Tema 2: Comunicación y Sincronización en Sistemas Distribuidos.

2.2 Sincronización en Sistemas Distribuidos

2.2.1. Sincronización de relojes.

- **2.2.1.1. Relojes lógicos.**
- **2.2.1.2. Relojes físicos.**
- **2.2.1.3. Algoritmos para la sincronización de relojes.**

2.2.2. Exclusión mutua.

- **2.2.2.1. Un algoritmo centralizado.**
- **2.2.2.2. Un algoritmo distribuido.**
- **2.2.2.3. Un algoritmo Token Ring.**
- **2.2.2.4. Comparación de los tres algoritmos.**

2.2.3. Algoritmos de elección.

- **2.2.3.1. El algoritmo del grandullón.**
- **2.2.3.2. Un algoritmo de anillo.**

2.2.4. Transacciones atómicas.

- **2.2.4.1. Introducción a las transacciones atómicas.**
- **2.2.4.2. El modelo de transacción.**
- **2.2.4.3. Implantación.**
- **2.2.4.4. Control de concurrencia.**

2.2.5. Bloqueos en Sistemas Distribuidos.

- **2.2.5.1. Detección distribuida de bloqueos.**
- **2.2.5.2. Prevención distribuida de bloqueos.**

Bibliografía:

- Andrew S. Tanenbaum: “Sistemas Operativos Distribuidos”; tema 3, Prentice-Hall, 1996.
- Coulouris, George: “Sistemas Distribuidos: conceptos y diseñ”; tema 11 y 12, Addison-Wesley, 2001
- William Stallings: “Sistemas Operativos”; 4ª Edición, tema 14, Prentice-Hall, 2001.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

¿Cómo conseguimos sincronizar los procesos en un S.D.?

- En un sistema centralizado: regiones críticas, exclusión mutua y sincronización mediante semáforos y monitores.
- En sistemas distribuidos esto es imposible: No existe Memoria compartida.

2.2.1. Sincronización de relojes.

Propiedades S.D:

- La información relevante se distribuye entre varias máquinas.
- Los procesos toman las decisiones sólo basándose en la información disponible en forma local.
- Debe evitarse un único punto de fallo en el sistema.
- No existe un reloj común o fuente precisa del tiempo global.
- En un sistema distribuido no es trivial poner de acuerdo a todas las máquinas en la hora (por ej. el programa *make* de UNIX no funcionaría).
- ¿Es posible sincronizar todos los relojes en un sistema distribuido?.

2.2.1.1. Relojes lógicos.

- Casi todas las computadoras tienen un circuito para registrar el tiempo.
  - RELOJ/CRONOMETRO = cristal de cuarzo sujeto a tensión que oscila con una frecuencia bien definida. Asociados dos registros (contador, mantenedor).
- Cuando el sistema arranca se pide la fecha y la hora, se convierte en un nº de marcas (a partir de una fecha conocida) y se almacena en memoria. Con cada marca de reloj se incrementa el tiempo almacenado.
- En un sistema centralizado, no importa cierto desfase porque todos los procesos utilizan el mismo reloj.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

- En un sistema distribuido, los relojes de las distintas máquinas sufren una pérdida de sincronía (distorsión del reloj). Los programas que esperan un tiempo correcto asociado a un fichero, proceso o mensaje pueden fallar.

Lamport:

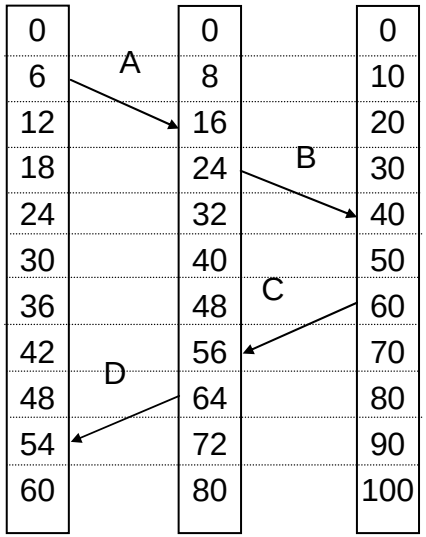
- la sincronización de relojes no tiene que ser absoluta. Si dos procesos no interactúan, no es necesario sincronizar sus relojes.
- lo que importa es que los eventos se produzcan en el orden en que ocurren.

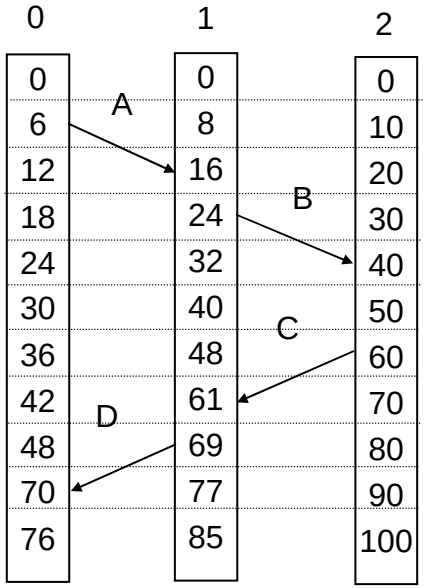
- En general, lo que importa es la consistencia interna de los relojes aunque no coincida con la real (relojes lógicos).
- Algoritmo de Lamport para sincronizar relojes lógicos.
- Cuando los relojes deben ser iguales y además no deben desviarse del tiempo real más de cierta magnitud, hablamos de relojes físicos.

- Def. ocurre-antes-de ($a \rightarrow b$): todos los procesos coinciden en que primero ocurre el evento *a* y después ocurre el evento *b*.
- Si *a* y *b* son eventos en el mismo proceso y *a* ocurre antes que $b \Rightarrow a \rightarrow b$ es verdadero.
- Si *a* es el evento del envío de un mensaje por un proceso y *b* es el evento de la recepción del mensaje por otro proceso $\Rightarrow a \rightarrow b$ es verdadero.

