CC5303 – Sistemas Distribuidos

5.- Sincronización

Parte 2

Sebastián Blasco V.

Contenidos

Relaciones Causales

Estado Global

Exclusion Mutua

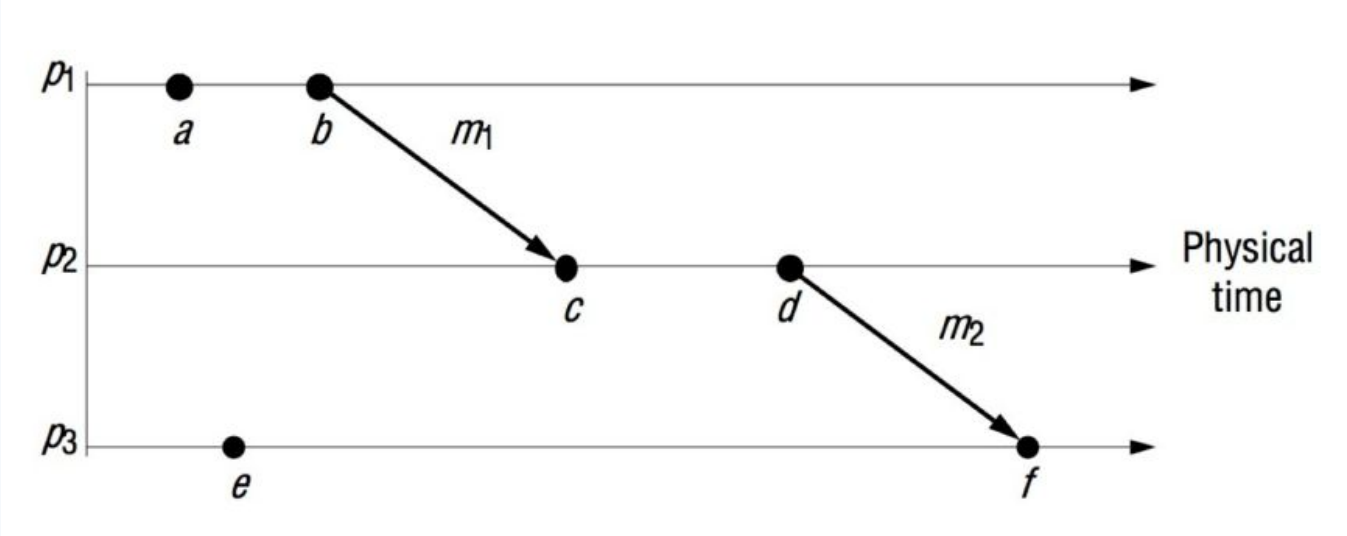
Algoritmos de elección

Recuento Clase Pasada

Lamport define los relojes lógicos, un mecanismo que nos permite establecer un orden parcial para los eventos que se suceden en un SSDD.

3 Reglas:

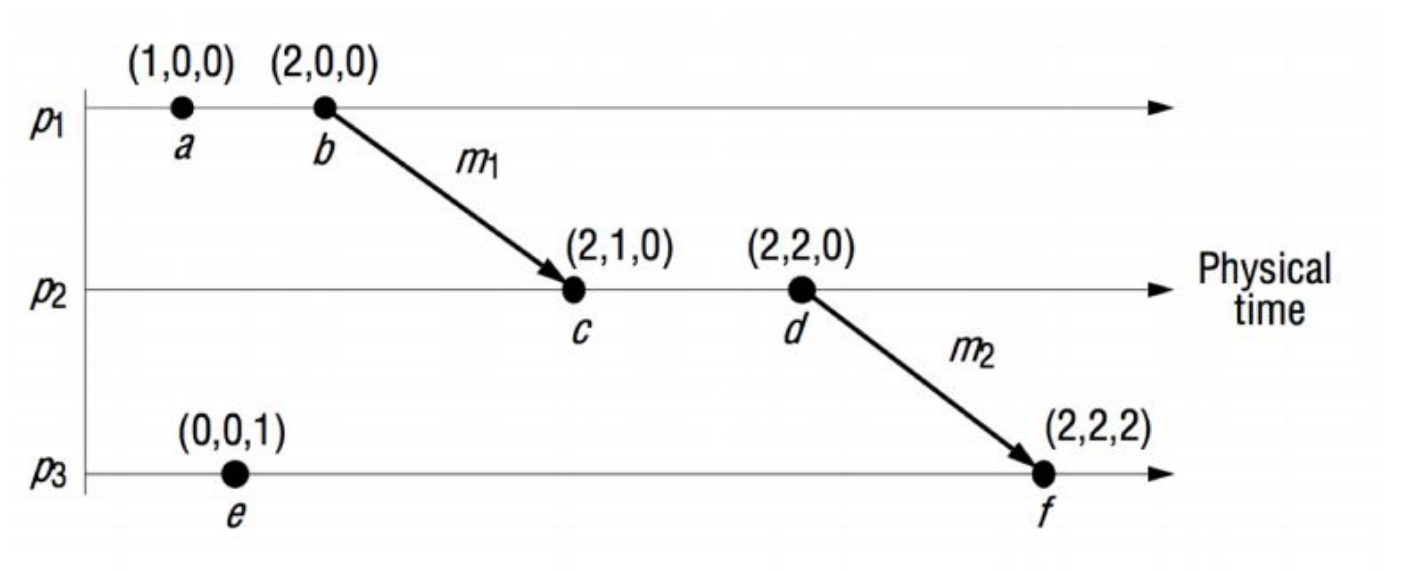
1. Temporalidad en un mismo proceso
2. Paso de mensajes entre procesos
3. Transitividad



