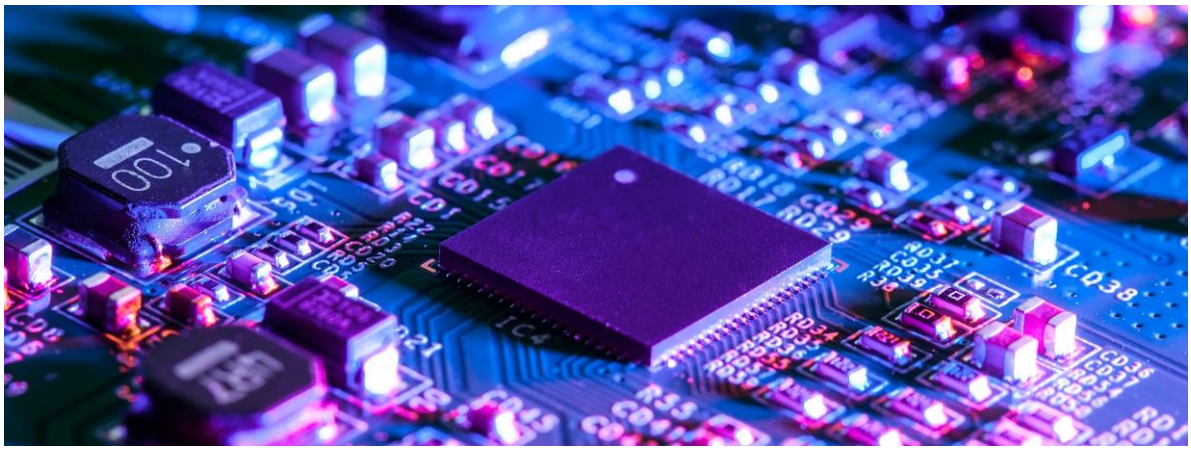
Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Coherencia de cachés

Contenido

Presentación



En este recurso veremos **cómo solucionar el problema de la coherencia de caché** cuando hay múltiples procesadores, tanto en sistemas mono procesador con varios núcleos de procesamiento, como en sistemas con varios procesadores distribuidos.

Analizaremos cuáles son los **problemas de la jerarquía de caché** en sistemas multiprocesadores y cuáles son los **métodos** para evitar errores de caché, que se clasifican en:

- Protocolo snooping (también llamado protocolo snoopy, o fisgoneo, en español):
  - Snooping para *write through.*
  - Snooping para write back.
- Coherencia basada en directorios:
  - Directorios full-map
  - Directorios limitados.
  - Directorios encadenados.

Objetivos

Los objetivos que se pretenden alcanzar con este recurso son los siguientes:

- Analizar los **problemas** de ordenadores **multi-procesador** y el problema de la **memoria caché.**
- Conocer los diferentes métodos que permiten asegurar la **coherencia de caché** en sistemas de memoria compartida.

Acceso a memoria caché en escritura

Ya hemos visto que la caché contiene una **copia** de algunos datos contenidos en la memoria. Nunca datos propios. Un acierto de caché se considera cuando el procesador intenta acceder a un dato, dato que siempre está en memoria, pero que a veces, **este dato está replicado en memoria caché.**

Y aquí es donde está la **ventaja de la caché**, si este dato se encuentra en caché, es la caché la que responde al procesador simulando que ya se ha producido el acceso al dato en memoria, con una gran ventaja, el tiempo de espera que el procesador ha tenido que emplear para recibir el dato solicitado es **muy inferior al tiempo** que emplearía si el dato no se encontrara en caché.

En detalle - Incoherencia de caché

Incoherencia de caché
Si los accesos son de tipo lectura, todo es perfecto pero, ¿qué pasa cuando un acceso a un dato es de tipo escritura y la caché tiene una réplica del mismo? El acceso, otra vez más se para en la caché, por lo que la **memoria nunca se entera** de que ha habido una **modificación del dato**. Consecuencia, el dato almacenado en caché y el dato almacenado en memoria no son los mismos. Esto se conoce como incoherencia de caché.

