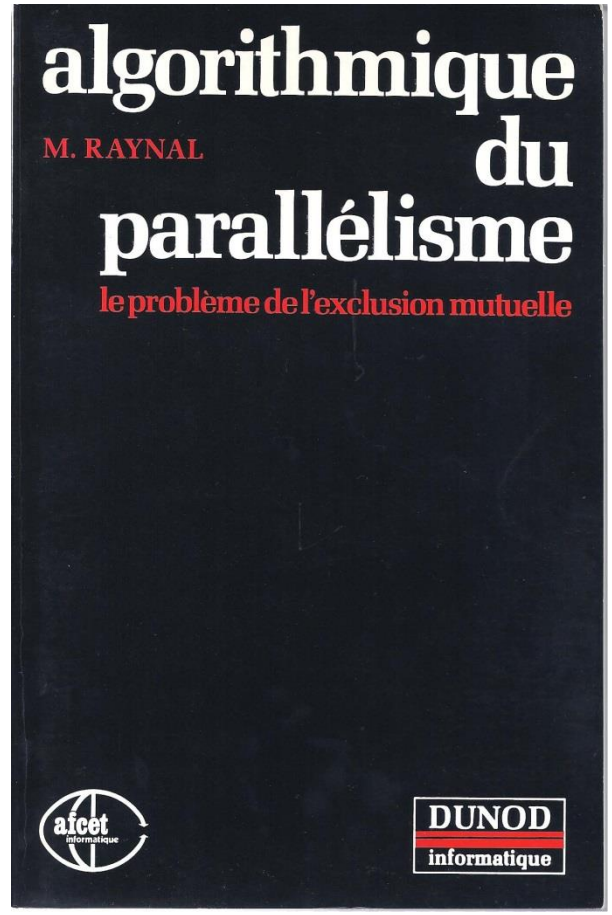
Distributed Mutual Exclusion



The mutex problem

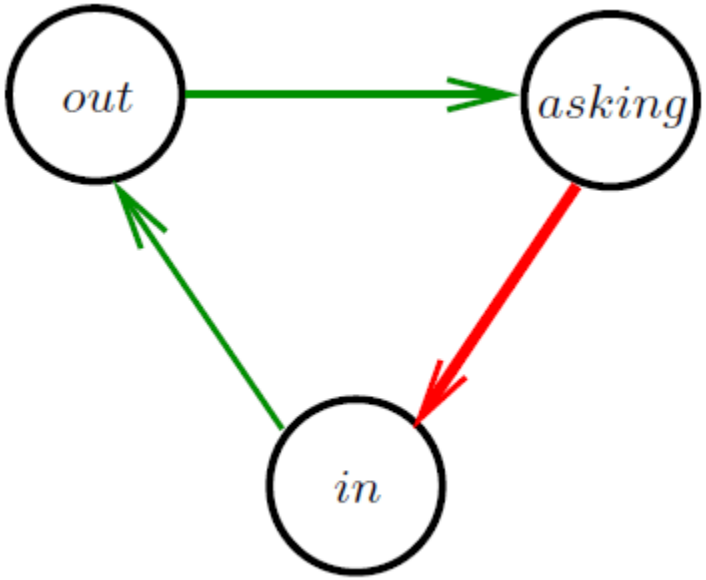
Concurrent access of nodes (processes) to a shared resource or data is executed in mutually exclusive manner.

Only one node is allowed to execute the critical section (CS) at any given time.

The mutex problem

Safety: no two nodes are concurrently in the CS

Liveness: any request is eventually granted



Distributed algorithms

In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.

Message passing is the sole means for implementing distributed mutual exclusion.

Distributed algorithms

Two or more successive rounds of messages are exchanged among the nodes to determine which node will enter the CS next.

A node enters the CS when an assertion, defined on its local variables, becomes true.

Distributed algorithms

Lamport, L. (1978). *Time, clocks and the ordering of events in a distributed system*.

Ricart, G. and Agrawala, A.K. (1981). *An optimal algorithm for mutual exclusion in computer networks*.

