Sincronización y Coordinación

Desarrollo de Sistemas Distribuidos

Importancia de la sincronización

Sistemas centralizados

- Reloj compartido

Sistemas distribuidos

- Diferentes relojes

Importancia de la sincronización

Conocer a qué hora del día ocurren diferentes eventos en una computadora.	
La precisión varía en cada caso.	
Ejemplos:	La herramienta *make*
	Programas de autenticación de mensajes
	Sistemas de respaldo de archivos.

Sincronización de relojes físicos

Sincronizar los relojes de todas las estaciones del sistema distribuido.

Que todos los equipos tengan la misma hora simultáneamente.

Cada computadora dispone de reloj físico

- Basado en un cristal de cuarzo que oscila a determinada frecuencia
- Programar para generar interrupciones a intervalos regulares
- Llevar la cuenta del tiempo (hora y fecha)

Fecha del sistema

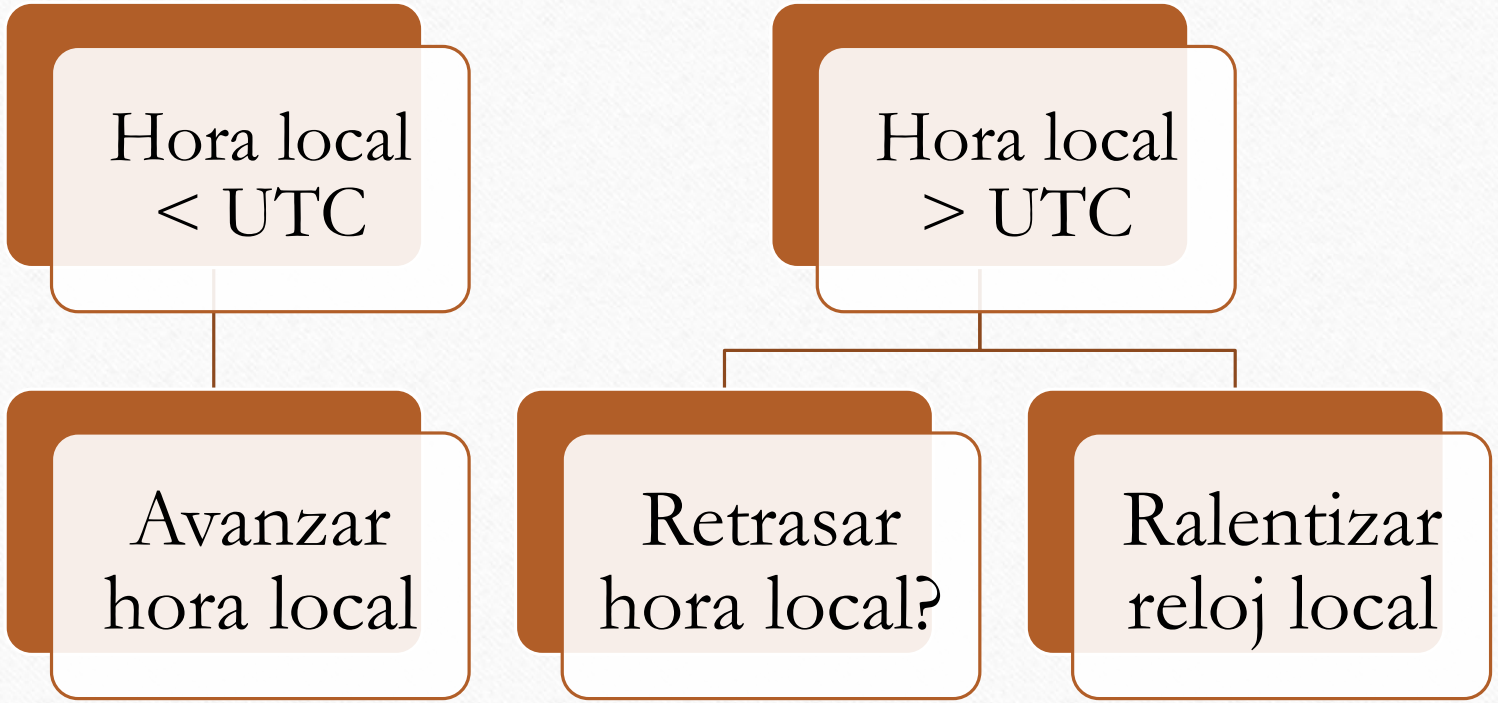
Sincronización de relojes físicos

- Es prácticamente imposible que dos relojes “iguales” oscilen a la misma frecuencia.
- Los relojes basados en cristales están sujetos a una desviación
  - Cuentan el tiempo a velocidades distintas
  - Progresivamente sus contadores van distanciándose.
  - Diferencia acumulada en el periodo de oscilación va creciendo.

Tiempo Universal Coordinado (UTC)

- Disponible a través de emisoras de radio que emiten señales horarias
- Mediante satélites geoestacionarios y GPS
- Precisión de 0,1 a 10ms, 0,1 ms y 1ms

Coordinación con UTC



Algoritmo de Cristian

