Tema 5. Interacción y cooperación

Los procesos distribuidos necesitan a menudo coordinar sus actividades. Debemos tener en cuenta la tolerancia a fallos o la exclusión mutua entre otros. Para solucionar este último problema, no se pueden utilizar:

* Variables compartidas.
* Facilidades dadas por un único núcleo central.

La solución se basa en el paso de mensajes.

## COORDINACIÓN DISTRIBUIDA

- Algunos servidores implementan sus propios métodos para sincronizar los accesos a sus recursos.

- Otros no incluyen sincronización, por ejemplo Sun NFS. Estos servidores requieren de:

* Un servicio de exclusión mutua.
* Exclusión mutua distribuida: dar a un único proceso el derecho de acceder temporalmente a los recursos compartidos.

- En otros casos se necesita elegir un proceso de un conjunto para que desarrolle un papel privilegiado (necesario un algoritmo de elección). Por ejemplo en redes Ethernet o inalámbricas.

### Requisitos en la exclusión mutua

**EM1: Seguridad**. En todo momento como máximo en la sección crítica, un proceso ejecutando.

**EM2: Vitalidad.** A todo proceso que lo solicita, se le da permiso para entrar en sección crítica en algún momento. Se evita el abrazo mortal (deadlock) e inanición (starvation).

**EM3: Ordenación.** La entrada en la región crítica debe concederse según la relación sucedió – antes.

### Algoritmo basado en servidor central

El servidor central concede permisos en forma de testigo que concede acceso a la sección crítica. Al salir de dicha sección, el proceso devuelve el testigo.

Suponiendo que no hay caídas ni pérdidas de mensajes:

* Se cumplen EM1 y EM2.
* EM3 está asegurada en el orden de llegada de los mensajes al servidor.

Se necesitarían 2 mensajes para entrar a la sección crítica y 1 mensaje para salir de ella.

Este algoritmo puede producir cuello de botella en el servidor, caídas en el mismo, lo cual nos llevaría a elegir un nuevo servidor y no aseguramos la EM3 o una caída o fallo del proceso en la sección crítica.

### Algoritmo basado en anillo

La exclusión se logra por la obtención de un testigo.

Anillo lógico: se crea dando a cada proceso la dirección de su vecino. El testigo siempre circula por el anillo. Cuando un proceso lo recibe:

* Si quiere entrar a SC lo retiene.
* Si no quiere entrar a SC lo envía a su vecino.
* Al salir de SC lo envía a su vecino.

Con este algoritmo se verifican EM1 y EM2 pero no aseguramos EM3. La obtención del recurso requiere entre 1 y (n-1) mensajes.

Se presentan algunos problemas:

* Carga de la red aunque ningún proceso quiera entrar a SC.
* Si un proceso cae necesita reconfiguración y si además tenía el testigo se debe regenerar el mismo.
* Para asegurarnos de que un proceso ha caído, es necesario varios testigos.
* Desconexión o ruptura de la red.

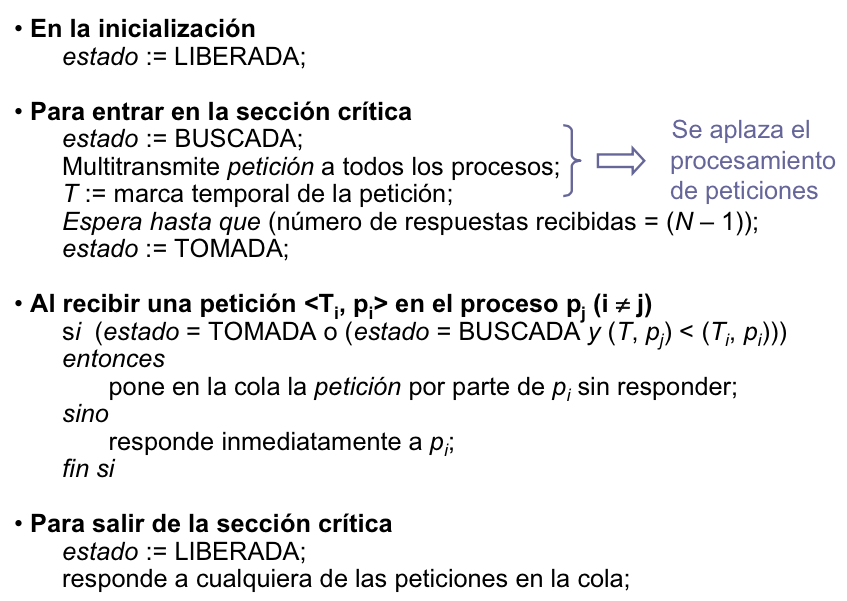
### Algoritmos basados en relojes lógicos

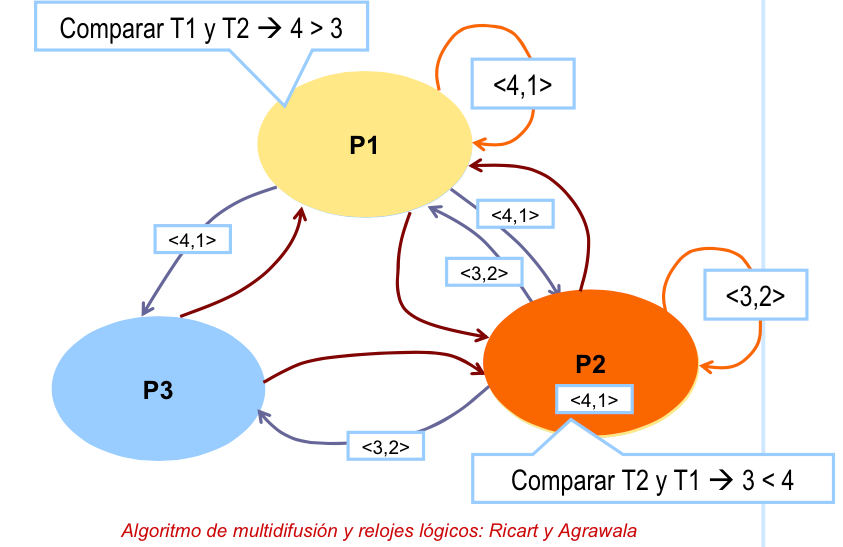
En cada proceso se conocerá la dirección de los demás. Cada proceso posee un reloj lógico.