- "Ocurre-antes-de": es una relación transitiva. Si dos eventos x e y, están en procesos diferentes que no intercambian mensajes $\Rightarrow x \rightarrow y$ no es verdadero al igual que $y \rightarrow x$ (son concurrentes).
- Necesitamos asociar a cada evento un valor de tiempo C(*a*) en el que todos los procesos estén de acuerdo. Esto es si se cumple $a \rightarrow b \Rightarrow C(a)<C(b)$.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.





- Cada mensaje acarrea el tiempo de envío (reloj del emisor). Cuando un mensaje llega y el reloj del receptor muestra un valor menor, el receptor adelanta su reloj para que tenga una unidad más que el tiempo de envío.

- En ciertas situaciones dos eventos no deben ocurrir exactamente al mismo tiempo. Podemos asociar el nº de proceso en que ocurre el evento y tiempo separados por un punto. Ej.) en los procesos 1 y 2 ocurre un evento en el tiempo 40, se convierten en 40.1 y 40.2.
- Mediante este método podemos asignar un tiempo a todos los eventos en un Sistema Distribuido, de tal forma que:
  - Si a ocurre antes que b en el mismo proceso, $C(a) < C(b)$.
  - Si a y b son el envío y recepción de un mensaje, $C(a) < C(b)$.
  - Para todos los eventos a y b, $C(a) \neq C(b)$.
- Este algoritmo nos da una forma de obtener un orden total de todos los eventos en el sistema.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.1.2. Relojes físicos.

- En ciertos sistemas (los de tiempo real), es importante la hora real del reloj. Se necesitan relojes físico externos.
- Por razones de eficiencia y redundancia, son recomendables varios relojes físicos ⇒ ¿Cómo los sincronizamos con los relojes del mundo real? ¿Cómo sincronizamos los relojes entre sí?.
- Primero debemos saber: ¿Cómo se mide el tiempo?.

1. El tiempo (s. XVII) se mide de manera astronómica.
   - Día solar: intervalo entre dos tránsitos del sol. Un seg. solar = 1/86400 de un día solar.
   - años 40: periodo de rotación de la Tierra NO es Cte. La velocidad de la tierra está disminuyendo debido a la fricción de las mareas y el arrastre atmosférico. Los días se hacen más largos.
   - la longitud del día sufre variaciones debido a las turbulencias del centro de hierro fundido de la Tierra. La longitud del día es el promedio de una gran número de días dividido por 86400.
2. 1948: inventa el reloj atómico. Seg= tiempo que tarda el cesio 133 para hacer exactamente 9.192.631.770 transiciones. Se eligió esta cantidad para hacerlo coincidir con el seg. solar promedio de ese año.
   - 50 laboratorios del mundo tienen relojes de cesio 133 que comunican el número de marcas a la Oficina Internacional de la Hora (BIH) en París. La oficina hace un promedio para dar el Tiempo Atómico Internacional (TAI).
   - Pero 86400 segundos TAI son unos 3 mseg < que el día solar medio. Con el paso de los años TAI difiere más del tiempo solar.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

- BIH introdujo los segundos de salto, siempre que TAI discrepe con el tiempo solar en 800 mseg = tiempo coordenado universal, UTC.
- SO necesitan de un software especial para registrar los segundos de salto conforme sean anunciados (hasta ahora se han introducido 30).
- ¿Cómo comunicar UTC a quién necesiten un tiempo preciso?
- uso radio corta (WWV) y de satélites.
- requieren un conocimiento preciso de la posición relativa del emisor y receptor, para compensar el retraso de la propagación de la señal.

2.2.1.3. Algoritmos para la sincronización de relojes.

- Si una máquina tiene un receptor WWV ⇒ el resto de las máquinas deben estar sincronizadas con ella.
- Si no es así, cada máquina lleva su propio registro del tiempo ⇒ debemos mantener el tiempo de todas las máquinas tan cercano como sea posible.

Algoritmos: Modelo genérico.

- Se supone que cada máquina tiene un cronómetro que provoca una interrupción *H* veces por seg.
- Cuando se detiene el cronómetro se añade 1 a un reloj software, que mantiene el núm. de marcas *C* a partir de una fecha acordada.
- Los cronómetros no son exactos. Por ej., en teoría si *H*=60 ⇒ se generan 216.000 marcas por hora; en realidad error 10-5 ⇒ [215.998-216.002].
- Todo cronómetro se aleja como máximo una Cte ρ para cada unidad de tiempo llamada **tasa máxima de alejamiento**.
- Si dos relojes se alejan de UTC en la dirección opuesta, en un instante Δt después de ser sincronizados ⇒ pueden estar alejados 2ρΔt.
- Si SO debe garantizar que dos relojes no difieran más de δ ⇒ los relojes deben sincronizarse al menos cada δ/2ρ seg.

$\Delta t -- 2\rho$
$x -- \delta$
$x = \delta / 2\rho$

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

Algoritmo de Cristian

- Sistemas en los que una máquina tiene un receptor WWV y nos proponemos que todas las máquinas se sincronicen con ella.
- Algoritmo: De forma periódica, en un tiempo $< \delta/2\rho$ seg, cada máquina envía un mensaje al servidor para solicitar el tiempo actual. La máquina responde con un mensaje con el tiempo actual C_{UTC}. Problemas:
- Si el reloj del emisor es más rápido, $C_{UTC} < C$. Solución: incrementar el reloj de forma reducida hasta que se haga la corrección.
- El servidor tarda un tiempo en responder $\neq 0$. Puede ser grande y depende de la carga de la red. Solución: intentar medirlo.

Algoritmo de Berkeley

- El servidor de tiempo está activo y realiza un muestreo periódico de las máquinas para preguntar su tiempo.
- Con todos los tiempos calcula el tiempo promedio y todas las máquinas ajustan sus relojes.

Algoritmos con promedio

- Los métodos anteriores son centralizados!!.

Algoritmo descentralizado:

- El tiempo se divide en intervalos: el i-ésimo se inicia en *T0+iR* y va hasta *T0+(i+1)R*, con *T0* tiempo inicial y *R* un parámetro del sist.
- Al inicio de cada intervalo, cada máquina transmite su tiempo al resto.
- Después cada máquina reúne las transmisiones del resto que lleguen en un intervalo S.
- Con lo tiempos reunidos se calcula un valor promedio al que se ajusta su valor:
- Promediar todas los tiempos.
- Descartar los m valores menores y mayores y promediar el resto.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.2. Exclusión mutua.