Un Servidor de Tiempo (sincronizado con UTC)

Las demás estaciones se sincronizan con él

- Cada $\delta/(2\rho)$ segundos enviar un mensaje al servidor para preguntar la hora actual
- El servidor responde con un mensaje con la hora actual T_{UTC}
- Actualizar su reloj
- Considerar el error cometido

Algoritmo de Cristian

- Problemas
  - Un único servidor
  - Posibilidad de un fallo
- Varios servidores de tiempo sincronizados
- No contempla malfuncionamientos o fraude
  - Algoritmo Marzullo (1984)

Sincronización entre equipos

La sincronización horaria entre los dos equipos es más importante que mantener mínima la diferencia con la hora UTC

Diferencia horaria entre los dos equipos después de haberse sincronizado

- $2\rho\Delta t$
- ρ máxima velocidad que deriva de un reloj hardware
- Δt momento transcurrido después de haberse sincronizado

Garantizar que la sincronización no difiera más de δ segundos

- Sincronizarla cada $\delta/(2\rho)$ segundos

Algoritmos de Berkeley

Elegir un coordinador (Maestro)

Establecer la hora de referencia

- El Maestro pregunta la hora periódicamente
- Elige solamente horas “razonables”
- Calcula una hora promedio

Actualizar los demás relojes

- Calcula desviación de cada equipo
- Envía la corrección a aplicar

Algoritmo de Berkeley

El algoritmo realiza un promedio tolerante a fallos con los tiempos recibidos, considera solamente el subconjunto de relojes que no difieren más de una cierta cantidad. Los demás no están funcionando bien.