Carvalho, O. and Roucairol, G (1983). On mutual exclusion in computer networks. Technical correspondence. (With authors' response).

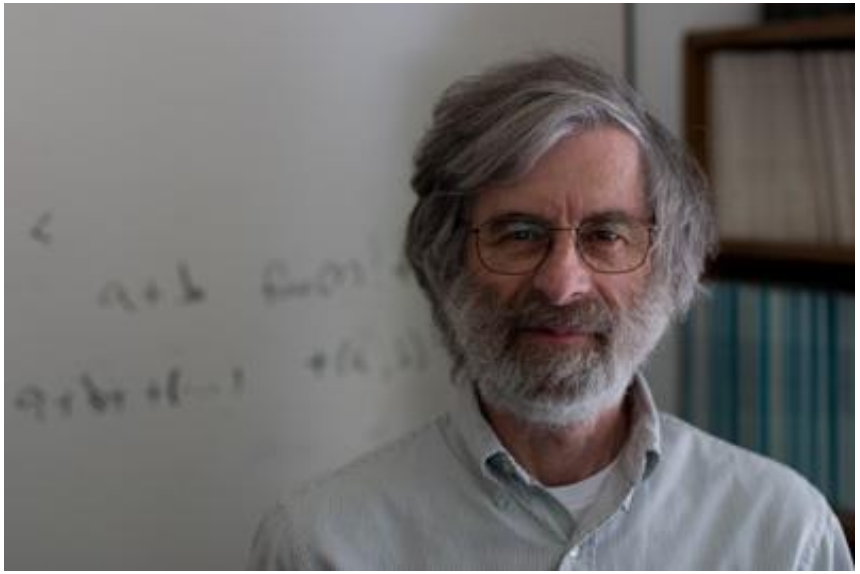
Maekawa, M. (1985). A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems.

Time, Clocks, and the **Ordering of Events** in a Distributed System

Lamport, L. (1978).
Communications of the ACM, Vol. 21, No. 7, pp. 558-565.

Leslie Lamport

2013 Turing award - For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and **logical clocks**, safety and liveness, replicated state machines, and sequential consistency.



Logical clocks

Sistemas Operativos Avanzados

Sincronización y Concurrencia

Cómputo Distribuido

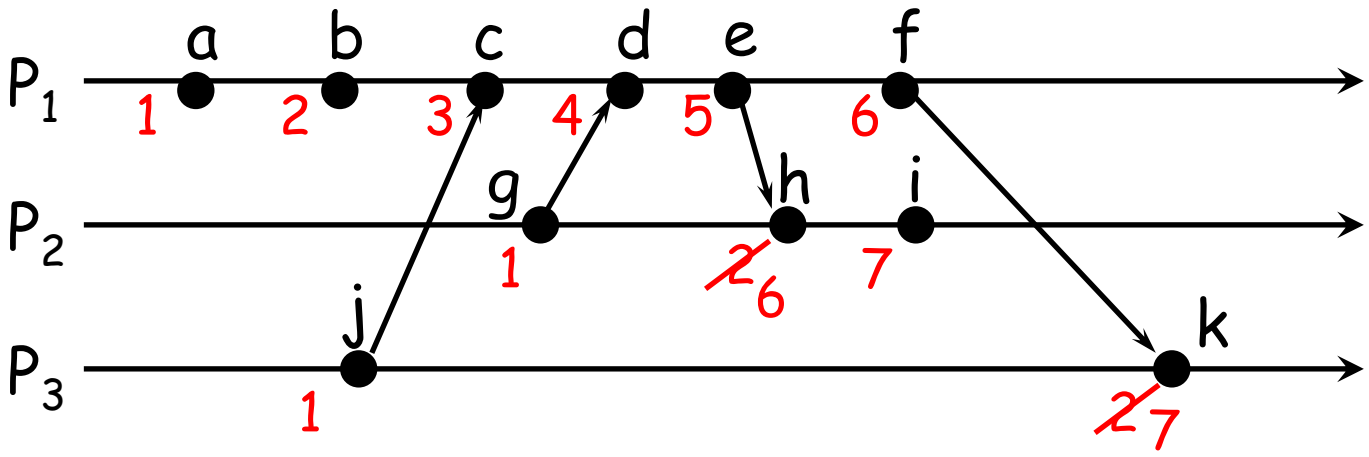
Algoritmos (de control) distribuidos

Coordinación

Exclusión mutua y FSM replicadas

Intercambio de mensajes

Logical clocks



$a \rightarrow b$

happened before relation

partial ordering

Logical clocks

A clock C is just a way of assigning a number to an event, where the number is thought of as the time at which the event occurred.