Recuento Clase Pasada

Los relojes lógicos no proveen un ordenamiento completo, sólo parcial (Los eventos concurrentes no tienen un orden establecido).

Definimos los relojes vectoriales.



Recuento Clase Pasada

Significado del vector

- **Vi[i]**, número de eventos marcados por ***Pi***
- **Vi[j]**, **j != i**, número de eventos marcados en j que **podrían haber afectado *Pj*** (potencialmente relacionados con ***Pj***)

Comparación de vectores

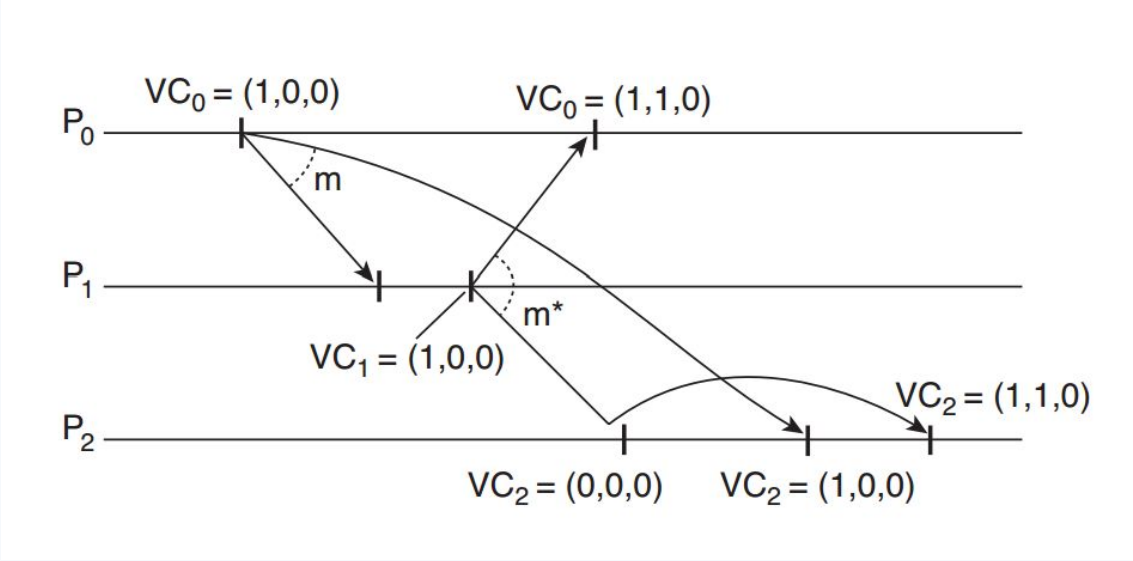
- $V = V' \Leftrightarrow V[j] = V'[j]$, para todo j
- $V \leq V' \Leftrightarrow V[j] \leq V'[j]$, para todo j
- $V < V' \Leftrightarrow V \leq V'$ y V != V'

e1 y e2 son paralelos o concurrentes cuando

$V(e1) \neg\leq V(e2)$ y $V(e2) \neg\leq V(e1)$

Recuento Clase Pasada

Usando relojes vectoriales es posible garantizar que un mensaje sea entregado solo si todos los mensajes que causalmente lo preceden hayan sido recibidos. A dicha restricción se le llama **imposición de la comunicación causal**.



Estados Globales

Para muchos sistemas es relevante poder detectar un estado global. Es decir, un momento del reloj lógico en que el sistema está en un estado “consistente”.