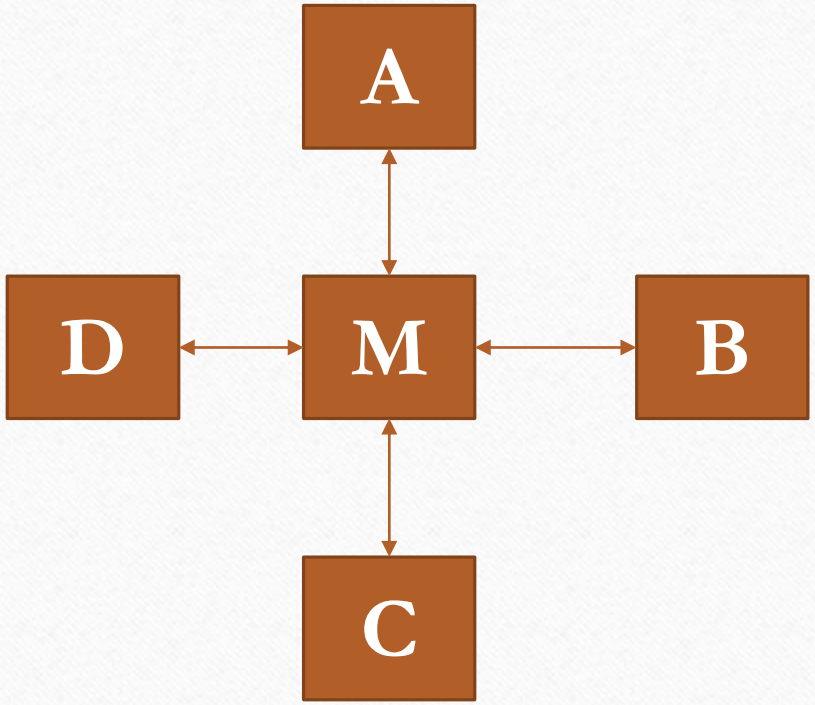
Algoritmo de Berkeley

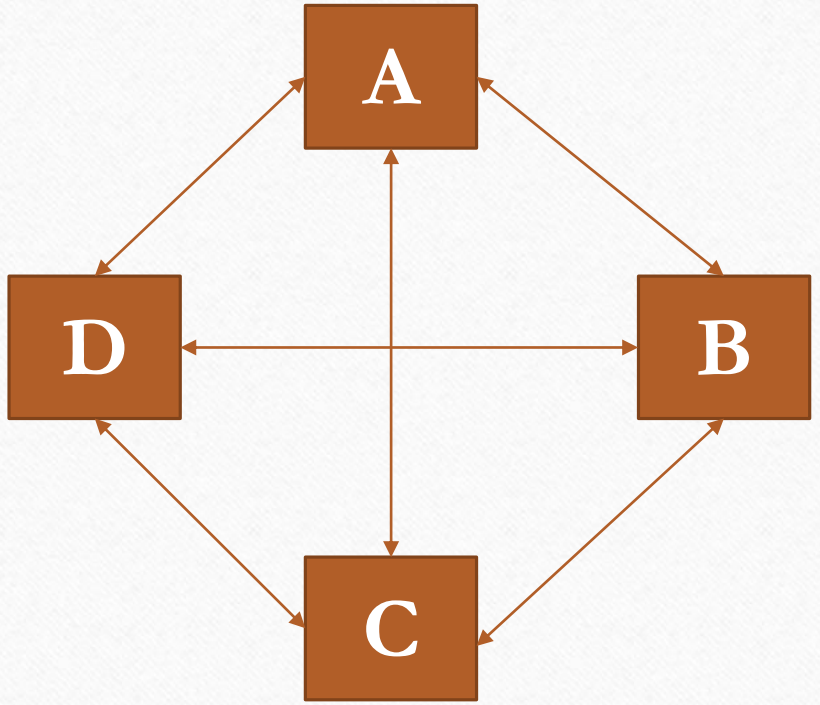
Si el maestro falla

- No pregunta
- No envía modificaciones
- Modificaciones demasiado “extrañas”

Se elige otro coordinador

Algoritmos distribuidos





Casos de estudio

- NTP
- SNTP

Relojes lógicos

- Es complicado sincronizar perfectamente varios relojes distribuidos
- No se requiere estar sincronizados con una hora UTC
- Dos procesos que no interactúan, no requieren sincronización

Algoritmos distribuidos

Se deben conocer los tiempos de propagación

Todos los equipos deben sincronizarse periódicamente

Al comienzo de cada intervalo, todos difunden su hora local

Esperan un tiempo para recibir la hora de los demás equipos

- Excluir valores extremos
- Calcular desviación media

Actualizar tiempo local con la desviación media

Todos los nodos tienen la misma hora

Relojes lógicos

Necesitamos que los procesos estén de acuerdo en el orden en el que se han producido los eventos.