- La exclusión mutua debe estar garantizada: en un instante dado, sólo un proceso estará en la sección crítica.
- Un proceso que no está en su sección crítica no puede interferir al resto.
- Ausencia de interbloqueo e inanición.
- No se pueden hacer suposiciones sobre las velocidades o el número de procesos.
- Un proceso permanecerá un tiempo finito dentro de su sección crítica.

2.2.2.1. Un algoritmo centralizado.

- Simular el modo de actuación de los sistemas centralizados.
  - Elegir un proceso como coordinador.
  - Siempre que un proceso desee entrar en la sección crítica, envía un mensaje de solicitud al coordinador.
  - Si ningún otro proceso está en esa sección crítica, el coordinador concede el permiso.
  - En otro caso, no responder o enviar un mensaje denegando el permiso, el proceso queda bloqueado en espera.
  - Cuando el proceso sale de la sección crítica, libera su recurso.
  - El coordinador toma el primer elemento en espera y envía a ese proceso un mensaje otorgando el permiso.

1. Sólo un nodo toma el control
2. Toda la información se concentra en este nodo (identidad y ubicación de los recursos y estado de asignación).

- Se garantiza la exclusión mutua, es justo (orden de llegada), no existe aplazamiento indefinido, es fácil, solo requiere tres mensajes (solicitud, otorgamiento, liberación). Se puede utilizar para asignación de recursos.
- Limitaciones: centralizado, si falla el servidor los procesos bloqueados, un sólo coordinador constituye un cuello de botella en un gran sistema**.**

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.2.2. Un algoritmo distribuido.

1. Todos los nodos disponen de igual cantidad de información.
2. Cada nodo tiene una representación parcial del sistema total y debe tomar decisiones basándose en ella.
3. Todos tienen igual responsabilidad en la decisión final.
4. Todos dedican igual esfuerzo en esta decisión.
5. El fallo de un nodo no provocará el colapso total del sistema.
6. No existe un reloj común.

- (Ricart y Agrawala) requiere de un orden total en los eventos en el sistema.
- Algoritmo: Cuando un proceso desea entrar en una región crítica, construye un mensaje con el nombre de ésta, su número de proceso y la hora actual.
  - Envía el mensaje a todos los procesos incluido él mismo.
  - Si el receptor no está en la sección crítica y no desea entrar, envía un mensaje de OK al emisor.
  - Si el receptor ya está en la sección, no responde, y deja la solicitud en una cola.
  - Si el receptor desea entrar, pero no lo ha logrado, compara la marca de tiempo con la marca contenida en le mensaje de cada uno. La menor marca gana.
- La exclusión mutua queda garantizada sin bloqueo e inanición.
- El número de mensajes necesarios por entrada es 2(n-1), con n = número de procesos.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.2.2. Un algoritmo distribuido.

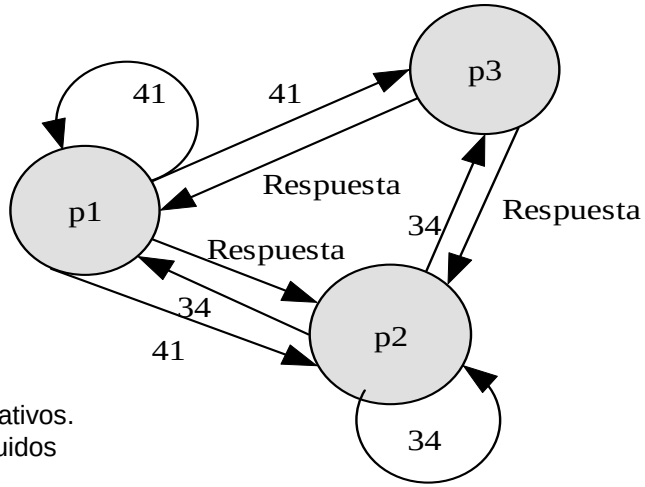
Problemas:

1. Si cualquier proceso falla, el silencio será interpretado como negación de permiso y los procesos quedarán bloqueados.
   - La probabilidad de que uno de los *n* procesos falle es *n* veces mayor que falle un único proceso. Tenemos un algoritmo *n* veces peor y que requiere un tráfico mayor de red.

- Posible solución: el receptor siempre envíe respuesta, otorgando o negando el permiso. El emisor espera hasta conseguir una respuesta o concluye que el destino está muerto. Después de recibir una negativa, el emisor debe bloquearse y esperar un mensaje OK.

2. Debe utilizar primitivas de comunicación en grupo.
3. Todos los procesos participan en todas las decisiones de entrada a las secciones críticas.

- Mejora: permitir que un proceso entre en una sección crítica cuando ha conseguido el permiso de una mayoría simple de los demás procesos (Algoritmo de votación de [Maekawa,1985]).
- Este algoritmo es más lento, más complejo, más caro y menos robusto que el algoritmo centralizado.



Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.2.3. Un algoritmo Paso de Testigo.

- En software, se construye un anillo lógico y a cada proceso se le asigna un posición en el anillo.
- Al inicializar el anillo, se le da al proceso 0 un token, que circula por el anillo. Se transfiere del proceso *k* al proceso *k+1*.
- Cuando un proceso obtiene el token de su vecino, verifica si intenta entrar a una sección crítica. Si es así, el proceso entra en la sección y no hace circular el token hasta salir de la sección crítica.
- Sólo puede haber un proceso en su sección crítica porque sólo uno tiene el token (libre bloqueo). No existe inanición porque los tokens circulan entre todos los procesos en orden.

Problemas :

- El token puede perderse. Es difícil detectar su pérdida, el tiempo que transcurre entre las apariciones sucesivas del token no es fijo ni acotado.
- El algoritmo falla si uno proceso tiene problemas. Es fácil detectar si un proceso está muerto haciendo circular el token. El proceso muerto puede eliminarse del grupo y tomar como vecino al siguiente de éste.