**RICART Y AGRAWALA**

La idea básica es que cuando un proceso quiere entrar en la SC les pregunta a los demás si puede entrar. Cuando todos los demás le contesten entra.

El acceso se realiza mediante testigo, de manera que cada proceso guarda el estado en relación a la SC: liberada, buscada o tomada. Los mensajes son tuplas: Tupla<Ti,Pi,SCi>.





El número de mensajes necesarios para obtener el recurso varía según las características del servidor:

* Sin soporte multicast: 2(n-1).
* Con soporte multicast: n.
* El algoritmo fue refinado hasta n mensajes sin soporte multicast por Raynal en 1988.

Los problemas que presenta el algoritmo son:

* Más costoso que el del servidor central.
* El fallo de cualquier proceso bloquea el sistema.
* Los procesos implicados reciben y procesan cada solicitud: igual o peor congestión que el servidor central.

En resumen, ninguno de los algoritmos vistos puede tratar el problema de caídas. En el algoritmo de servidor es el que tiene menor número de mensajes, pero supone cuello de botella.

En conclusión, es preferible que el servidor que gestiona el recurso implemente también la exclusión mutua.

## ALGORITMOS DE ELECCIÓN

Procedimiento para elegir a un proceso dentro de un grupo, por ejemplo en el caso de que caiga un coordinador o maestro y haya que seleccionar otro.

La principal exigencia es la elección única. Hay dos algoritmos principales:

* Basado en anillo: Chang y Roberts.
* Algoritmo del matón (bully): Silberschatz.

### Anillo

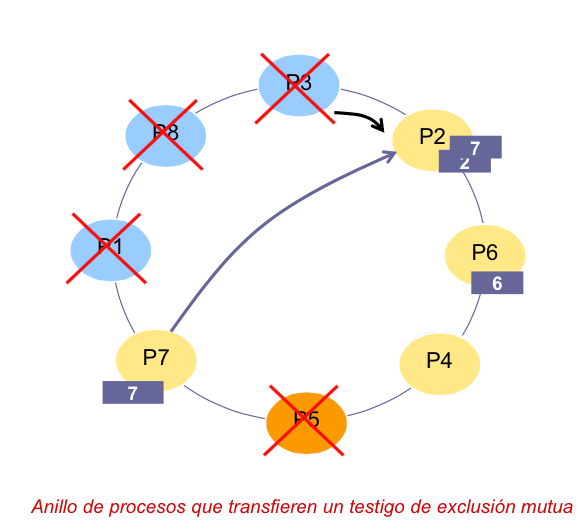
Inicialmente todos los procesos son no-candidatos. Cualquiera de ellos podría empezar una elección, de modo que se marcaría como candidato y se enviaría un mensaje de elección con su identificador.

Cuando un proceso recibe un mensaje de elección:

* Si identificador del mensaje es mayor que el suyo: envía mensaje a sus vecinos.
* Si es menor: si no es candidato se sustituye el identificador y envía mensaje al vecino, marcando como candidato.
* Si es el suyo: se marca como no candidato y se envía mensaje de elegido a su vecino añadiendo su identidad.

Cuando un proceso recibe un mensaje de elegido:

* Se marca como no-candidato.
* Lo envía a su vecino.



**Anillo lógico**: cada proceso sólo sabe comunicarse con su vecino. Se elige al proceso con identificador más alto. Tanenbaum propone un modelo donde los procesos pueden caer.

El número de mensajes para elegir coordinador puede variar:

* en el peor caso: lanza elección sólo el siguiente al futuro coordinador. 3n-1 mensajes.
* En el mejor caso: lanza elección el futuro coordinador. 2n mensajes.

No detecta fallos.

### Bully

Para este algoritmo de elección, todos los miembros del grupo deben conocer las identidades y direcciones de los demás miembros. Se supone una comunicación fiable.

El algoritmo selecciona al miembro superviviente con mayor identificador. Los procesos pueden caer durante la elección. Hay 3 tipos de mensajes:

* mensaje de elección: para anunciar una elección.
* Mensaje de respuesta a un mensaje de elección.
* Mensaje de coordinador: anuncia identidad de nuevo coordinador.

El número de mensajes para elegir coordinador:

* En el caso mejor: se da cuenta el segundo más alto. (n-2) mensajes.
* En el caso peor: se da cuenta el más bajo. 0(n2).

Un proceso inicia una elección al darse cuenta de que el coordinador ha caído:

* Envía un mensaje de elección a los procesos con identificador mayor que el suyo.
* Espera algún mensaje de respuesta. Si el temporizador vence, el proceso actual se erige como coordinador y envía mensaje de coordinador a todos los procesos con identificadores más bajos.
* Si recibe respuesta, espera mensaje de coordinador. Si vence el temporizador, lanza una nueva elección.

Si un proceso recibe un mensaje de coordinador, guarda el identificador y trata a ese proceso como nuevo coordinador.

Si un proceso recibe un mensaje de elección, contesta con un mensaje de respuesta y lanza una elección.

Cuando un proceso se reinicia, lanza una elección a menos que sea el de identificador más alto, en cuyo caso se convertiría en el nuevo coordinador.

