Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.1. ¿En qué consiste la memoria compartida distribuida?

4.2. Modelos de Consistencia.

- **4.2.1. Consistencia estricta.**
- **4.2.2. Consistencia secuencial.**
- **4.2.3. Consistencia casual.**
- **4.2.4. Consistencia PRAM.**
- **4.2.5. Consistencia débil.**
- **4.2.6. Consistencia de liberación.**
- **4.2.7. Consistencia de entrada.**

4.3. Memoria compartida con base en páginas.

- **4.3.1. Diseño Clásico.**
- **4.3.2. Réplica.**
- **4.3.3. Granularidad.**
- **4.3.4. Obtención de la consistencia secuencial.**
- **4.3.5. Búsqueda del propietario.**
- **4.3.6. Búsqueda de las copias.**
- **4.3.7. Reemplazo de página.**
- **4.3.8. Sincronización.**

4.4. Memoria compartida distribuida con variables compartidas.

- **4.4.1. Munin.**
- **4.4.2. Midway.**

Bibliografía:

- Andrew S. Tanenbaum: “Sistemas Operativos Distribuidos”; tema 6, Prentice-Hall, 1996.
- Coulouris, George: “Sistemas Distribuidos: conceptos y diseño”; tema 16, Addison-Wesley, 2001

Tema 4: Memoria Compartida en Sistemas Distribuidos.

- Existen dos tipos de sistemas con varios procesadores:
  - Multiprocesadores. Dos o más CPUs comparten la MP común.
  - Multicomputadoras. Cada CPU tiene su memoria particular.
- Implicaciones software y hardware:
  - los multiprocesadores son difíciles de construir (los basados en bus pueden convertirse en un cuello de botella y los basados en conmutadores son caros, lentos, complejos y difíciles de mantener). Son fáciles de programar (comunicación y sincronización resuelto con semáforos o monitores).
  - Las multicomputadoras son fáciles de construir (conectar en red un conjunto de máquinas no tiene ningún problema). Son difíciles de programar, la comunicación se centra en la transferencia de mensajes y esto trae problemas (flujo de control, la pérdida de mensajes, el uso de buffer y el bloqueo). Las RPC no consigue solucionar todos los problemas.
- ¿Se pueden ofrecer sistemas fáciles de construir y fáciles de programar?
  - Es necesario definir algún tipo de abstracción que nos permita compartir datos entre procesos que se encuentran en computadores que no comparten su memoria física.
  - Los procesos accederán a una única memoria compartida, pero de hecho la memoria física está distribuida.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.1. ¿En qué consiste la memoria compartida?

- Al principio se consideró que los programas sin memoria Compartida debían ejecutarse en diferentes espacios de direcciones. En 1986, Li propuso un esquema diferente: **Memoria Compartida Distribuida.**
- **DMS**: una colección de máquinas conectadas con LAN compartiendo un solo espacio de direcciones virtuales con páginas. Cuando un proceso referencia una página no residente, se produce una señal y el SO busca la página y la asocia al proceso. En vez de obtener la página de disco, la obtiene de otro procesador de la red.
- Este sistema es fácil de construir y fácil de programar pero exhibe un desempeño pobre, las páginas andan de un lado a otro de la red.
- En los últimos años, existe una fuerte investigación que intenta minimizar el tráfico de la red y reducir la latencia entre el momento de una solicitud a memoria y el momento en el que se satisface ésta.
- Una solución: no compartir todo el espacio de direcciones. Sólo las variables o estructuras de datos que se necesitan en más de un proceso. Esta estrategia no sólo reduce la cantidad de datos compartidos, sino que, se dispone de información sobre los datos compartidos disponibles.
- Una posible mejora sería repetir estas variables en varias máquinas. Las lecturas se hacen de manera local y las escrituras mediante un protocolo de actualización de varias copias.
- Se podría incluso compartir objetos, esto es, no sólo los datos sin también los procedimientos (métodos).

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2. Modelos de consistencia.