Algoritmo	Mensajes por E/S sección crítica	Retraso antes de E/S sección crítica (en mensajes)	Problemas
Centralizado	3	2	Fallo del coordinador
Distribuido	2(n-1)	2(n-1)	Fallo de cualquier proceso
Token Ring	1 a ∞	0 a n-1	Token perdido, fallo del proceso

2.2.2.4. Comparación de los tres algoritmos.

- Algoritmo centralizado: más sencillo y eficiente.
- Los algoritmos distribuidos son más sensibles a los fallos.
- En sistemas con pocos fallos los tres algoritmos son aceptables.

Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

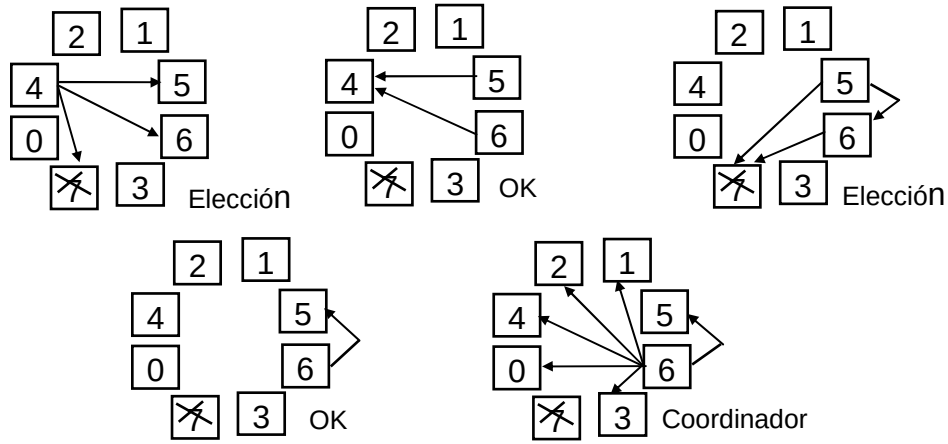
2.2.3. Algoritmos de elección.

- Los algoritmos distribuidos necesitan que un proceso actúe de coordinador, iniciador, secuenciador.
- ¿Cuál de los procesos debe ser el elegido?. Supongamos que:
  - cada proceso tiene un único número.
  - existe un proceso en cada máquina.
  - cada proceso sabe el número de proceso de todos los demás. Desconoce si están activos o inactivos.
- Objetivo: algoritmo de elección garantice la determinación del coordinador

2.2.3.1. El algoritmo del grandullón.

(García-Molina, 1982)

- Cuando un proceso observa que el coordinador ya no responde a las solicitudes, inicia una elección.
  - *P* envía un mensaje *ELECCION* a los demás procesos con un número mayor.
  - Si nadie responde, *P* gana la elección y se convierte en el coordinador.
  - Si uno de los procesos con número mayor responde, toma el control. El trabajo de *P* termina y estos nuevos procesos continúan la búsqueda.
  - Llegará un momento en el que todos se rindan menos uno, el nuevo coordinador. Entonces envía un mensaje a todos los procesos para indicarles que es el nuevo coordinador.
  - Si un proceso inactivo se activa, realiza una elección.

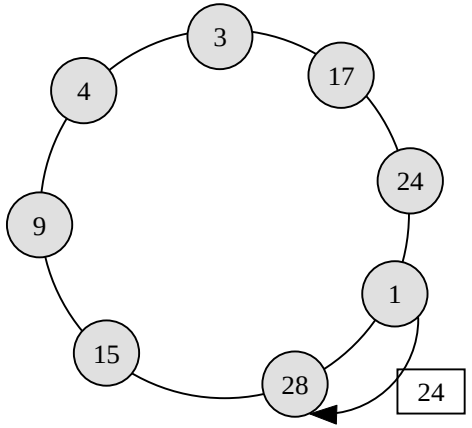


Tema 2.1.- Sincronización en Sistemas Operativos Distribuidos.

2.2.3.2. Un algoritmo de anillo.

(Chang y Roberts, 1979)

- Suponemos que los procesos tienen un orden (físico o lógico).
  - Inicialmente, cada proceso se etiqueta como *no participante* en la elección.
  - Cualquier proceso puede iniciar una elección.
  - Cuando un proceso observa que el coordinador no funciona, se marca como *participante* y construye un mensaje *ELECCION* con su número de proceso y envía el mensaje a su sucesor.
  - Si el sucesor está inactivo, el emisor pasa sobre el sucesor y se lo envía al siguiente.
  - En cada paso el proceso que recibe el mensaje de ELECCION, compara el identificador del emisor con el suyo.
  - Si el identificador es mayor, hace avanzar el mensaje.
  - Si el identificador es menor y él es no participante, cambia el identificador del mensaje por el suyo. Si ya es participante no envía nada.
  - Cuando el mensaje regresa al proceso que lo inició, es porque es el identificador mayor y envía un mensaje COORDINADOR con los miembros del nuevo anillo.



Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.4. Transacciones atómicas.

- Transacción atómica es una abstracción que facilita la construcción de algoritmos que trabajan en paralelo.
- Conjunto de operaciones sobre los objetos que se realizará como un unidad indivisible por los servidores que los gestionan.

2.2.4.1. Introducción a las transacciones atómicas.

- Un proceso anuncia que desea comenzar una transacción con uno o más procesos. Pueden negociar varias operaciones, crear y eliminar objetos y llevar a cabo ciertas operaciones.
- El coordinador anuncia que desea que todos se comprometan con el trabajo.
- Si todos coinciden, los resultados se vuelven permanentes.
- Si uno o más procesos se niegan o fallan, la situación regresa al estado anterior a la transacción.
- Propiedad del todo o nada.