Ejemplos:

- Realizar una recolección de basura
- Detectar terminación
- Detectar DeadLocks
- Recuperación de fallas

Se obtiene haciendo una **captura (snapshot) consistente del sistema**

Estados Globales

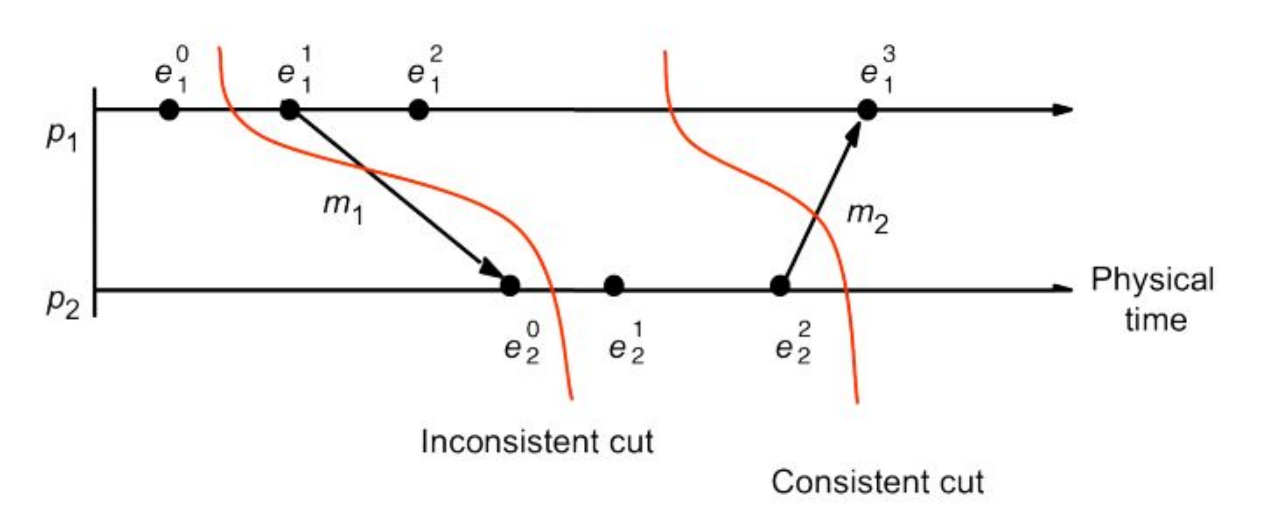
Ejemplo práctico:

- Se tiene el SSDD de un banco. Se necesita calcular el balance monetario de una cuenta pero otros nodos o cuentas podrían estar modificando el valor de la cuenta a consultar.
- En ese caso, simplemente preguntar el balance a cada participante podría generar un resultado incorrecto si es que **algún nodo ha enviado un mensaje a otro nodo, y el mismo aún no ha sido recepcionado**.

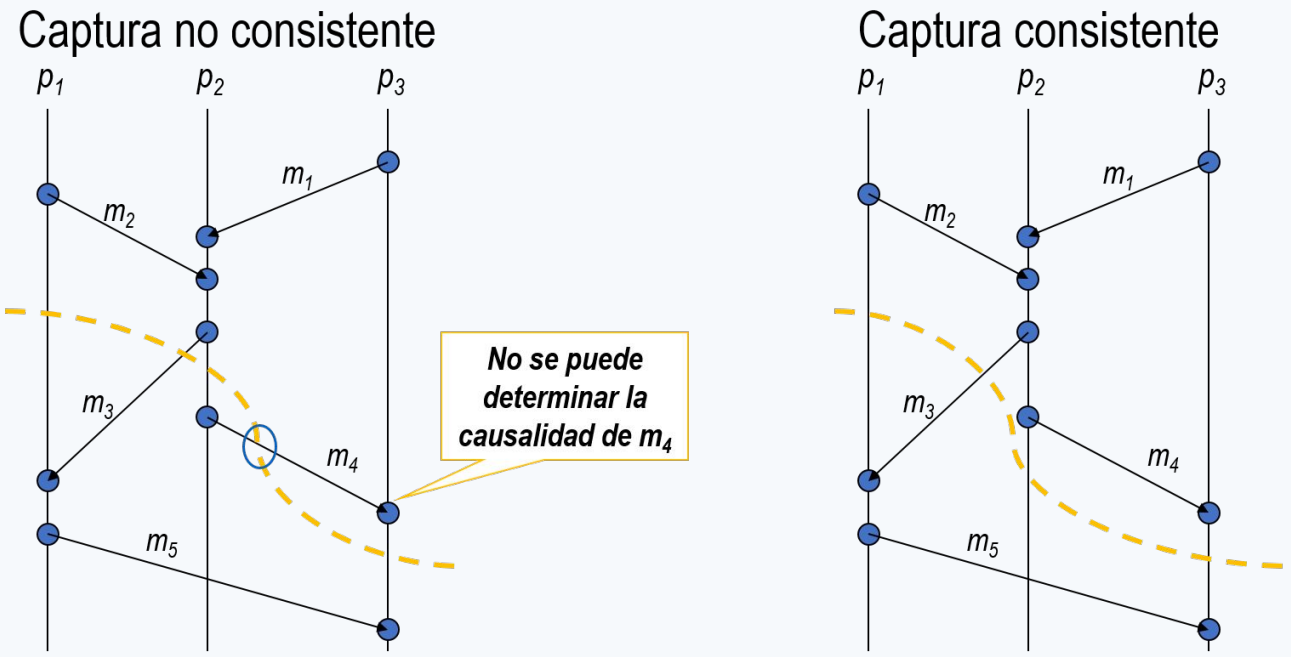
Estados Globales

Se define un **Corte Consistente (C)**

- Por cada evento, también están los que ocurrieron **antes que** él.
- C: $\forall e \in C, f \rightarrow e \Rightarrow f \in C$