- Si una copia de cada página => si está demasiado compartida => cuello de botella
- Si varias copias de una página => mantener copias consistentes que se encuentran en máquinas diferentes.
- Podemos relajar el significado de la consistencia!!! El significado preciso de la consistencia y su relajación sin hacer insoportable la programación es un tema fundamental en DSM.
- Un modelo de consistencia: especifica las garantías que un sistema DMS realiza sobre los valores que los procesos lee desde los objetos, ya que:
  - en realidad acceden a un réplica de cada objeto y
  - múltiples procesos pueden actualizar los objetos.
- es un contrato entre el software y la memoria. Si el software obedece ciertas reglas, la memoria promete trabajar de forma correcta de acuerdo al contrato establecido.
- Los contratos con restricciones menores no trabajan bien pero aquellos que imponen demasiadas reglas hacen imposible la programación normal.
  - Ejemplo: el proceso 1 espera encontrar algunas de estas combinaciones (ar=0, br=0; ar=1, br=0; ar=1, br=1). Se cumple siempre ar >= br.
  - Pero es posible actualizar a y b en otro orden => ar=0, br=1 es posible.

Proceso1	**Proceso2**

br := b;
ar :=a;
if (ar>=br) then
  print("bien");
	a:= a + 1;
	b := b + 1;

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2.1. Consistencia estricta.

- Es el modelo más estricto: *Cualquier lectura a una localidad de memoria x devuelve el valor guardado por la operación de escritura más reciente en x*.
- Requiere un tiempo global absoluto. Esto será difícil porque necesitamos de un respaldo del hardware que garantice esta consistencia estricta.

P1: W(x)1	P1: W(x)1
P2: R(x)1	P2: R(x)0 R(x)1
Memoria consistente estricta	Memoria sin consistencia estricta

P1: W(x)1	P1: W(x)1
P2: R(x)1 R(x)1	P2: R(x)0 R(x)1
Memoria consistente secuencial	Memoria consistencia secuencial

4.2.2. Consistencia secuencial.

- El modelo de consistencia estricta es casi imposible de implantar. No se deben hacer hipótesis sobre las velocidades relativas de los procesos ni el intercalado que sufran sus instrucciones.
- La consistencia secuencial tiene un problema de eficiencia: tiene que garantizar que se respeta el orden de las escrituras.
- La consistencia secuencial (definida por Lamport 1979): *El resultado de cualquier ejecución es el mismo que si las operaciones de todos los procesos fueran ejecutadas en algún orden secuencial, y las operaciones de cada proceso individual aparecen en esta secuencia en el orden especificado por el programa*.
- Esto significa que: un proceso ve las escrituras de todos los procesos pero sólo sus propias lecturas. Los resultados no son deterministas.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2.3. Consistencia causal.

- Representa un debilitamiento mayor: *Las escrituras potencialmente relacionadas de forma causal son vistas por todos los procesos en el mismo orden. Las escrituras concurrentes pueden ser vistas en orden diferente en máquinas diferentes.*
- Como ejemplo, supongamos que la W(x)2 depende potencialmente W(x)1, ya que 2 puede ser un resultado de un cálculo en el que interviene el valor leído por R(x)1.

P1: W(x)1	P1: W(x)1
P2: R(x)1 W(x)2	P2: W(x)2
P3: R(x)2 R(x)1	P3: R(x)2 R(x)1
P4: R(x)1 R(x)2	P4: R(x)1 R(x)2
Incorrecto si hay dependencia	Correcto si no hay dependencia

4.2.4. Consistencia PRAM.

- El siguiente paso en la relajación es eliminar el que las relaciones de forma causal tengan que mantenerse en el mismo orden: *Las escrituras realizadas por un proceso son recibidas por los otros procesos en el orden en que son realizadas, pero las escrituras de procesos diferentes pueden verse en un orden diferente por procesos diferentes.*
- En este modelo todas las escrituras generadas por procesos diferentes son concurrentes. La consistencia PRAM conduce a resultados poco intuitivos:

Proceso1	**Proceso2**

a=1;
if (b==0)
kill (P2);
	b := 1;
	if (a==0)
	kill (P1);

- Resultados posibles: se elimina P1, se elimina P2 o no se elimina ninguno.
- Con PRAM ambos se pueden eliminar!!! Si P1 lee *b* antes de ver la asignación de *b* en P2 y si P2 lee *a* antes de la asignación de *a* en P1.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2.5. Consistencia débil.