Ej.) Aplicación bancaria: un cliente llama al banco mediante un PC con MODEM, quiere retirar dinero de dos cuentas (A, C) y depositarlo en otra (B):

Transacción T:

a.extrae(catidad1);

b.deposita(cantidad1);

c.extrae(cantidad2);

b.deposita(cantidad2);

- Si la conexión falla después de la primera etapa pero antes de la segunda, la transacción se queda a medio camino.
- Necesitamos agrupar las dos operaciones en una transacción atómica. Debemos volver al estado inicial si la transacción no puede concluir.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.4.2. El modelo de transacción.

- El sistema consta de varios procesos independientes que pueden fallar de manera aleatoria.
- La comunicación es no confiable pero el software subyacente maneja de forma transparente los errores de comunicación.

Almacenamiento estable

- Se trata de dos copias exactas de la información en dos discos. Siempre se actualiza un bloque primero en una unidad y luego en la otra.
- Es adecuado para las aplicaciones que requieren un alto grado de tolerancia a fallos, como las transacciones atómicas.

Primitivas de transacción

- La programación que usa de transacciones requiere de primitivas especiales:
  - abreTransacción()→ trans. Comienza una transacción y se devuelve un TID, un identificador que será utilizado para el resto de las operaciones de la trasacción.
  - cierraTransacción(trans)→(consumado, abortado). Termina la transacción y se intenta un compromiso.
  - abortaTransacción(trans). Se elimina la transacción; se recuperan los valores anteriores.
  - Todas las operaciones (llamadas al sistema o procedimientos de biblioteca) entre un abreT y un cierraT se deben ejecutar todas o ninguna.

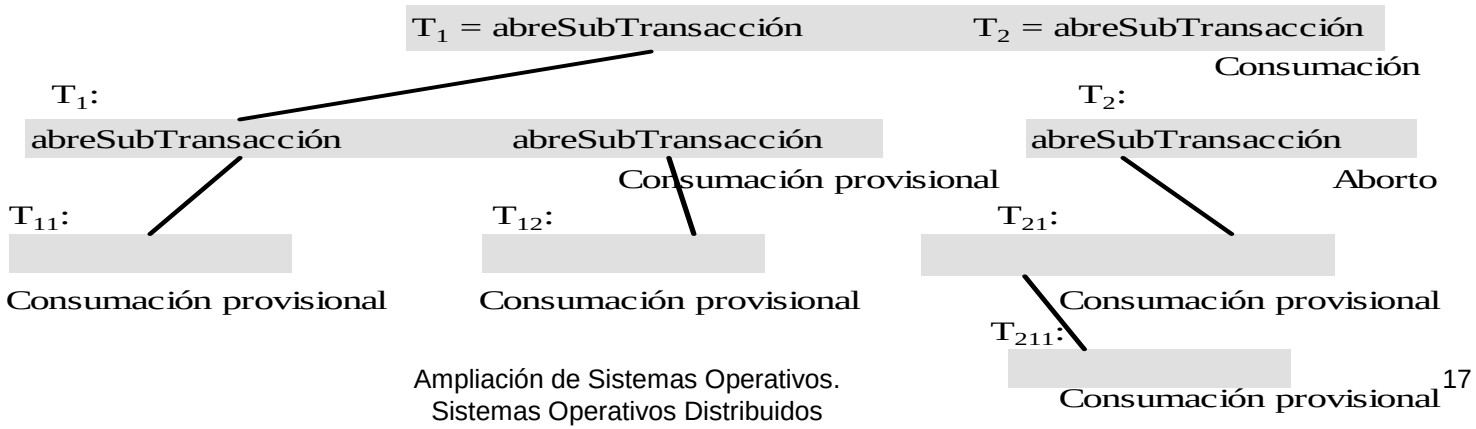
Propiedades de las transacciones

- Aislamiento: Las transacciones concurrentes no interfieren entre sí, los efectos intermedios de una transacción no deben ser visibles para las demás.
- Atomicidad: La transacción ocurre de manera indivisible.
- Permanencia: Una vez comprometida una transacción, los cambios son perdurables, son guardados en disco.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

Transacciones anidadas

- Una transacción de nivel superior puede producir hijos que se ejecuten en paralelo entre sí para mejorar el rendimiento y hacer fácil la programación.
- Problema subtransacciones: la transacción padre puede abortar cuando se han comprometido algunas subtransacciones.
- Reglas para la consumación de transacciones anidadas:
  - Una transacción se puede consumar o abortar sólo después de que se han completado sus transacciones hijas.
  - Cuando una subtransacción finaliza, decide independientemente si consumarse provisionalmente o abortar.
  - Cuando un padre aborta, todas sus subtransacciones son abortadas.
  - Cuando una subtransacción aborta, el padre decide si abortar o no.
  - Si se consuman las transacciones de alto nivel, entonces todas las subtransacciones que se han consumando provisionalmente, se consuman.



Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

Control de Concurrencia

- Problemas relacionados con las transacciones
- Actualizaciones perdidas: tenemos tres cuentas A, B y C con balances 100E, 200E y 300E. De A y C se extrae el 10% de B y se incrementan la cuenta de B dos veces. Después de ejecutar T y U, la cuenta de B debería incrementarse dos veces en un 10% (200 + 20 + 22 = 242E).
- La actualización de U se pierde porque T escribe sin mirar. Ambas han leído el valor antes de escribir el nuevo valor.
- Recuperaciones inconsistentes: tenemos dos cuentas A, B con 200E. V transfiere de A a B 100E y W calcula el balance de todas las cuentas.
- Equivalencia secuencial: Un solapamiento de las operaciones en las que el efecto es el mismo que si se hubieran realizado en orden secuencial se dicen que son secuencialmente equivalente.