Clock (consistency) condition:

if $\boldsymbol{a \rightarrow b}$ then $\mathbf{C(a) < C(b)}$

Transitive relation:

if $\boldsymbol{a \rightarrow b}$ and $\boldsymbol{b \rightarrow c}$ then $\boldsymbol{a \rightarrow c}$

Logical clocks

The clock condition is satisfied if the following two conditions hold.

C1. If a and b are events in node P_i, and a comes before b, then $C_i(a) < C_i(b)$.

C2. If a is the sending of a message by node P_i and b is the receipt of that message by node P_j, then $C_i(a) < C_j(b)$.

Logical clocks

Implementation Rules to satisfy the clock condition:

IR1. Each node P_i increments C_i between any two successive events.

IR2.

(a) If event a is the sending of a message m by node P_i, then the message m contains a timestamp $Tm = C_i(a)$.

(b) Upon receiving a message m, node P_j sets C_j greater than or equal to its present value and greater than Tm.

Logical clocks

We define a relation $\Rightarrow$ as follows: if a is an event in node P_i and b is an event in node P_j, then $a \Rightarrow b$ if and only if either

(i) $C_i(a) < C_j(b)$ or

(ii) $C_i(a) = C_j(b)$ and $P_i < P_j$.

The relation $\Rightarrow$ is a way of completing the *happened before* partial ordering to a total ordering.

Symmetrical algorithm – Lamport (1978)

Lamport developed a distributed mutual exclusion algorithm as an illustration oh his clock synchronization scheme.

Symmetrical algorithm – Lamport (1978)

I. A node which has been granted the resource must release it before it can be granted to another node.

II. If every node which is granted the resource eventually releases it, then every request is eventually granted.

III. Different requests for the resource must be granted in the order in which they are made.

Requirements

Symmetrical algorithm – Lamport (1978)

Assumptions:

- Completely connected network
- Reliable nodes
- Reliable FIFO channels
  - Transmission delay is variable but finite

- Not anonymous: use nodes' id
- Not uniform: use N

Symmetrical algorithm – Lamport (1978)

Each node maintains a priority queue with all pending REQUEST messages in the system (ordered by the relation ⇒).

Symmetrical algorithm – Lamport (1978)

1. A requesting node i sends a REQUEST(TS_i:i) message to every other node and puts it on its queue.

2. Each node j places every REQUEST message it receives on its queue and sends a REPLY(TS_j:j) message to the originator (acknowledge).

Symmetrical algorithm – Lamport (1978)

A node enters its critical section when:

L1: Its REQUEST is at the front of its own queue.

L2: It has received a message with TS higher than the TS of its own REQUEST from all other nodes (the command is stable).

Symmetrical algorithm – Lamport (1978)

3. Upon completion of its critical section, node i removes its REQUEST from its queue and sends a RELEASE(TS_i:i) message to every other node.

4. Each node j receiving the RELEASE message removes the corresponding REQUEST from its queue.

Symmetrical algorithm – Lamport (1978)

Theorem: The algorithm is safe.

Proof is by contradiction.

Suppose two nodes P_i and P_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the nodes concurrently.

Symmetrical algorithm – Lamport (1978)

This implies that at some instant in time, say t, both P_i and P_j have their own requests at the top of their request queues and condition L2 holds at them.

Without loss of generality, assume that P_i 's request has smaller timestamp than the request of P_j.