Relojes lógicos

En algunos entornos no se requiere una sincronización exacta con una hora externa de referencia (UTC)

Trabajar con relojes lógicos

- Solamente importa el orden de los eventos

Relojes lógicos

Relación "Sucedió antes" $a \rightarrow b$ (*a* sucede antes que *b*)

- Dos eventos en el mismo proceso suceden en el orden que indica su reloj común
- En procesos comunicados por un mensaje, el envío es siempre anterior a la recepción

Relojes lógicos

Si dos eventos x y y se producen en procesos diferentes que no intercambian mensajes directa ni indirectamente

- No se cumple $x \rightarrow y$ ni $y \rightarrow x$
- Tales eventos no son concurrentes

Relojes lógicos

Transitividad de la relación $a \rightarrow b$

- Si $a \rightarrow b$ y $b \rightarrow c$ entonces $a \rightarrow c$

Relojes lógicos

Necesitamos medir el tiempo para todos los eventos

- Evento a se le asigna un valor de tiempo $R(a)$
- Todos los procesos estén de acuerdo
- Se cumple $a \rightarrow b \leftrightarrow R(a) < R(b)$

Algoritmo de Lamport

Es un simple contador software monótono creciente

Ese valor no tiene relación con un reloj físico

En cada computadora hay un reloj lógico R

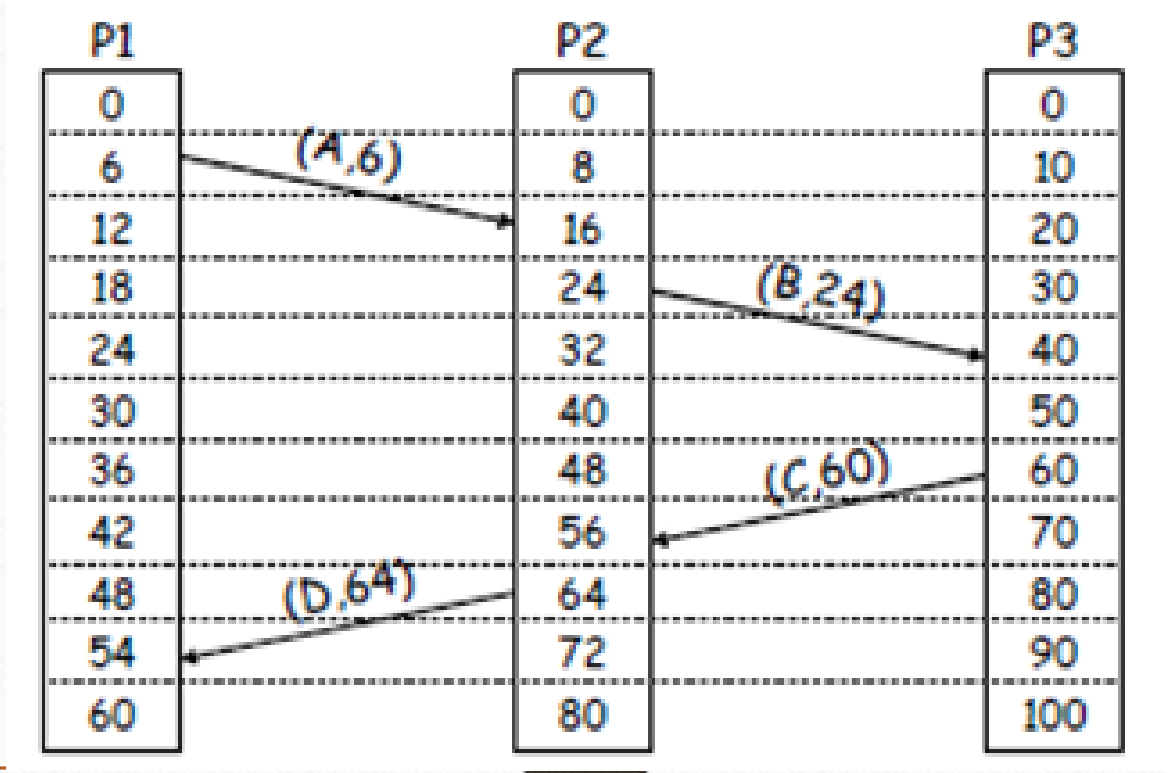
Poner marcas de tiempo a los eventos que se producen $R_p(a)$