La **incoherencia de caché** tiene dos posibles **soluciones**, en función de la política de escritura que empleemos:

Escritura inmediata: write through	Los accesos a caché en escritura, **siempre producen fallo de caché,** por lo que la petición de escritura siempre llega a la memoria principal. No se puede dar problemas de caché, porque en ningún momento el dato en memoria se queda desactualizado. La operación de escritura se realiza siempre en la **memoria principal** y en la **memoria**

	actualizados por una escritura en caché, y estos datos antes de ser eliminados de caché, son escritos en **memoria**. Los datos permanecen desactualizados solo mientras hay una copia de los mismos en caché, pero nunca se pierden datos, porque la caché es responsable de **actualizar la memoria** antes de eliminar los datos actualizados.

Ambos métodos funcionan bien, pero como siempre, **no son perfectos.**

La **escritura inmediata** es segura, pero poco eficiente, la **escritura retardada**, por el contrario, penaliza más los fallos de caché.

Poco eficiente
Todos los accesos a memoria en escritura **provocan fallo de caché**, lo que reduce a la mitad el índice de aciertos por parte de la caché. Hay técnicas que mejoran este problema, dejando la responsabilidad a la caché de actualizar el dato en memoria y que el **procesador continúe con la ejecución** de código antes de que la memoria se actualice por completo. El inconveniente es que carga de trabajo a la caché, haciéndola sustancialmente más lenta.

Penaliza más los fallos de caché.
Si hay un fallo de caché y hay que reemplazar un bloque de datos de la misma cuando el bloque tiene activada la señal de bloque modificado. Antes de proceder al reemplazo, hay que **escribir el bloque en memoria** y después proceder a la **carga del nuevo bloque**. Esto añade retrasos en la caché, cada vez que

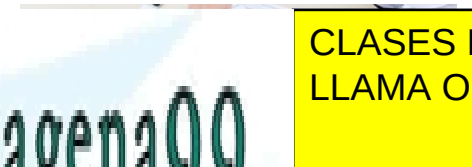
Multiprocesadores o multinúcleos

En la actualidad, la mayoría de las computadoras que se comercializan incluyen un **sistema multiprocesador o multinúcleo.** A la hora de diseñar estos sistemas hay que pensar cómo se organiza la jerarquía de memoria. Es especialmente crítico el uso de las memorias cachés y el establecimiento de las políticas de ubicación y escritura más óptimas.

Diferenciemos **multiprocesadores y multinúcleos:**

Multiprocesador
Maquina con **dos o más procesadores completos.** Normalmente, podemos hablar de una sola maquina con más de un procesador o incluso en dos máquinas diferentes.
Multinúcleo
Procesador con dos o más núcleos en un solo chip. Una máquina que tiene un solo procesador instalado. Este procesador internamente está construido por más de un núcleo funcional.

En ambas maquinas, la jerarquía de memoria es muy diferente, pero ambas tienen un problema, aunque sea a niveles diferentes.

 ¿Qué pasa con un dato que se necesite en más de un procesador?	Una variable compartida por más de un procesador, está replicada en los sistemas de memoria caché de cada procesador. Esto permite **agilizar los accesos** a esa variable, y explotar al máximo la existencia de la caché. La variable compartida está replicada en la memoria común a todos los procesadores, por tanto, **no es un dato local**. Además, está replicada en cada una de las memorias locales ya que el acceso a los datos debe ser lo más rápido posible, consiguiendo reducir latencia (tiempo de lectura) en los accesos.
	Tras una serie de accesos al dato en escritura, las tres copias

	a él en lectura.  En detalle - Máquina con dos procesadores

Máquina con dos procesadores

Supongamos una maquina con dos procesadores. Cada procesador tiene una memoria caché local y una memoria compartida a los dos procesadores para el intercambio de información. Supongamos que ambos procesadores acceden al dato compartido "i" donde guardan información de sincronización sobre la tarea que están ejecutando conjuntamente. Supongamos la **secuencia de accesos** a la variable "i" tal como sigue:

Paso	Procesador	Acción	Memoria compartida	Memoria caché procesador 1	Memoria caché procesador 2
		Valor inicial	5	¿?	¿?
1º	P.1	Lee i	5	5	¿?
2º	P.2	Lee i	5	5	5
3º	P.1	Incrementa i en 3	8	8	5
4º	p.2	Incrementa i en 1	6	8	6

Interpretación: Cuando los procesadores en los pasos 1 y 2 leen la variable compartida, todo va bien. En el paso 3, el procesador 1 accede a la variable en escritura, guardando un 8 (5+3), por lo que su copia local almacena un 8 y pospone la actualización de la copia en memoria compartida, para futuros accesos. Pero la copia local del procesador 2 no se actualiza. De aquí el problema, cuando esta guarda un 6 en la misma variable, después de calcular (5+1), en su copia local queda una 6, en memoria compartida otra 6. Pero ¿quién tiene el valor correcto de la variable "i"? Nadie. El valor correcto debería ser 9 (5+3+1), valor que no está almacenado en ninguna de las memorias.

Este problema ocurre siempre que haya una **memoria más rápida** enmascarando a una memoria mayor y compartida, pero lenta, y haya **datos compartidos** entre todos los procesos. Si analizamos la idea que acabamos de ver, significa que cuando hay una memoria caché local y hay datos compartidos, los datos

Jerarquía de caché

Hoy en día el problema de la incoherencia de caché, está **prácticamente a todos los niveles**, ya que son muy pocos lo procesadores que se comercializan que no son multinúcleos. Además, seguro que hemos escuchado hablar de que nuestro sistema tiene caché L1, L2 e incluso L3.

Ya hemos hablado de que cada nivel de caché, es una caché de la caché anterior. Por lo tanto, estas cachés, las podemos incluir como **etapas en nuestra jerarquía de memoria** o podemos hablar de que la caché está a su vez jerarquizada dentro de la jerarquía de memoria. Es ambos casos, la idea es la misma.

El problema es **cómo se organiza esta caché** en los procesadores actuales. Tomemos como ejemplo dos procesadores actuales:

Microprocesadores Intel Core i7
Esto es lo que pone Intel (Intel, 2016) en la hoja de características de uno de los microprocesadores Intel Core i7 (Legit Reviews, 2016) con respecto a su caché: • Up to 6 execution cores, each core supports two threads (Intel® Hyper-Threading Technology) for up to 12 threads. • A 32-KB instruction and 32-KB data first-level cache (L1) for each core. • A 256-KB shared instruction/data mid-level (L2) cache for each core. • Up to 15 MB last level cache (LLC): up to 2.5 MB per core instruction/data last level cache (LLC), shared among all cores. **Resumiendo**: caché L1 dedicada a cada núcleo, además dividida en datos e instrucciones. La L2, caché clásica, una caché local para cada procesador. Y por último, la caché L3, compartida para los todos los núcleos. Por lo tanto, las **políticas de coherencia de caché**, toman una gran importancia actualmente, ya que tienen utilidad desde los súper computadores con multitud de procesadores hasta el ordenador personal, con múltiples procesadores en un chip.
Procesador AMD Phenom II
Esto es lo que pone AMD de su **procesador** AMD Phenom™ II (AMD, 2016):

Esquemas básicos de coherencia de caché

La primera manera de asegurar la coherencia de caché en sistemas de memoria compartida es mediante la migración y la **replicación**:

Migración
Un dato puede moverse a una caché local y utilizarse ahí de forma transparente. Mientras un dato está migrado, ese dato, no puede estar en ninguna otra caché. La migración reduce la latencia de acceso a los datos y la demanda de ritmo de transferencia de los datos compartidos con la memoria compartida.
Replilcación
Los datos compartidos, se pueden encontrar en varias cachés simultáneamente. La replicación reduce la latencia de acceso y la disputa de la lectura de los datos compartidos.

Estos dos métodos son críticos en las prestaciones de multiprocesadores y multinúcleos por lo que los sistemas suelen emplear diferentes protocolos para asegurar la coherencia de caché.