Estados Globales



Un corte consistente define un **estado global consistente**

La ejecución se puede caracterizar por una secuencia de estados globales consistentes **S0** → **S1** → **S2** → **. . .**

Estados Globales

Algoritmo de Snapshot: Chandy y Lamport (1985)

- Permite observar estados globales consistentes
  - Captura estados de procesos y de canales de comunicación.
  - Los estados capturados pueden no haber ocurrido conjuntamente en el sistema, sin embargo el conjunto define un estado global consistente
- Suposiciones
  - Existe un canal de comunicación FIFO unidireccional entre toda pareja de procesos emisor-receptor.
  - Ni los procesos ni los canales fallan durante la ejecución del algoritmo.
  - Cualquier proceso puede iniciar el algoritmo en cualquier momento.
  - Los procesos pueden seguir enviando y recibiendo mensajes normales (mensajes de aplicación) durante la ejecución del algoritmo.

Estados Globales

Algoritmo de Snapshot: Chandy y Lamport (1985)

- El proceso que toma la SNAPSHOT (Proceso observador)
  - Guarda su estado local y
  - Envía un mensaje SNAPSHOT (con un token snapshot) a todos los otros procesos.
- Un proceso que recibe un token snapshot por primera vez:
  - Responde con su propio estado local al proceso observador
  - Adjunta un token Snapshot a todos sus mensajes subsecuentes (propagando el token)
- Cuando un proceso que ya había recibido un token snapshot recibe un mensaje libre del token, el proceso notifica de dicho mensaje al proceso observador para incluirlo en la fotografía

Exclusión Mutua

¿Cómo compartir los recursos en un SSDD? Es crucial evitar que los accesos concurrentes **corrompan** a los mismos.

Lo anterior se garantiza accediéndolos "**atómicamente**", es decir solo un proceso a la vez (A esto se le menciona como entrar a la región crítica).

Se deben diseñar mecanismos para garantizar que los procesos tengan accesos **mutuamente exclusivos** sobre los recursos que así lo requieran.

Exclusión Mutua

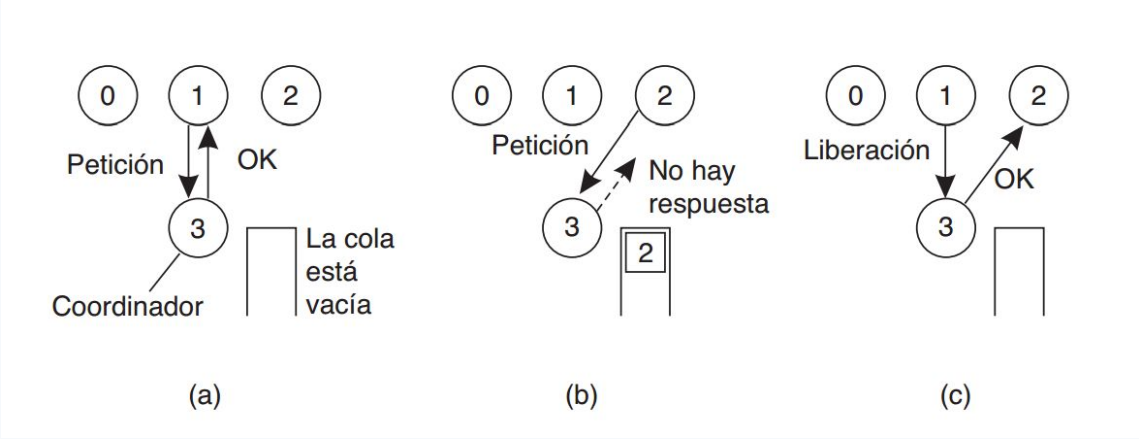
Dos tipos de soluciones:

- Basadas en token
  - Pasando un mensaje especial (sólo hay uno disponible) entre procesos, cuya propiedad garantiza disponibilidad del recurso.
  - Todos tienen chance de acceder a los recursos (Evitan la inanición).
  - Sortean bien los deadlocks.
  - Dependen de que no se pierda el token.
- Basadas en permisos
  - Un proceso que desee usar un recurso necesita el permiso de los demás procesos.