- Marca de tiempo del evento a en el proceso p

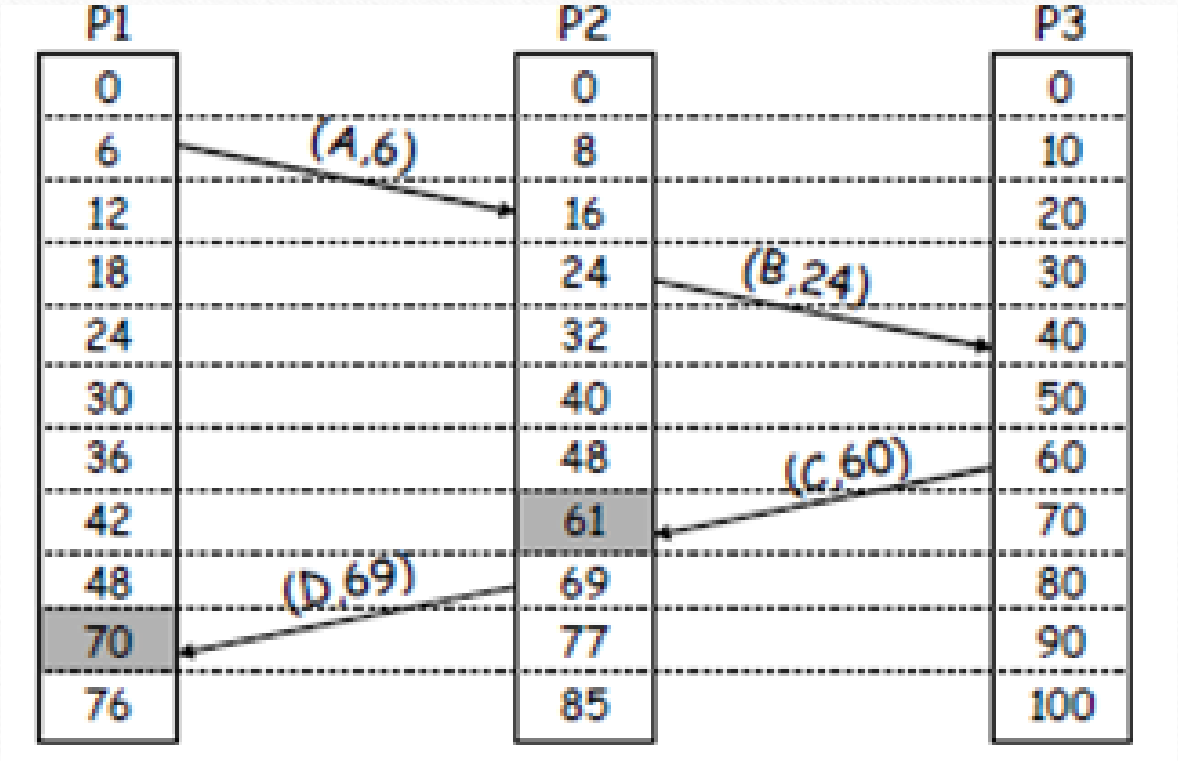
Algoritmo de Lamport

1. R_p se incrementa en una unidad antes de que se produzca cada evento en el proceso: $R_p = R_p + 1$
2.
   - Cuando un proceso p envía un mensaje m se incluye el valor $t = R_p$
   - Cuando el proceso q recibe el mensaje (m, t) calcula $R_q = \max(R_q, t)$
   - Aplicar el paso 1
   - Marcar el evento de recepción del mensaje

Algoritmo de Lamport



Algoritmo de Lamport



Algoritmo de Lamport

Cuando dos eventos, en dos computadoras distintas, ocurren simultáneamente, a la marca de tiempo se le agrega el número del proceso con valor decimal.