- La consistencia PRAM y de procesador sigue siendo restrictiva: No todas las aplicaciones requieren todas las escrituras y menos en orden.
- Por ejemplo las que ocurren dentro de una sección crítica porque ningún otro proceso puede acceder a los datos accedidos bajo exclusión mutua. Es redundante que un sistema DSM propague las actualizaciones de los datos antes de que el proceso deje la sección crítica.
- Ahora necesitamos de una variable de sincronización para sincronizar la memoria. Cuando termina una sincronización, todas las escrituras se propagan hacia fuera y todas las escrituras de otras máquinas se traen hacia esa máquina.

La consistencia débil tiene tres propiedades:

1. Los accesos a las variables de sincronización son secuencialmente consistentes.
2. No se permite realizar un acceso a una variable de sincronización hasta que las escrituras anteriores hayan terminado en todas partes.
3. No se permite realizar un acceso a los datos (lectura o escritura) hasta realizar todos los accesos anteriores a las variables de sincronización.

- Una lectura ha sido realizada cuando ninguna escritura posterior afecta el valor devuelto. Una escritura ha sido realizada en el momento en que todas las lecturas posteriores devuelven el valor escrito por la escritura. Un sincronización ha sido realizada cuando todas las variables compartidas han sido actualizadas.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2.6. Consistencia de liberación.

- La consistencia débil tiene el problema de que la memoria no sabe si el acceso a la variable de sincronización es porque el proceso ha terminado la escritura o está a punto de iniciar su lectura. En ambos casos, debe terminar todas las escrituras locales y recoger las escrituras de las demás máquinas.
- La consistencia de liberación proporciona dos tipos de operaciones de sincronización. Los accesos de adquisición indican a la memoria que está a punto de entrar en la sección crítica. Los de liberación dice que acaba de salir de una sección crítica.
- El contrato entre la memoria y el software consisten en: si el software realiza una adquisición, la memoria asegura que sus copias locales de las variables protegidas sean actualizadas con los valores remotos; si realiza una liberación, las variables protegidas modificadas se propagan hacia las demás máquinas.

Una memoria distribuida compartida tiene consistencia de liberación si cumple:

1. Antes de realizar un acceso ordinario a una variable compartida, deben terminar con éxito todas las adquisiciones anteriores del proceso en cuestión.
2. Antes de permitir la realización de la liberación, deben terminar las lecturas y escrituras anteriores del proceso.
3. Los accesos de adquisición y liberación deben ser consistentes con el procesador (no se pide la consistencia secuencial).

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.2.7. Consistencia de entrada.

- Cada variable compartida debe asociarse a un objeto de sincronización.
- Reduce el costo asociado a la adquisición y liberación puesto que sólo hay que sincronizar unas cuantas variables compartidas.
- Permite una mayor concurrencia, se incrementa el paralelismo. Pero supone un costo adicional y una complejidad mayor, la programación es más compleja y propensa a errores.

4.2.8 Resumen de modelos de consistencia

Modelos de consistencia que No utilizan operaciones de sincronización	
Estricta	• Ordenamiento absoluto con respecto al tiempo de todo lo relacionado con el acceso a memoria. Todas las modificaciones son visibles de modo inmediato. Es imposible su implantación
Secuencial	• Todos los procesos ven todos los accesos compartidos en el mismo orden. Es factible y de amplio uso pero tiene un desempeño pobre.
Causal	• Todos los procesos ven los accesos compartidos relacionados causalmente en el mismo orden.
PRAM	• Todos los procesos ven las escrituras de cada procesador en el orden en que fueron ejecutadas. Las escrituras de procesadores diferentes podrían no ser vistas en el mismo orden.