Exclusión Mutua

Algoritmo Centralizado

- Se elige a un proceso que actúa como coordinador, el cual administra el acceso al recurso con una cola de prioridad.
- Los demás procesos le solicitan acceso al coordinador para usar el recurso.
- Similar a lo que se hace en un sistema único.



Exclusión Mutua

Algoritmo Centralizado

- Beneficios
  - Es sencillo
  - Garantiza la exclusión mutua
  - Es justo, no hay inanición.
  - Sólo requiere de 3 mensajes por solicitud de recurso (petición, autorización, liberación)
- Problemas
  - Único punto de falla
  - Potencia cuello de botella

Exclusión Mutua

Algoritmo Descentralizado

- Cada recurso se replica n veces, teniendo cada réplica su controlador de acceso.
- Cuando un proceso desee acceso a un recurso, necesita una votación mayoritaria de $m \geq n/2$ coordinadores.

- ¿Qué pasa si muchos nodos intentan acceder al mismo recurso?
- ¿Cómo se soluciona un bloqueo?

Exclusión Mutua

Algoritmo Distribuido (Ricart y Agrawala (1981))

- Requiere un ordenamiento total de todos los eventos del sistema (TGVC).
- Cuando un proceso desea acceder a un recurso, empaqueta un mensaje con un su tiempo lógico, y lo envía a todos los participantes, incluido él mismo.
- Para cada nodo que recibe el mensaje puede pasar que:
  - él no quiere el recurso, entonces envía solo un OK al emisor
  - él ya tiene el recurso, entonces encola el mensaje localmente
  - él también quiere el recurso, entonces compara el timestamp del mensaje entrante con el suyo. El menor gana, enviando el perdedor un OK.

Exclusión Mutua

Algoritmo Distribuido (Ricart y Agrawala (1981))

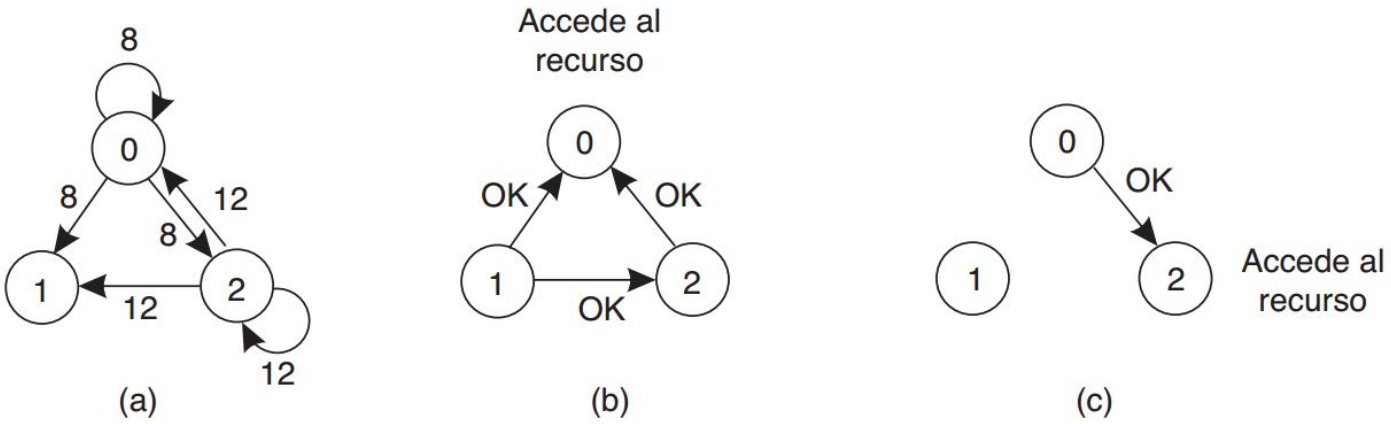


Figura 6-15. (a) Dos procesos desean acceder a un recurso compartido en el mismo momento. (b) El proceso 0 contiene el menor registro de tiempo, de modo que gana. (c) Cuando el proceso 0 se lleva a cabo, envía también un *OK*, de modo que el 2 puede seguir adelante.

Exclusión Mutua

Algoritmo Distribuido (Ricart y Agrawala (1981))

- Se necesitan 2(n-1) mensajes por acceso.
- Sorteado el punto único de fallo!... reemplazado por n puntos de falla.
- Alto tráfico de red.
- Todos los procesos participan en todas las decisiones de acceso a un recurso.. alto trabajo.

Resumen: más lento, más complicado, más caro y menos robusto que el algoritmo centralizado… Puros problemas!