- 40.1, 40.2

Exclusión mutua

Regiones críticas en Sistemas Distribuidos

Exclusión mutua

- Resolver el problema de las regiones críticas
- Recursos compartidos que no pueden ser utilizados por más de un proceso al mismo tiempo

Requisitos de los algoritmos

Seguridad

- Sólo un proceso puede ejecutarse dentro de la región crítica

Viveza

- Los procesos que deseen entrar en una región crítica deben hacerlo en algún momento

Orden

- La entrada a la región crítica deberá hacerse en el orden “sucedió antes” definido por Lamport

Exclusión mutua

Dos enfoques

- Centralizado
- Distribuido

Algoritmo centralizado

Utilizar un proceso coordinador de la región crítica

Los procesos envían una solicitud al coordinador

Este algoritmo cumple con las 3 condiciones

Tres mensajes

- Solicitud
- Concesión
- Liberación

Algoritmo centralizado

Inconvenientes

- Fallo de los clientes en la región crítica
- Fallo del coordinador
- Cuello de botella del coordinador

Algoritmos distribuidos

- No se necesita un coordinador
- Los procesos envían peticiones a los demás procesos
- Construyen mensajes de petición

Algoritmo por marcas de tiempo

Cuando un proceso quiere entrar en la región crítica

Envía un mensaje de petición a todos los procesos del grupo

- Nombre de la región crítica
- Su identificador de proceso
- Hora actual

Algoritmo por marcas de tiempo

Cuando un proceso recibe una petición

- Si el receptor no está en la región crítica y no quiere entrar envía un OK
- Si el receptor está en la región crítica, encola la recepción
- Si el receptor quiere entrar a la región crítica, compara su marca de tiempo con la del mensaje
- Si su marca es menor, lo encola
- Si no le envía un mensaje OK y su petición

Algoritmo por marcas de tiempo

Cuando un proceso sale de la región crítica

- Envía un OK a todos en su cola
- Borra esas peticiones de su cola

Algoritmo por marcas de tiempo

Se consigue la exclusión mutua

Para entrar a la región crítica se necesitan

- $2(n-1)$ mensajes
- n es el número de procesos en el sistema

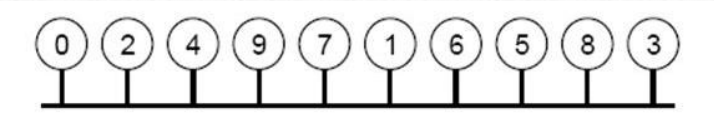
Algoritmo por marcas de tiempo

Inconvenientes

- Hay *n* puntos de fallo
- Más tráfico de la red
- Cuello de botella en todos los eqipos

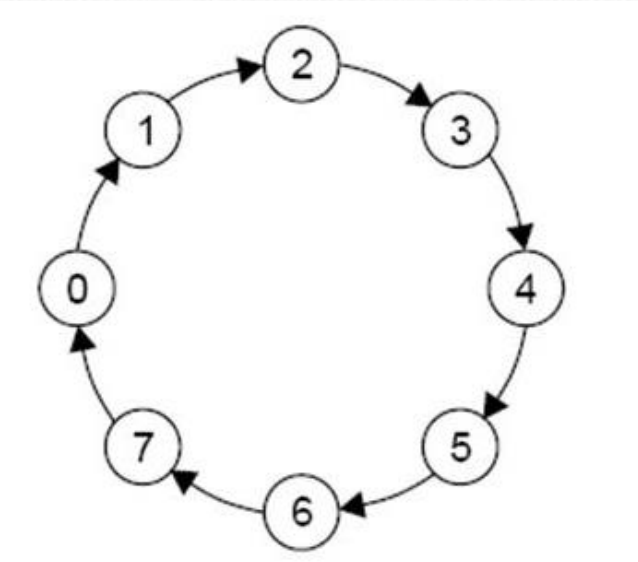
Algoritmo Token-Ring

Dada una serie de equipos interconectados



Formar un anillo lógico

- Cada equipo es un nodo del anillo
- Asignarle una posición en el anillo
- Cada nodo debe saber la posición de su vecino



Algoritmo Token-Ring

Al comienzo al proceso 1 se le da un Token

Cuando el proceso k tenga el token

- Si quiere entrar a la región crítica retiene el token y entra
- Transfiere el token mediante mensaje al proceso $k + 1$