Modelos de consistencia que Utilizan operaciones de sincronización	
Débil	Los datos compartidos (dentro de una sección crítica) sólo pueden considerarse como consistentes después de realizar una sincronización
De liberación	Los datos compartidos son consistentes al salir de la sección crítica. Dos operaciones de sincronización
De entrada	Cada variable compartida tiene asociada una sincronización

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3. Memoria compartida con base en páginas.

- Esquema de memoria compartida distribuida clásica. No existe apoyo del hardware, se trata de un sistema NORMA: sin acceso a memoria remota.
- Las estaciones de trabajo sólo pueden hacer referencia a su memoria local. Se debe añadir software para cuando un procesador haga referencia a una página remota, esta página sea traída.

4.3.1. Diseño Clásico.

- El espacio de direcciones se separa en trozos, que están dispersos en todos los procesadores del sistema. Cuando se hace referencia a una dirección que no es local, se produce una señal, y el software DMS trae ese trozo que contiene la dirección => reinicia la instrucción suspendida.

4.3.2. Réplica.

- Una mejora es duplicar aquellos trozos de solo lectura (códigos, constantes…)
- Se podría duplicar todos los trozos => si se modifica, hay que evitar la inconsistencia de la memoria.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3.3 Granularidad

- Un aspecto importante de diseño es el tamaño del trozo de memoria que se trae desde una máquina remota: una palabra, un bloque (unas cuantas palabras), una página, un segmento (varias páginas).
- Una elección obvia es traer toda la página. Hace más fácil su integración con un sistema de memoria virtual. Cuando ocurre un fallo de página se trae del disco o de otro espacio de memoria.
- Otra opción, es traer una región de 2, 4 u 8 páginas. Se reduce el número de transferencias y la cantidad transferida no tarda mucho más tiempo. Esto es útil, puesto que la mayoría de los programas exhiben localidad de referencia.
- Pero, tenemos el problema de estar compartiendo variables falsamente. Si en una misma página aparecen dos variables compartidas no relacionadas y dos máquinas hacen uso frecuente de una de ellas, la página viajará constantemente entre las dos máquinas.
- Los compiladores inteligentes pueden colocar estas variables en espacios distintos de memoria. Esto es complicado y, a veces, imposible: dos procesadores hacen uso de diferentes casillas de un vector.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3.4. Obtención de la consistencia secuencial

- Si se admite duplicar páginas de lectura-escritura tenemos un problema de consistencia!!!
- Se pueden adoptar dos métodos: actualización o invalidación.

1. Actualización: se crea una copia secreta de la página que se va a modificar, cuando se modifica la página se compara con la copia para ver cuál palabra ha sido modificada => se transmite por la red su dirección y su valor. Las CPUs que reciben el paquete verifican si tienen la página y la actualizan.

- Problemas: trabajo enorme; además no es infalible.
- Si se actualizan varias copias al tiempo, se pueden realizar las actualizaciones en distinto orden, no existe consistencia secuencial!!!
- Un proceso puede realizar miles de escrituras consecutivas en las misma página => muy caro!!

2. Invalidación: Se elimina la dirección de la caché, debe buscar de nuevo esa dirección.

- Cuando un procesador Pw intenta escribir sobre un página p sobre la que no tiene acceso o sólo tiene acceso de lectura, se genera un fallo de página. Son posibles seis casos y en todos ellos se garantiza que sólo existe una copia de la página.
- Antes de realizar una escritura, el protocolo garantiza que sólo existe una copia de la página. Esta página se lleva al espacio de direcciones que está apunto de realizar la escritura.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3.5. Búsqueda del propietario.