Transacción T:	**Transacción U:**
balance = b.obtenBalance();	balance = b.obtenBalance();
b.ponBalance(balance*1.1);	b.ponBalance(balance*1.1);
a.extrae(balance/10);	c.extrae(balance/10);
balance = b.obtenBalance(); 200*E*	
	balance = b.obtenBalance(); 200*E*
	b.ponBalance(balance*1.1); 220*E*
b.ponBalance(balance*1.1); 220*E*	
a.extrae(balance/10); 80*E*	
	c.extrae(balance/10); 280*E*

Transacción V:	**Transacción U:**
a.extrae(100);	unasucursal.totalSucursal();
b.deposita(100);	
a.extrae(100); 100E	total = a.obtenBalance(); 100E
	total = total + b.obtenBalance(); 300E
	total = total + c.obtenBalance(); 300E
b.deposita(100); 300E	● ● ●

Transacción T:	**Transacción U:**
balance = b.obtenBalance();	balance = b.obtenBalance();
b.ponBalance(balance*1.1);	b.ponBalance(balance*1.1);
a.extrae(balance/10);	c.extrae(balance/10);
balance = b.obtenBalance(); 200*E*	
b.ponBalance(balance*1.1); 220*E*	
	balance = b.obtenBalance(); 200*E*
	b.ponBalance(balance*1.1); 242*E*
a.extrae(balance/10); 80*E*	
	c.extrae(balance/10); 278*E*

Transacción V:	**Transacción U:**
a.extrae(100);	unasucursal.totalSucursal();
b.deposita(100);	
a.extrae(100); 100E	
b.deposita(100); 300E	total = a.obtenBalance(); 100E
	total = total + b.obtenBalance(); 300E
	total = total + c.obtenBalance(); 300E
	● ● ●

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

- Operaciones conflictivas: son un par de operaciones cuyo efecto combinado depende del orden en el que se ejecutan.
- Para que dos operaciones sean *secuencialmente equivalentes*, es necesario y suficiente que todos los pares de operaciones conflictivas de las dos transacciones se ejecuten en el mismo orden sobre los objetos a las que ambas acceden.
- Son necesarios protocolos de control de concurrencia: **bloqueo**, **control de concurrencia optimista** y **ordenación por marcas de tiempo**.
- Bloqueo: el servidor bloquea objetos, sólo la transacción que ha realizado el bloqueo tiene acceso.

Operaciones de diferentes transacciones		Conflicto	Causa
Lee	Lee	No	Porque el efecto de un par de operaciones de *lectura* no depende del orden en el que son ejecutadas
Lee	Escribe	Sí	Porque el efecto de una operación de *lectura* y una de *escritura* depende del orden de su ejecución
Escribe	Escribe	Sí	Porque el efecto de un par de operaciones de escritura depende del orden de su ejecución

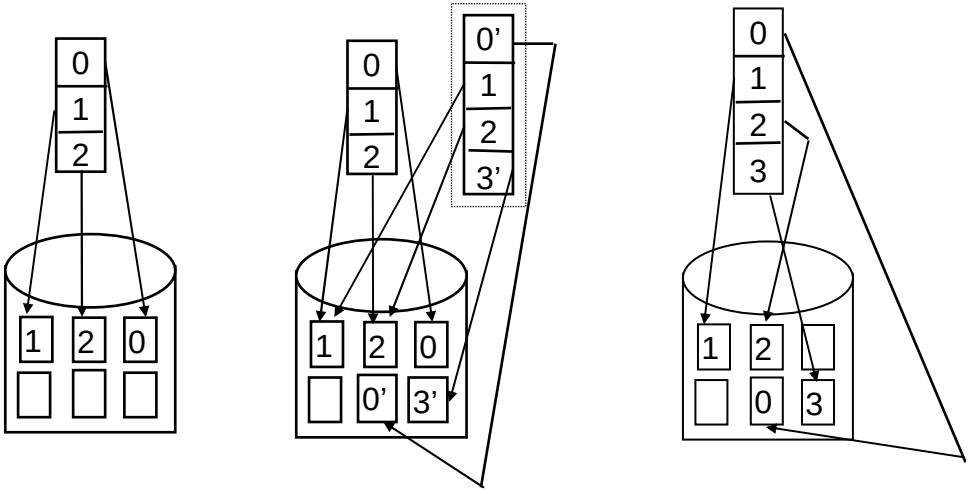
- Control optimista: una transacción avanza hasta que solicita la consumación. El servidor comprueba si hay conflictos, en cuyo caso, la aborta y hay que reiniciar.
- Marcas de tiempo: el servidor registra el tiempo de lectura y escritura de cada objeto. En cada operación se comparan las marcas de tiempo.
- Este control se puede obtener:
  - Los clientes de las transacciones esperen uno por otro,
  - Reiniciando las transacciones después de detectar conflictos,
  - Combinación de las dos anteriores.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.4.3. Implantación.

¿Cómo implementamos las transacciones?

- Necesitamos de métodos que nos permitan realizar transacciones atómicas y abortarlas y secuenciarlas.



Espacio de trabajo particular

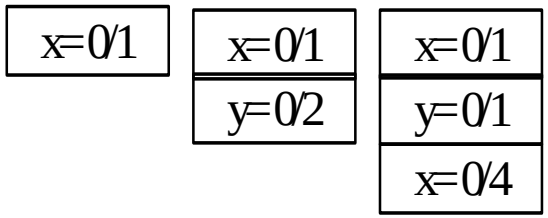
- A cada proceso que inicie una transacción se le otorga un espacio de trabajo particular.
- El espacio particular contiene todos los objetos a los que tiene acceso y hasta que la transacción se comprometa o aborte, las E/S irán a este espacio.
- Problema: el costo de copiar todo el espacio particular.
- Se pueden realizar optimizaciones: sólo copiar en caso de escritura. Mejor sólo copia el i-nodo y los bloques que sean modificados y los bloques añadidos (shadow blocks).
- El proceso que ejecuta la transacción ve el fichero modificado, pero los demás ven el fichero original.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

Bitácora de escritura anticipada (lista de intenciones)