Algoritmo Token-Ring

- Debe partirse de una red sin pérdida de mensajes
- Problemas si falla algún proceso
- Ocupación inútil de la red

Algoritmo de Ricart y Agrawala

- Dados N procesos
- Todos los procesos pueden comunicarse entre sí
- Cada proceso mantiene un reloj lógico de Lamport Ci
- Mensaje de solicitud de entrada (Ci(t), pi)

Coordinación

Algoritmos de elección

Elección del coordinador

Algoritmos distribuidos que necesitan un coordinador

- Sincronización de relojes
- Exclusión mutua
- Restauración de token

Si el coordinador falla

- Es necesario elegir otro coordinador

Algoritmos de elección

Determinar el proceso que va actuar de coordinador

Prioridad de los procesos

- El identificador del proceso es su prioridad
- Es decir P_i tiene prioridad i
- El coordinador es el proceso con mayor prioridad

Una vez elegido el coordinador

- Enviar su identificador a los demás procesos

Algoritmos de elección

El objetivo de los algoritmos de elección es asegurar que al seleccionar un nuevo coordinador todos los demás procesos estén de acuerdo en cuál es el nuevo coordinador

Algoritmos de elección

El objetivo del algoritmo de elección es asegurar que cuando se comience una elección, se concluya con que todos los procesos están de acuerdo en cuál es el nuevo coordinador

Algoritmo Bully

Diseñado por García-Molina en 1982

Suposiciones

- Los mensajes se entregan en T_p segundos
- Un nodo responde a todos los mensajes en T_t segundos
- Los proceso están ordenados física o lógicamente
- Cada proceso tiene un identificador
- Todos los procesos saben cuántos procesos hay en total

Detectar que un nodo ha caído

Si no se responde el mensaje en

- $T = 2T_p + T_t$

Selección del nuevo coordinador

Detectado por un proceso *P*
P envía un mensaje de elección a todos los procesos con identificadores más altos
Espera un mensaje de OK
Si ningún proceso responde P gana la elección
Si cualquiera de los procesos responde, P pierde la elección
El proceso ganador envía un mensaje de coordinador a todos los demás

Un proceso caído re-arranca

Iniciar una elección como la anterior

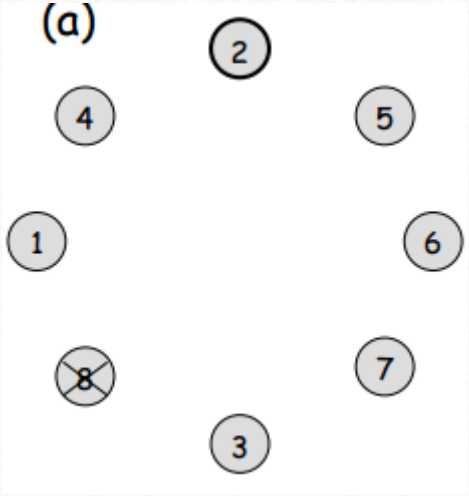
Notas sobre este algoritmo

- En el peor de los casos requiere del orden de n^2 mensajes
- n es el número de procesos
- En el mejor de los casos el proceso con el segundo identificador más alto detecta el fallo

Algoritmo para anillos

Se utiliza cuando

- Los procesos están organizados en una anillo
- No se conoce el número total de procesos
- Cada proceso se comunica con su vecino



Algoritmo para anillos

Un proceso advierte que el coordinador no funciona

Elabora un mensaje de elección

- Su identificador de proceso

Envía el mensaje a su sucesor

Si el sucesor falló

- El remitente lo salta
- Hasta encontrar un proceso activo

En cada paso los remitentes agregan su identificador a la lista del mensaje

Algoritmo para anillos

Cuando el mensaje llega al proceso inicial

Cambia el mensaje a COORDINADOR

Circula nuevamente informando quién es el coordinador

- Y los miembros del nuevo anillo

Cuando el mensaje circula una vez

- Se borra