¿Cómo localizar al propietario de la página?:

1. Realizar una transmisión y solicitar la respuesta del propietario => usa un ancho de banda considerable, sólo uno de los procesadores es el propietario.
2. Emplear un controlador de páginas.
   a) Un proceso es designado como controlador de páginas. Registra quién posee las páginas. Se necesitan cuatro mensajes (2 controlador, 2 propietario)
   b) Mejora: el controlador envía la solicitud al propietario, y éste contesta al proceso emisor (se ahorra un mensaje).
3. Carga excesiva del controlador => Varios controladores de páginas. Problema: encontrar el controlador correcto. Utilizar los bits de menor orden del número de página como índice en una tabla de controladores. El controlador también lleva los cambios de propiedad. Cuando un proceso desea escribir en una página, el controlador lo registra como nuevo propietario.
4. Otro algoritmo: cada proceso lleva un registro del propietario probable, si ha cambiado, debe volver a trasmitirse. Después de n cambios se puede volver a trasmitir la posición del nuevo propietario.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3.6. Búsqueda de las copias.

¿Cómo localizar las copias para invalidarlas?

- Transmitir un mensaje con el número de página y solicitar a los procesadores que la contengan que la invaliden. Funciona si los mensajes nunca se pueden perder.
- Un controlador de páginas mantiene una lista con los procesadores que la contienen. El antiguo propietario, el nuevo o el controlador envía un mensaje a cada procesador que contiene la página.

4.3.7. Reemplazo de página

- Hay que seleccionar de las páginas válidas cuál se debe sacar de la memoria. Se empleará alguna aproximación al algoritmo LRU
- Un buen candidato sería, una página duplicada pues existe otra copia de la página. Así no tenemos que guardarla. Pero si se lleva el registro de las copias por un controlador debe ser informado de esta decisión.
- La segunda elección es una página duplicada de un proceso saliente. Se debe transferir la propiedad a una de las copias e informar en caso de existir un controlador.
- Si no existen páginas duplicadas => la página válida de uso menos reciente. Dos posibilidades para deshacernos de ella: a) escribirla en disco b) mandarla a otro procesador.
- Problema: tráfico generado en la red cuando los procesos de máquinas diferentes comparten de manera activa una página para escritura. Se puede reducir el tráfico obligando un determinado tiempo de permanencia.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

4.3.8. Sincronización

- Los procesos necesitan con frecuencia sincronizar sus acciones. La exclusión mutua es un ejemplo de sincronización. Se suele utilizar la instrucción TSL: si la variable es 0 ningún proceso está en la sección crítica, y 1 si está.
- La variable estará contenida en una página. Si un proceso A está en la sección crítica y otro proceso B quiere entrar, debe comprobar el valor de la variable => se lleva la página. Cuando A sale, intenta escribir en la variable => fallo de página y la solicita a B. Después B la vuelve a obtener para entrar. Peor desempeño pero aceptable.
- Problema: Otros procesos intentan entrar en la sección crítica.
- La ejecución de TSL siempre implica una escritura => siempre hay que traer la página. Si varios procesos ejecutan TSL cada pocos nanosegundos => tráfico intolerable!!!
- Solución: Mecanismo adicional => Controlador de Sincronización al que se pide mediante mensajes entrar/salir S.C.

4.4. Memoria compartida distribuida con variables compartidas.

- DSM basada en páginas: considera un espacio lineal de direcciones y permite que las páginas emigren dinámicamente según la demanda.
- Los procesos tienen acceso a toda la memoria y este acceso es transparente.
- Un método más estructurado consisten en compartir variables y estructuras de datos. Ahora en vez de paginación, se mantiene un BD distribuida. El uso de variables tiene la ventaja de no compartir falsamente.
- ¿Las variables compartidas deben duplicarse? ¿de manera parcial o total?

Tema 4: Memoria Compartida en Sistemas Distribuidos..