Symmetrical algorithm – Lamport (1978)

From condition L2 and the FIFO property of the communication channels, it is clear that at instant t the request of P_i must be present in request queue j when P_j was executing its CS.

This implies that P_j 's own request is at the top of its own request queue when a smaller timestamp request, P_i 's request, is present in the request queue j.

This is a contradiction!

Symmetrical algorithm – Lamport (1978)

Theorem: The algorithm is fair.

Proof is by contradiction.

Suppose a node P_i's request has a smaller timestamp than the request of another site P_j and P_j is able to execute the CS before P_i.

Symmetrical algorithm – Lamport (1978)

For P_j to execute the CS, it has to satisfy the conditions L1 and L2.

This implies that at some instant in time say t, P_j has its own request at the top of its queue, and it has also received a message with timestamp larger than the timestamp of its request from all other nodes.

Symmetrical algorithm – Lamport (1978)

But request queue at a node is ordered by timestamp, and according to our assumption P_i has lower timestamp.

So P_i's request must be placed ahead of the P_j's request in the request queue j.

This is a contradiction!

Symmetrical algorithm – Lamport (1978)

For each CS execution, the algorithm requires (N−1) REQUEST messages, (N−1) REPLY messages, and (N−1) RELEASE messages.

For critical section invocation, the algorithm requires

3*(N-1) messages

An Optimal Algorithm for Mutual Exclusion in Computer Networks

Ricart, G. and Agrawala, A. (1981).
Communications of the ACM, Vol. 24, No. 1, pp. 9-17.

Recherche d'optimalité

Optimal algorithm - Ricart & Agrawala (1981)

Assumptions:

- Completely connected network
- Reliable nodes
- Reliable non FIFO channels
  - Transmission delay is variable but finite

Optimal algorithm - Ricart & Agrawala (1981)

1. A node making an attempt to invoke mutual exclusion sends a timestamped REQUEST message to all other nodes.

2. Upon receipt of the REQUEST message, each of the other nodes either sends a REPLY immediately or defers a response until after it leaves its own critical section.

3. A node enters its critical section after all other nodes have been notified of the request and have sent a REPLY granting their permission.

Optimal algorithm - Ricart & Agrawala (1981)

Each node i maintains a Reply_Deferred boolean array, RDi, the size of which is the same as the number of nodes in the system.

Initially, $\forall i \, \forall j : RD_i[j]=false$. Whenever node i defer the request sent by node j, it sets $RD_i[j]=true$ and after it has sent a REPLY message to node j, it sets $RD_i[j]=false$.

Shared database

```
CONSTANT
        ME, N;
INTEGER     // Sequence Numbers and counter
        Our_SN                          initial (0),
        Hgst_SN                         initial (0)
        Oustanding_Reply_Count  initial (0); // “authorization”
BOOLEAN
        Waiting                         initial (false),
        Reply_Deferred[1:N]             initial (false),
```

Procedures

```
procedure REQUEST-RESOURCE
begin INTEGER j;
    Waiting = true;
    Our_SN := Hgst_SN + 1
    for j = 1 step 1 until N do
          Outstanding_Reply_Count := N-1;
          if j ≠ ME then
                send(REQUEST(Our_SN, ME), j)
          fi
    od;
    waittfor (Oustanding_Reply_Count=0)
end REQUEST-RESOURCE.
```

Procedures

```
procedure TREAT-REPLY-MESSAGE(j)
begin
    Oustanding_Reply_Count := Oustanding_Reply_Count - 1
end TREAT-REPLY-MESSAGE.
```

Procedures

```
procedure TREAT-REQUEST-MESSAGE(Their_SN, j)
begin BOOLEAN Defer_it, Our_Priority;
    Hgst_SN := max(Hgst_SN,Their_SN);
    Defer_it = Waiting and // Our_PRIORITY
            (Their_SN > Our_SN or ((Their_SN = Our_SN) and j > ME));
    if Defer_it then
            Reply_Deferred[j] := true
    else
            send (REPLY(ME), j)
    fi;
end TREAT-REQUEST-MESSAGE.
```

