# Desarrollo de Aplicaciones en Red

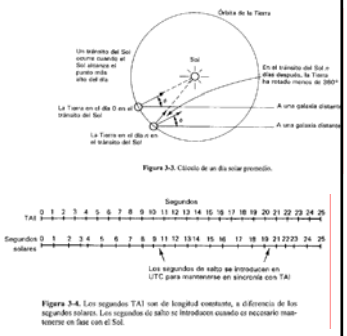
*José Rafael Rojano Cáceres*
*http://www.uv.mx/rrojano*

## General concepts

- The distributed algorithms have the some properties that makes coordination a key for process communication:
  - *Information is distributed around host*
  - *Process takes decisions based in local information*
  - *One point of failure must be avoided (single host)*
  - *There is not global clock*
- The last point is we are going to talking about
  - *It's possible to synchronize clocking in a DS?*

## *How is the real time measure?*

- Universal time → translation and rotation of planets
  - *It's not constant it changes for different factors*
- IAT (International Atomic Time)
- UTC



## Logical Clock

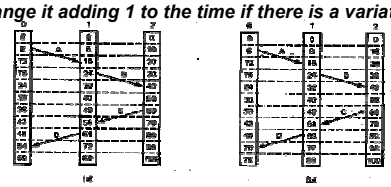
- Chronometer in stead of clock
  - *Counting register and maintaining register*
- In a DS scenario each host has its own clock
- Lamport demonstrated that absolute synchronization is not necessary if two process do not cooperate, the important thing is the process agree about one time.
- In such case when real time it's not important only the internal time of each process we say that we are using logical clocks.

## Synchronization in logical clocks

- Lamport defined the relation "it happens before" a →b, where a happens before b. For this purpose the process need to agree what event happen before.
- In the case of distributed process it happen the same with message that arrive

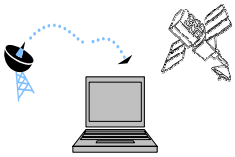
## Lamport algorithm

- *In each equipment there is a clock R to give the time stamp. Each clock run independently for each process. A message is send with the local time, Lamport algorithm check if there is a variation a change it adding 1 to the time if there is a variation*



## Physical Clock

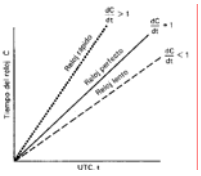
- When it's important that clocks are equal and it doesn't differ from certain magnitude of time we refer as physical clocks.
- The UTC is used to coordinate clocks through:



## Clock synchronization

- Local time < UTC → forward time
- Local time > UTC → backward time?
  - *Slow down time, later synchronizes with UTC*
- Internal clock should synchronizes each φ/2p, where p is the maximum rate of delay and φ is the required difference in seconds

RR1



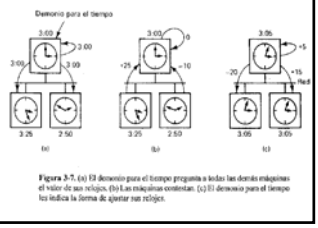
## Cristian's Algorithm

- Designed for equipment where a sever is sync through UTC
- Each φ/2p seconds equipments send to server request of time
- This algorithm should consider the time of propagation through the network:
  - *Sync time* = $T_{utc} + ((T_r - T_e - I) / 2)$

## Berkeley's Algorithm

- Designed for environments where there is no receptor of UTC
- For coordinating the equipments:

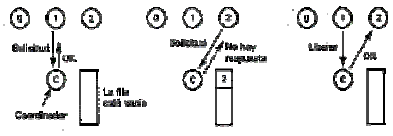
1. A coordinator is elected (dynamically)
2. Setting reference time
   - *Master ask to slaves time*
   - *Select the best time*
   - *Calculate an average time*
3. Update slave clocks
   - *Calculate the deviation of each slave*
   - *Inform the correction to apply for each slave*



## Distributing the computing time

- Scheme centralized → fail of server, bottle neck, etc
- Another view consider that equipments resync periodically
  - For each $I_t$ interval each node send time
  - Wait for time arriving or reach maximum delay
    - Discard extreme values
    - Calculate an average time
  - Update local time with the computed value
  - All local clock get updated

## Mutual exclusion through a centralized algorithm



## Diapositiva 8

**RR1** Especificado por el fabricante
Rafael Rojano, 02/04/2008

## Ricart & Agrawala distributed mutual exclusion algorithm

- *Each process creates a message with the name of critical region, process id and time. According to time, are access region. All the process agree to wait. All request are queue.*

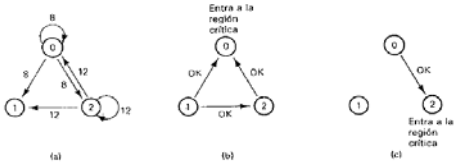



**Figura 3-9.** (a) Dos procesos desean entrar a la misma región crítica en el mismo momento. (b) El proceso 0 tiene una marca de tiempo menor, por lo que gana. (c) Cuando termina el proceso 0, envía un *OK*, por lo que 2 puede ahora entrar en la región crítica.

## Mutual exclusion through tokens

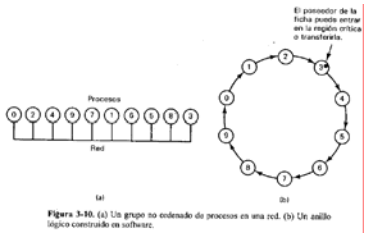



**Figura 3-10.** (a) Un grupo no ordenado de procesos en una red. (b) Un anillo lógico construido en software.

- **Losing of token**
- **Fail of any process**
- **Multiple token in the network**
- **Useless network**

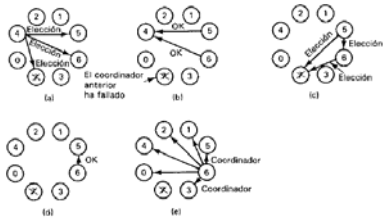
## Comparative of algorithms

| *Algoritgm* | *Message send* | *Message delay* | *problems* |
|---|---|---|---|
| *Centralized* | *3* | *2* | *Coordinator fail* |
| *Distributed* | *2(n-1)* | *2(n-1)* | *Fail of any process* |
| *Token ring* | *1 a ∞* | *0 a n-1* | *Token lost, process fail* |

## Election

- **Many of the seen algorithm require election for a coordinator**
- **This algorithms are:**
  - ***Bully algorithm***
  - ***Ring algorithm***

## Bully Algorithm




1. P process sends Election message to all process with a superior ID
2. If nobody answers, P wins
3. If somebody respond P stop and the cycle start again

**Figura 3-12.** El algoritmo del grandulón para la elección. (a) El proceso 4 hace una elección. (b) Los procesos 5 y 6 responden e indican a 4 que se detenga. (c) Ahora, 5 y 6 hacen una elección. (d) El proceso 6 indica a 5 que se detenga. (e) El proceso 6 gana y se lo informa a todos.

## Ring Algorithm

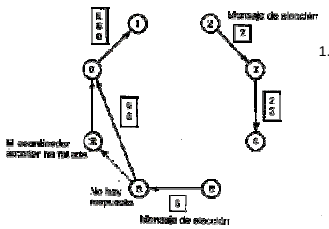



**Figura 3-13.** Algoritmo de elección mediante un anillo.

This algorithm does not use a token, this works in the next way:

1. each process knows their successor, when a process does not see the coordinator it:
   a. Creates a election message and add their ID number
   b. Pass it to the successor, which also add its ID
   c. It go on up to the message pass through all the process
2. The bigger ID is the winner