4.4.1 Munin.

- Es un sistema DSM que se basa en objetos del software, y que puede colocarlos en páginas distintas para que el hardware pueda detectar el acceso a los objetos compartidos.
- Objetivo: Realizar pequeñas variaciones en los programas multiprocesadores para que se puedan ejecutar eficientemente en multicomputadoras con DSM.
- Anotan las variables compartidas con la palabra reservada *shared*. El compilador la reconoce y las coloca en páginas independientes, las de gran tamaño ocuparán varias páginas. El programador puede especificar la colocación de variables del mismo tipo en la misma página.
- Para acceder a la variables se utilizan instrucciones de lectura y escritura, no se emplean métodos de protección. Si se intenta usar una variable compartida no presente, ocurre un fallo de página y Munin toma el control.
- La sincronización para la exclusión mutua estará relacionada con el modelo de consistencia de memoria que se realice.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Consistencia de liberación

- Mientras un proceso está activo dentro de una sección crítica, el sistema no garantiza la consistencia de las variables compartidas, pero cuando sale, las variables modificadas son actualizadas en todas las máquinas.

Munin distingue tres clases de variables:

- Variables ordinarias: no se comparten y sólo pueden ser leídas o escritas por el proceso que las creó.
- Variables de datos compartidos: son visibles para varios procesadores y parecen secuencialmente consistentes siempre que todos los procesos las utilicen sólo en las secciones críticas.
- Variables de sincronización: son especiales y sólo se puede acceder a ellas por medio de procedimientos del sistema. Son las que permiten el buen funcionamiento de la DSM.

Protocolos múltiples.

- Munin permite otras técnicas que mejoran el desempeño: Permitir al programador que realice anotaciones en las declaraciones de las variables compartidas, clasificándolas como: Exclusiva para lectura, migratoria, de escritura compartida y convencional.

1. Exclusiva para lectura: Son la más sencillas. Cuando una referencia a una variable exclusiva para lectura provoca un fallo de página, Munin la busca en el propietario en el directorio de variables y le solicita copia de la página.

- No tienen ningún problema de consistencia porque no se modifican.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Protocolos múltiples.

2. Migratoria: utilizan un protocolo de adquisición/liberación. Están protegidas por variables de sincronización. Estas emigran de una máquina a otra conforme se hace uso de ellas pero NO se duplican.
3. De escritura compartida: el programador indica que es seguro la escritura por dos o más procesos al tiempo. Ejemplo: un proceso modifica las casillas pares y otro las impares de un vector.
   a. Las páginas con variables de escritura compartida se marcan como exclusivas para lectura.
   b. Cuando sucede una escritura, el controlador de fallos de página crea una copia, la marca como sucia, y activa el hardware de memoria para permitir escrituras posteriores.
   c. Al hacer la liberación, Munin compara cada página sucia con su gemelo y envía las diferencias a los procesos que las necesiten.
      - El receptor verifica si NO ha modificado la página, se aceptan las modificaciones recibidas. Si ha sido modifica de manera local, se comparan la copia local, el gemelo y la página recibida.
      - Si las palabras local y recibida han sido modificadas, se señala un error, si no existen conflictos se reemplaza la página local.
4. Convencional. Sólo se permite una copia de estas páginas para escritura y se desplaza entre procesos según la demanda.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Protocolos múltiples.

- con consistencia secuencial pura: Cada almacenamiento requiere una transferencia de página. Gran tráfico de red.
- con consistencia de liberación: La primera asignación provoca una copia de la página. En la segunda barrera se calculan las diferencias, se actualizan los valores y se envían a los demás procesos. El tráfico sólo en las barreras.

Proceso1	Proceso2

/* Espera a proceso 2*/
wait_at_barrier(b);
for (i=0; i<n; i+=2)
a[i] = a[i] + f(i);
/*Espera a que termine el proceso 2*/
wait_at_barrier(b);
	/* Espera a proceso 1*/
	wait_at_barrier(b);
	for (i=1; i<n; i+=2)
	a[i] = a[i] + g(i);
	/*Espera a que termine el proceso 2*/
	wait_at_barrier(b);