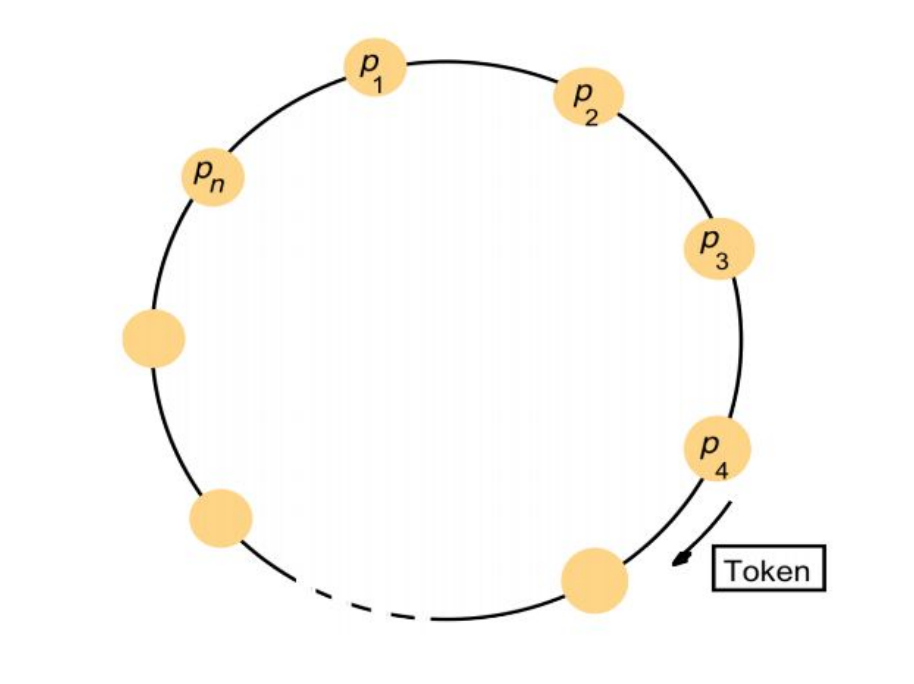
Exclusión Mutua

Token Ring

- Se organizan los procesos en un anillo, cada uno con su sucesor claro.
- El anillo parte cediendo un token al proceso 0.
- El token circula por el anillo, según la aritmética **(k++)%n**
- Cuando un proceso recibe el token, verifica si requiere acceso al recurso. En tal caso, **el proceso tiene garantizado el acceso**. Una vez completado su trabajo con el recurso, pasa el token al siguiente proceso.
- No está permitido usar el token inmediatamente para dos accesos al recurso.

Exclusión Mutua

Token Ring



- Si nadie necesita el recurso, el token circula a alta velocidad por el anillo.

Exclusión Mutua

Token Ring

- OJO! Si el token se pierde, debe reponerse.
  - Es difícil determinar que el token está perdido, el tiempo de espera (en escenarios esperados) por el mismo podría ser muy largo :(
  - Frente a fallas de un proceso, se debe aprovechar la configuración de anillo para llegar al siguiente proceso (siempre y cuando el anillo se mantenga como en su inicio)

Exclusión Mutua

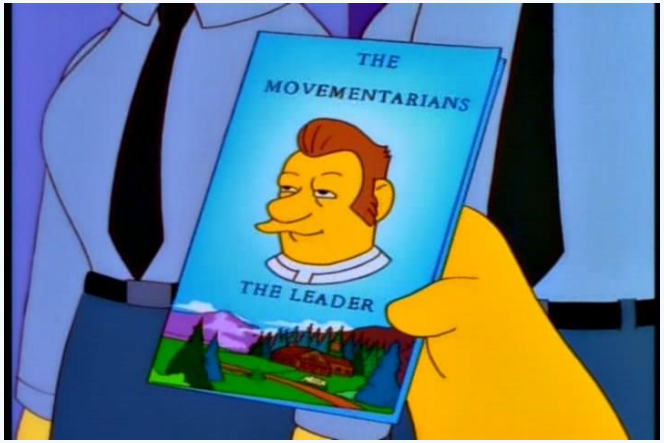
Algoritmo	Mensajes por Entrada/salida	Retraso antes de la entrada	Problemas
Centralizado	3	2	Falla el coordinador
Descentralizado	3mk, k=1, 2, ...	2m	inanición, baja eficiencia
Distribuido	2(n-1)	2(n-1)	Falla de cualquier proceso
Token Ring	1 a infinito	0 a n-1	Pérdida del token, falla de un proceso

Algoritmos de Elección

Muchas veces necesitamos que en una tarea, un proceso tome un rol especial (coordinador, master, etc.). En general, cuando no importa qué proceso tenga dicha responsabilidad, empleamos un algoritmo de elección.

- Caída de máster
  - Cualquiera puede tomar su rol… pero todos deben estar de acuerdo.
- Cualquier nodo puede iniciar una elección
  - Podrían haber N elecciones simultáneas… pero en todas deberíamos llegar al mismo consenso.