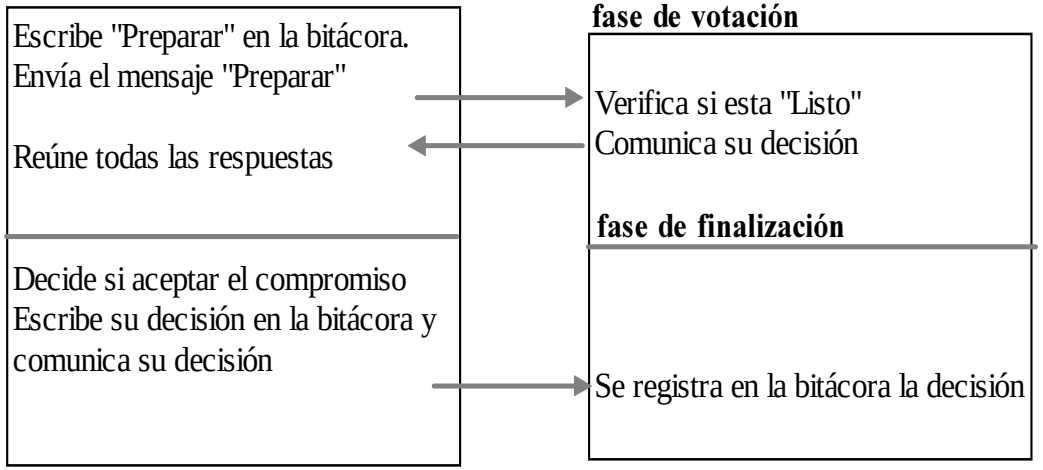
- los ficheros se modifican pero antes se registran los cambios en la bitácora de escritura anticipada en un espacio de memoria estable.
- si la transacción se compromete, las estructuras de datos ya están modificadas.
- si se aborta se pueden restablecer los valores originales.
- la bitácora se puede utilizar para recuperar los fallos.

```
x= 0;
y= 0;
abreTransacción
      x=x+1;
      y=y+2;
      x= y*y;
cierraTransacción
```



Protocolo de compromiso de dos fases

- en un sistemas distribuido, el compromiso puede necesitar la cooperación de varios procesos en ≠ máquinas (cada uno contiene algunas variables, ficheros y BD u otros objetos).
- Este mecanismo es capaz de recuperarse en presencia de fallos.
- Rendimiento: en ausencia de fallos, N mensajes puedeConsumar?, y sus respuestas y N mensajes Consuma (3N).



Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.4.4. Control de concurrencia.

- Cuando ejecutamos ≠ transacciones simultáneamente en ≠ procesadores, se necesita un **algoritmo de control de concurrencia**.

Cerradura o bloqueo (Locking)

- Cuando un proceso necesita leer o escribir en un fichero como parte de una transacción, primero cierra el fichero.
- Este cierre se puede realizar desde un manejador central o local.
- Este esquema es muy restrictivo, se puede autorizar el acceso en caso de lectura.
- El cierre se puede aplicar a elementos más pequeños como un registro, un página: granularidad del bloqueo.
- Una granularidad fina permite un mayor paralelismo, pero necesita un número mayor de bloqueos difíciles de coordinar y con más probabilidad de interbloqueo.
- la adquisición y liberación de cerraduras puede conducir a inconsistencias y bloqueos.
- Se suele utilizar la **cerradura de dos fases**: fase de crecimiento, en el que el proceso adquiere todos los recursos y fase de reducción, los libera. Aplicando este método todas las planificaciones serán secuenciadas.
- Se pueden producir bloqueos si dos procesos intentan adquirir la misma pareja de objetos, en orden opuesto.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

Control optimista de la concurrencia (Kung y Robinson, 81)

- Sólo nos preocupamos de los problemas cuando surjan.
- Los conflictos son raros pero debemos saber como manejarlos.
- Se mantiene un registro de los objetos leídos o escritos. En el momento del compromiso se verifican las demás transacciones para ver si algún objeto ha sido modificado. Si esto ocurre, la transacción se aborta.
- Ventajas: ausencia de bloqueos, máximo paralelismo.
- Desventaja: puede fallar, se pierde la transacción.
- Adecuado para cargas de tamaño no muy grande.

Marcas de tiempo (Reed, 83)

- Asociar al comienzo de cada transacción una marca de tiempo.
- Cada objeto del sistema tiene una marca de cuando fue su última lectura y cuando su última escritura.

Escritura

if ($T_c \geq$ MaxTlectura D && $T_c >$ TescrituraConsumada D)
  realiza una escritura tentativa en D con marca T_c
else aborta la transacción T_c

Lectura

if ($T_c >$ TescrituraConsumada D)
  sea D_s la versión de D con MaxTescritura
  if (se ha consumada D_s)
    realiza una lectura de la versión D_s
  else
    espera hasta consumación de D_s
else aborta la transacción T_c

Tentativa T_c	Otras T_i	
Escritura	Lectura	T_c no debe escribir un objeto que ha sido leído por otra T_i donde $T_i > T_c$, $T_c \geq$ la mayor marca de tiempo de lectura del objeto.
Escritura	Escritura	T_c no debe escribir un objeto que ha sido escrito por otra T_i donde $T_i > T_c$, $T_c >$ la marca de tiempo de escritura del objeto consumado.
Lectura	Escritura	T_c no debe leer un objeto que ha sido escrito por otra T_i donde $T_i > T_c$, $T_c >$ la marca de tiempo de escritura del objeto consumado.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.5. Bloqueos en Sistemas Distribuidos.

- Son más difíciles de evitar, prevenir, detectar y recuperar.

Dos tipos de bloqueos distribuidos:

- bloqueos de comunicación: una cadena circular de envío de mensajes en un sistema sin búferes y
- bloqueos de recursos: los procesos pelean por dispositivos de E/S, ficheros, cerraduras u otros recursos.

- En realidad no existe esa distinción porque los canales de comunicación, búferes son recursos que se pueden modelar como bloqueos de recursos. Además es raro que la condición de espera circular se de una comunicación.

4 estrategias para el manejo de los bloqueos:

- El algoritmo de la avestruz (ignorar el problema).
- Detección (permitir que ocurran bloqueos, detectarlos e intentar recuperarse de ellos).
- Prevención (hacer que los bloqueos sean imposibles).
- Evitarlos (evitar los bloqueos mediante la asignación cuidadosa de los recursos.

- Prevenir los bloqueo es más difícil que en los sistemas monoprocesador.
- En sistemas distribuidos nunca se evitan los bloqueos. El algoritmo del banquero necesita saber de antemano la proporción de cada recurso que necesitará cada proceso.