Directorios

- Munin utiliza un directorio de variables compartidas, una entrada por cada variable. En esta entrada está la categoría de la variable, si existe copia y quién es el probable propietario.
- Una variable compartida para escritura no tiene un propietario, una variable convencional, el propietario es el último que la adquirió, una variable migratoria, el propietario es quién la posee en ese momento.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Se lleva un registro con el algoritmo:

1. Al iniciar un proceso, el raíz posee todas las variables compartidas.
2. Un proceso P1 hace una referencia posterior => ocurre un fallo, genera un mensaje para el proceso raíz.
3. La raíz proporciona la página y anota que la tiene P1.
4. Si P2 solicita la página, la raíz le indica que la tiene P1 y P2 se la solicita.
5. Si P2 desea escribir o la página es migratoria, P2 se convierte en el nuevo propietario y P1 lo registra.

- Para las copias también se utiliza el directorio. Las actualizaciones se propagan en la liberación. Cada proceso tiene registrado algunos de los procesos que tienen copia (los que se la han solicitado) que a su vez pueden tener nota de copias de otros procesos.
- Para reducir el costo de enviar páginas que ya no interesan, se utiliza un cronómetro. Si un proceso no hace referencia a ella en cierto lapso y recibe una actualización, elimina la página. La siguiente vez, que recibe una actualización, le indica al emisor que ya no la posee.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Sincronización

- Munin mantiene un segundo directorio con las variables de sincronización. Las cerraduras actúan de manera centralizada pero están distribuidas para evitar el tráfico excesivo.
- Cuando una máquina solicita una cerradura, verifica si la posee.
- Si la tiene y está libre, se otorga.
- Si no es local, se localiza mediante el directorio, que tiene un registro del probable poseedor.
- Si no está libre, el solicitante se pone en cola. Cuando se libere la cerradura, el propietario la transfiere al siguiente de la lista.

4.4.2 Minway

- Es un sistema DSM cuyo objetivo es permitir que los programas multiprocesador y los nuevos se ejecuten de manera eficiente en las multicomputadoras.
- Los programas Minway utilizan el paquete de hilos C de Mach. Un hilo puede bifurcarse en más hilos, todos los hilos comparten el mismo espacio lineal de direcciones, que contiene datos compartidos y privados.
- Minway: mantiene las variables compartidas consistentes de manera eficiente.

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Consistencia de entrada.

- Todo acceso a una estructura compartida se hace dentro de una sección crítica protegida con una variable de sincronización.
- Cada variable está asociada con la cerradura de esa sección crítica mediante una llamada a procedimiento, lock, donde se indica si es exclusiva o no.
- Si las variables sólo serán leídas => cerradura no exclusiva, permite varios procesos al mismo tiempo en la sección crítica.
- El sistema de ejecución Minway adquiere la cerradura cuando se llama a lock, y actualiza las variables compartidas asociadas a la cerradura.
- A diferencia de la consistencia de liberación la actualización se realiza a la entrada y no cuando se libera la cerradura.
- Para que funcione la consistencia de entrada:
  - Las variables compartidas deben declararse como shared.
  - Cada variable compartida debe estar asociada con una variable de sincronización.
  - Sólo se puede acceder a las variables compartidas dentro de las sección crítica**.**

Tema 4: Memoria Compartida en Sistemas Distribuidos.

Implantación.

- Para obtener una cerradura exclusiva, necesitamos localizar al propietario, el último que la adquirió.
- Cada proceso lleva el registro del probable propietario como Munin y se sigue la cadena de propietarios sucesivos.
- Si el propietario no la está usando se transfiere la propiedad.
- Cuando se adquiere la cerradura, el proceso que la adquiere debe actualizar todas sus variables compartidas (sólo las que han sido modificadas desde la última vez que las adquirió).
- Por tanto, se debe guardar nota de los cambios que se produzcan mediante una marca de tiempo. Se debe utilizar el algoritmo de Lamport.
- La consistencia de entrada es más eficiente porque la comunicación sólo ocurre cuando un proceso realiza una adquisición.
- Tiene las desventaja de ofrecer un interfaz con el programador más compleja y propensa a fallos.