Procedures

```
procedure RELEASE-RESOURCE
begin INTEGER j;
    Waiting = false;
    for i = 1 step 1 until N do
            if Reply_Deferred[j] then
                    Reply_Deferred[j] := false;
                    Send(REPLY(ME), j)
            fi
    od
end RELEASE-RESOURCE.
```

Optimal algorithm - Ricart & Agrawala (1981)

For critical section invocation, the algorithm requires

2*(N-1) messages

“this number is at a minimum if paralell (requests are processed by each node concurrently), distributed, symmetric control is used”

On Mutual Exclusion in Computer Networks

Carvalho, O. and Roucairol, G. (1983).
Technical correspondence.
Communications of the ACM, Vol. 26, No. 2, pp. 146-147.

Minimiser le nombre de messages

Carvalho & Roucairol (1983)

Once a node i has received a REPLY message sent by a node j, the implicit authorization in this message remains valid until i sends a REPLY message to j (which happens only after node i receives a REQUEST message from node j).

During this interval, node i will be able to enter its critical section any number of times without consulting node j.

Carvalho & Roucairol (1983)

For critical section invocation, the algorithm requires:

between 0 and 2*(N-1) messages

- Dynamic (adaptive): the size of the (consulted) request set varies dynamically for each process

Shared database

```
CONSTANT
        ME, N;
INTEGER     // Sequence Numbers
        Our_SN                  initial (0),
        Hgst_SN                 initial (0);
BOOLEAN
        Using                   initial (false),
        Waiting                 initial (false),
        Reply_Deferred[1:N]     initial (false),
        A[1:N]                  initial (false);  // “authorization”
```

Procedures

```
procedure REQUEST-RESOURCE
begin INTEGER j;
    Waiting = true;
    Our_SN := Hgst_SN + 1
    for j = 1 step 1 until N do
        if j ≠ ME and (not A[j]) then
            send(REQUEST(Our_SN, ME), j)
        fi
    od;
    waittfor (A[j] = true for all j ≠ ME);
    Waiting := false; Using := true
end REQUEST-RESOURCE.
```

Procedures

```
procedure TREAT-REPLY-MESSAGE(j)
begin
    A[j] := true
end TREAT-REPLY-MESSAGE.
```

Procedures

```
procedure TREAT-REQUEST-MESSAGE(Their_SN, j)
begin BOOLEAN Our_Priority;
    Hgst_SN := max(Hgst_SN,Their_SN);
    Our_Priority := Their_SN > Our_SN
          or ((Their_SN = Our_SN) and j > ME);

    if Using or (Waiting and Our_Priority) then
          Reply_Deferred[j] := true
    fi;
```

Procedures

```
    if not (Using or Waiting)
          or (Waiting and (not Our_Priority) and (not A[j])) then
          A[j] = false;
          send (REPLY(ME), j)
    fi;

    if Waiting and (not Our_Priority) and A[j] then
          A[j] := false;
          send (REPLY(ME), j);
          send (REQUEST(Our_SN, ME), j)
    fi
end TREAT-REQUEST-MESSAGE.
```

Procedures

```
procedure RELEASE-RESOURCE
begin INTEGER j;
    Using = false;
    for i = 1 step 1 until N do
        if Reply_Deferred[j] then
            Reply_Deferred[j] := false;
            A[j] := false;
            Send(REPLY(ME), j)
        fi
    od
end RELEASE-RESOURCE.
```

Authors' response

In the Carvalho and Roucairol modification nodes give an indefinite permission through the REPLY message and not a specific response.

In this way, one node (the one last entering its critical section) is always designated as a "master node" which retains a **special right** - the right to enter its critical section without coordination.

Authors' response

Carvalho and Roucairol have raised an interesting question about what constitutes a **"symmetric"** network algorithm.

Our standards (symmetric behaviour) admit ours and that of Lamport. If the line is drawn in other ways (symmetric text, for example) the door is open to several different algorithms.

A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems

Maekawa, M. (1985).
Technical correspondence.
ACM Transactions on Computer System s, Vol. 3, No. 2, pp. 145-159..