Vamos a distinguir dos tipos de problemas:

- Los **multinúcleos**, es decir, asegurar la coherencia de caché entre procesadores cercanos, normalmente, dentro de un mismo chip. Para lo que se utilizan **algoritmos de fisgoneo (snooping)** o sistemas de memoria no cacheable.
- Los **multiprocesadores**, es decir, sistemas con varios procesados lejanos, normalmente situados en maquina diferentes y unidos por sistemas de transferencia de información tipo Ethernet Gigabit. Este tipo de arquitecturas no tiene un bus común que los comunica a todos los procesadores, por lo que las técnicas de snooping no funcionan y se sustituyen por **protocolos basados en directorios**.

En detalle - Sistemas de memoria no cacheable

Sistemas de memoria no cacheable
El sistema de memoria no cacheable es uno de los **más simples**, pero de los **menos eficientes**, por lo que su uso no está muy extendido. Se basa en **bloquear una porción de memoria compartida**, para que los datos almacenados en esta **nunca puedan estar replicados en cachés**. De manera que siempre que el

ser estos datos conocidos, el Sistema Operativo (SO) puede asignarle una posición en memoria determinada y perteneciente a la memoria no cacheable.

Es un método simple, pero que obliga a los SO a tener una **porción de memoria asignada a una tarea**, que no siempre es utilizada. Además, aumenta considerablemente la latencia en acceso a datos compartidos, aunque estos sean accedidos siempre en lectura, lo que nunca habría problemas de coherencia de caché.

Protocolo snooping (o fisgoneo)

El **protocolo snooping** se basa en el principio de **invalidación de copia de datos** en cachés, una vez que un procesador accede en modo escritura a un dato compartido.

Mientras haya operaciones de lectura no hay problema de coherencia de datos, el dato original se encuentra en la memoria principal, y cuando un procesador accede a un dato compartido, realizará una copia local en su memoria caché. El riesgo de coherencia se produce en el momento en el que haya operaciones de escritura. En ese momento debemos garantizar que cualquier procesador cuando accede a un dato compartido (por ejemplo, una variable compartida), que ha sido modificado recientemente, está utilizando el valor actual de dicha variable, y así evitar que utilice un valor antiguo, y por tanto erróneo, de dicha variable.

Este protocolo funciona en arquitecturas que tienen como sistema de comunicación un bus de datos digital para transferir la información entre cachés y memoria principal. Existe un dispositivo snooping conectado a cada memoria caché.

Se le conoce como snooping, porque está constantemente **observando el bus de comunicación** entre cachés y memoria compartida, para detectar cuándo hay datos duplicados en las distintas cachés locales, y cuándo se ejecutan operaciones de escritura sobre los datos compartidos.

Hay **dos variantes** de este algoritmo:

- Para cachés con política de escritura con escritura inmediata (write through).
- Con escritura retardada (write back).

Protocolo snooping write through

Este protocolo es **simple**, y la mejor manera de verlo, es como una **máquina de estados**. Solo hay dos posibles estados para un bloque de memoria: estado **válido e inválido.**

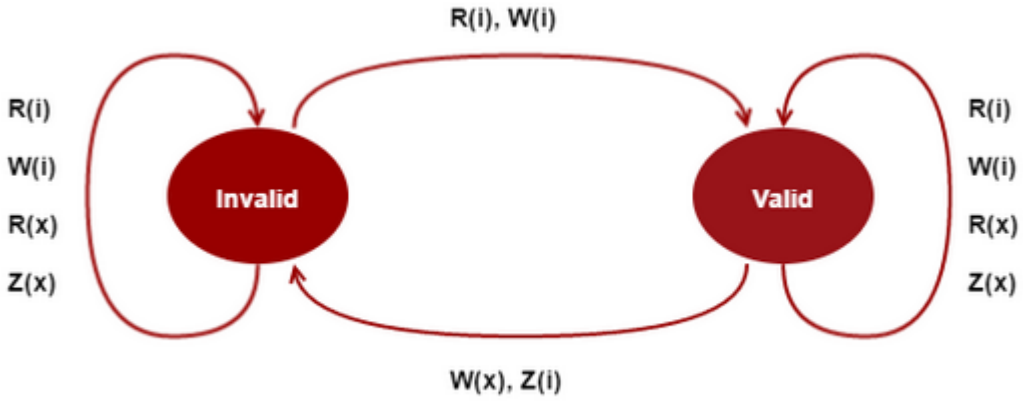
La caché, internamente almacena un **bit de validez** por línea de memoria (por bloque), que almacena este estado y el algoritmo se ejecuta internamente en la caché. Cada caché tiene por tanto su **propio algoritmo en ejecución**, de manera que es ella misma la que escanea el bus, para actualizar el estado de cada línea.