2.2.5.1. Detección distribuida de bloqueos.

- Cuando se detecta un bloqueo en un SO convencional, se resuelve eliminando uno o más procesos.
- Cuando se detecta un bloqueo en un sistema basado en transacciones atómicas, se resuelve abortando una o más transacciones. En este caso, se restaura el estado que hubiera antes de iniciar la transacción.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

Detección centralizada de bloqueos

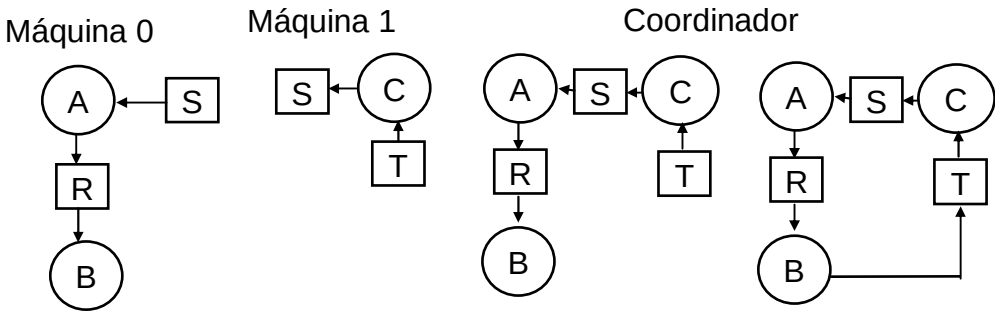
- Un coordinador central mantiene la gráfica de recursos de todo el sistema (la unión de todas las gráficas de cada máquina).
- Cuando el coordinador detecta un ciclo, elimina un proceso para romper el bloqueo.

¿Cómo construir esta gráfica?, tres opciones:

1. siempre que se añade o elimina un arco a la gráfica de recursos, se envía un mensaje al coordinador para actualizar su información.
2. cada proceso puede enviar de manera periódica una lista con los arcos añadidos o eliminados. Necesita menor número de mensajes.
3. el coordinador puede pedir la información cuando la necesite.

Ninguno de estos métodos funciona bien:

Bloqueos Falsos!!!



- Posible solución: algoritmo de *Lamport* para disponer de un tiempo global.
- Cuando el coordinador reciba un mensaje que le haga sospechar de un bloqueo envía al resto de las máquinas del sistema: he recibido un mensaje con tiempo T del mensaje que conduce a bloqueo. Si tiene un mensaje para mí con marca menor, envíemela de inmediato.
- Elimina el bloqueo pero necesita un tiempo global y es caro.

Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

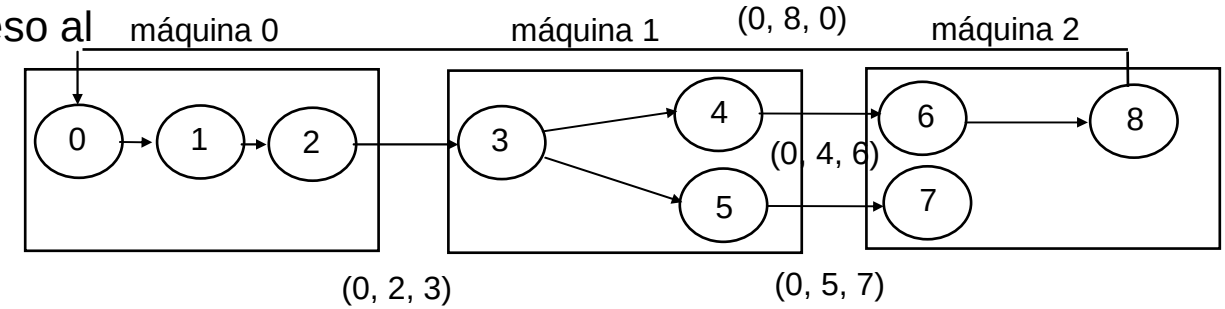
Detección distribuida de bloqueos

Algoritmo de Candy-Misra-Haas

- Los procesos pueden solicitar varios recursos (cerraduras) al mismo tiempo.
- Se utiliza cuando un proceso se bloquea porque espera cierto recurso ⇒ se genera un mensaje **exploración**, que se envía a los procesos que mantienen los recursos necesarios.
- El mensaje consta de tres números: el proceso recién bloqueado, el proceso que envía el mensaje y el proceso al que se envía.
- Al llegar el mensaje, el receptor verifica si él espera procesos. En este caso, el mensaje actualiza el 2º campo con su núm. de proceso y el 3º con el núm. al que espera.
- Si el mensaje regresa al emisor original, existe un ciclo.

Varias formas de romper el bloqueo:

- el proceso que inicia la exploración se suicida. Peligroso!! si varios procesos ejecutan el algoritmo. Demasiadas eliminaciones.
- que cada proceso añada al mensaje su identidad, el emisor conoce todo el ciclo y eliminar el de número mayor.



Tema 2.2.- Sincronización en Sistemas Operativos Distribuidos.

2.2.5.2. Prevención distribuida de bloqueos.

- Consiste en diseñar un sistema en el que los bloqueos sean imposibles.

1. Permitir a los procesos que sólo conserven un recurso al tiempo
2. Exigir a los procesos que soliciten los recursos todos los recursos desde un principio y si necesitan uno nuevo deben liberarlos todos.
3. Ordenar los recursos y que los procesos los adquieran en orden creciente.

- En sistemas distribuidos con tiempo global y transacciones atómicas:
  - Algoritmos que se basan en asociar a cada transacción una marca de tiempo global en el momento de su inicio.
  - Cuando un proceso está a punto de bloquearse porque espera un recurso utilizado por otro, verifica cuál tiene marca mayor.

Dos posibles resoluciones:

- Método **espera-muerte**: Si la transacción solicitante es más vieja, se le permite la espera. Si la transacción solicitante es más joven se aborta.
- Método **herida-espera**: Si la transacción solicitante es más vieja tiene derecho de prioridad y la transacción del joven queda eliminada. Si la transacción solicitante es más joven se mantiene a la espera.