Algoritmos de Elección



Dado que los procesos son idénticos, los distinguiremos por un número (un id, ej. dirección de red, pid, etc.). **La elección intenta localizar el proceso con el número más grande**.

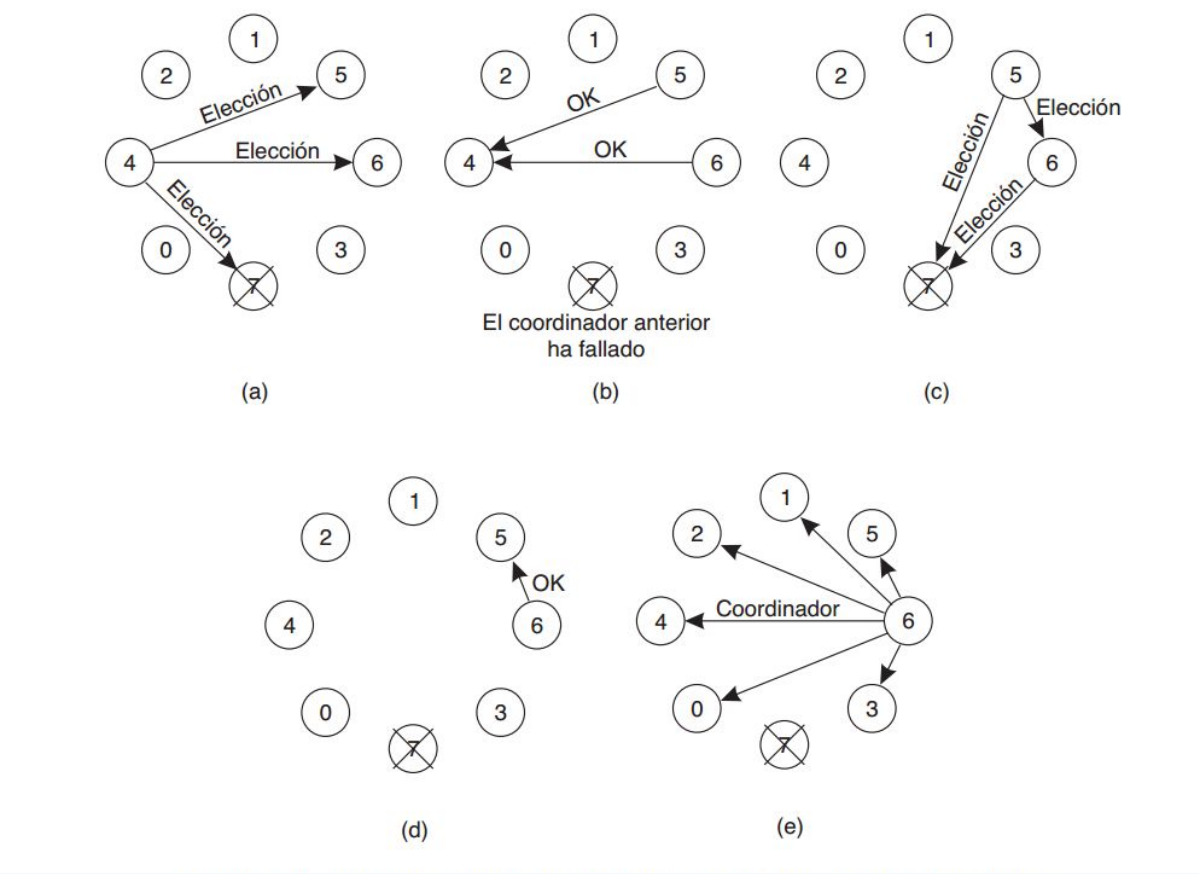
Algoritmos de Elección

Algoritmo del Matón (bully) (García Molina 1982)

- Cuando cualquier proceso advierte que el coordinador ya no está respondiendo peticiones, inicia una elección.
- Un proceso P celebra una elección de la siguiente manera:
  - P envía un mensaje de ELECCIÓN a todos los procesos con números superiores.
  - Si ningún proceso responde, P gana la elección y se convierte en el coordinador.
  - Si uno de los procesos superiores responde, toma el mando, y levanta una nueva elección siguiendo los mismos pasos.
- Finalmente, el nodo elegido debe comunicar a todos los demás nodos que él es el líder.

