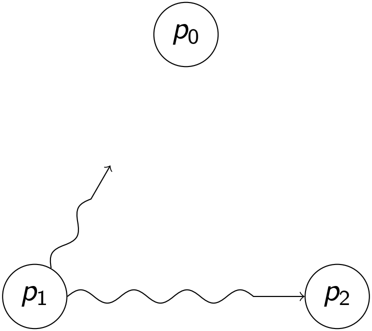
A wrong solution, 1/3

We can't use a central scheduling process which grants requests in the order they are received. Suppose that p_1 sends a message to p_0, the scheduler, to request the resource.



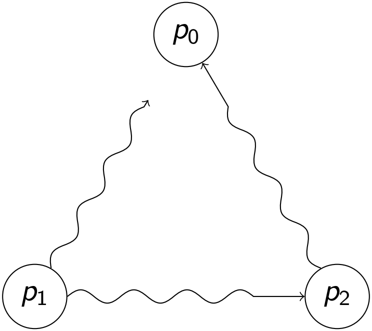
A wrong solution, 2/3

Then p_1 sends a message to p_2. Suppose that this message is received while the other is still traveling due to network congestion.



A wrong solution, 3/3

Finally, p_2 sends a message to p_0 requesting the resource. p_0 then grants the resource to p_2, which requested the resource after p_1.



Some assumptions

To simplify the problem, we make some assumptions.

1. For any two processes P_i and P_j, the messages sent from P_i to P_j are received in the same order as they are sent.
2. Every message is eventually received.
3. A process can send messages directly to every other process.

The request queue

Let P_0 be the process to which the shared resource is initially allocated. Let T_0 be less than the initial value of any logical clock in the system. Each process mantains a private *request queue*, which initially contains one message: "T_0: P_0 requests resource".

Resource request and acknowledgment

1. Process P_i sends the message "T_m: P_i requests resource" to every other process where T_m is the process clock's value at the time of the request. P_i also puts the request message on its request queue.
2. When process P_j receives P_i's request message, it puts it in its queue and sends an acknowledgment message to P_i.

Resource release and acknowledgement

3. Process P_i removes request message "T_m: P_i requests resource" from its queue and sends the release message "P_i releases resource" to every other process.
4. Process P_j removes the "T_m: P_i requests resource" from its queue.

Resource allocation

5. Process P_i is allocated the resource when:
   - There is a "T_m: P_i requests" resource in P_i's request queue which is before any other request according to $\Rightarrow$.
   - P_i has received messages from every other process timestamped later than T_m.

Proof of correctness (sketch)

Theorem

The algorithm described by rules 1 to 5 is a correct solution to the mutual exclusion problem.

Proof (sketch).

The ordering is guaranteed by the fact that relation "$\Rightarrow$" extends relation "$\rightarrow$". $\square$