Algoritmos de Elección

Algoritmo del Matón (bully) (García Molina 1982)



Algoritmos de Elección

Algoritmo del Matón (bully) (García Molina 1982)

- #mensajes
  - Mejor caso: el que tiene el segundo id más grande es el único que detecta la caída:
    - N − 1 mensajes
  - Peor caso: el que tiene id menor es el que detecta la caída
    Cada uno inicia una elección:
    - N − 1 elecciones O(N^2) mensajes

- Ventaja
  - Tolerancia a la falla de algún proceso durante la elección

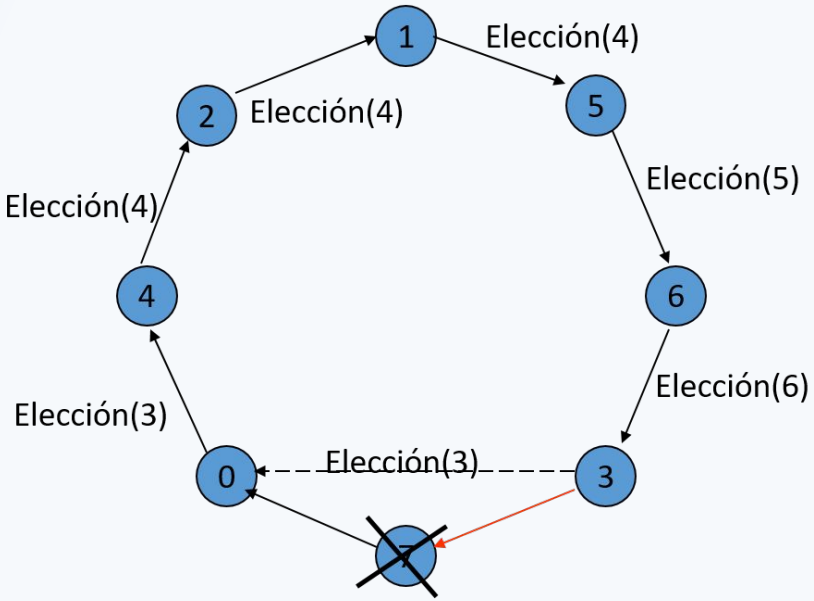
Algoritmos de Elección

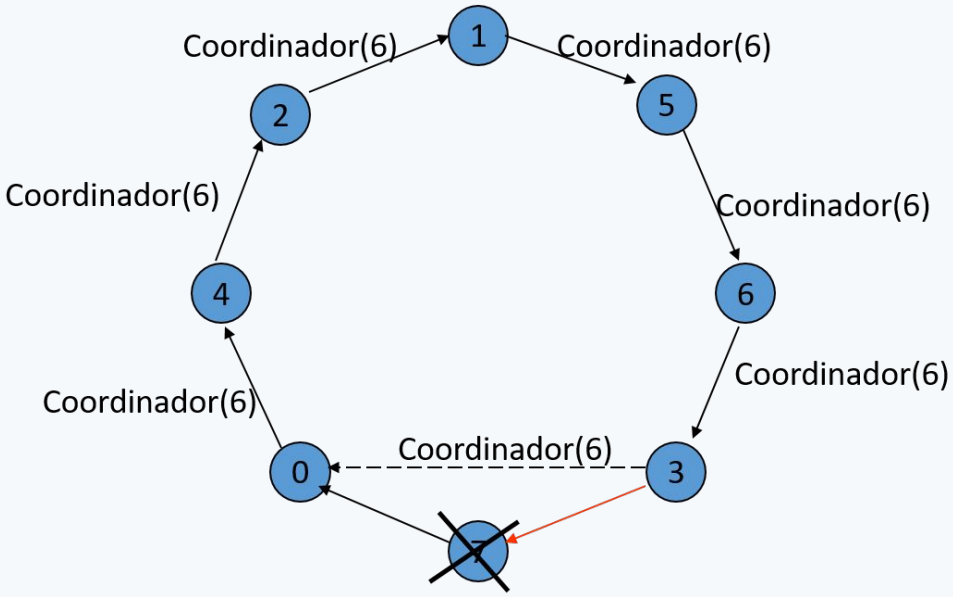
Algoritmo Ring-Based

- Se ordenan los procesos lógicamente en un anillo.
- Si cualquier nodo advierte la necesidad de un líder, elabora un mensaje de ELECCIÓN que incluye su #proceso y lo envía por el anillo al siguiente nodo disponible.
- En cada paso, cada nodo agrega su #proceso al mensaje, postulándose como un candidato.
- Cuando el mensaje regresa al nodo que gatilló la elección, el nodo crea un mensaje COORDINADOR el cual lleva el #proceso mayor que se registró en la elección.
- El mensaje da una última vuelta, notificando a todos los nodos del anillo del nuevo líder.
- No se basa en un token.

Algoritmos de Elección

Algoritmo Ring-Based





Algoritmos de Elección

Algoritmo Ring-Based

- #mensajes:
  - Peor caso: el líder es el vecino anterior a quien inició la elección
    - N − 1 mensajes para encontrar al líder
    - N mensajes para propagar al líder

Total: 2N − 1 mensajes