Esta es la **nomenclatura** del diagrama

- Acción: R=lectura, W=escritura, Z=reemplazo
- Procesador: i=procesador actual, x=demás procesadores.

Si analizamos el diagrama, vemos que el **principio de funcionamiento** es:

- Una línea pasará de válida a inválida cuando un procesador externo accede a un dato en escritura.
- Una línea pasara de inválida a válida cuando se accede a ella, bien en escritura o lectura.
- Para los demás casos, el estado no variará.



En detalle - Protocolo snooping write through

llamaremos X, según la secuencia:

	Acción	Memoria compartida	Memoria caché procesador 1	Memoria caché procesador 2	Memoria caché procesador 3
1º	V. Iniciales	X = 3	X=?, estado=I	X=?, estado=I	X=?, estado=I
2º	P1: Lee X	X=3	X=3, estado=V	X=?, estado=I	X=?, estado=I
3º	P2: Lee X	X=3	X=3, estado=V	X=3, estado=V	X=?, estado=I
4º	P3: Lee X	X=3	X=3, estado=V	X=3, estado=V	X=3, estado=V
5º	P1: Escribe X=9	X=9	X=9, estado=V	X=3, estado=I	X=3, estado=I
6º	P3: Escribe X=5	X=5	X=9, estado=I	X=3, estado=I	X=5, estado=V
7º	P2: Lee X	X=5	X=9, estado=I	X=5, estado=V	X=5, estado=V

Analizando el ejemplo, podemos indicar que partimos de una situación inicial en la que solo hay un dato en memoria, y que este es compartido por todos los procesadores. En las etapas 2º, 3º y 4º los tres procesadores **replican dicho valor** en sus respectivas cachés. Como todos los accesos son en lectura el valor de X queda en estado **válido en todas ellas.** En la etapa 5º, el procesador P1, accede a dicho valor en escritura.

Como es una caché write through, esta actualización se refleja en la **memoria principal**. Al mismo tiempo, las demás cachés detectan la actualización de este dato, por lo que ponen sus respectivas líneas en inválida. En la fase 6º, es el procesador P3 el que actualiza el valor de X, al igual que en el caso anterior, se **actualiza la memoria principal**, y las cachés que tienen el dato en valido lo pasan a inválido. Por último, en la fase 7º, el procesador P2 lee el contenido de la variable X. Su caché detecta que ese dato está invalidado, por lo que genera un fallo de caché, lo que provoca que se traiga de nuevo el dato desde memoria y se pasa el dato a válido. Fíjese que en este caso, al ser un acceso en lectura, **no invalida** las demás cachés.

Snooping para write back

Para cachés con política write back, al no estar la memoria actualizada, este algoritmo tiene algunos inconvenientes, por lo que **se utilizan otros métodos**. El más utilizado es el construido a partir del autómata que se ve en la imagen inferior.

Como se puede apreciar, ahora el autómata tiene **tres estados distintos**, nombrados RO=ReadOnly, RW=ReadWrite e INV=INValido. El resto de la nomenclatura coincide con el ejemplo del caso anterior.

El **principio de funcionamiento** de este algoritmo, es muy similar, solo que cuando un proceso tiene una línea de caché en modo escritura, esta estará en estado RW, obligando que todas las demás estén en inválido. Por el contrario, si nadie tiene el dato en modo escritura, entonces todas las copias pueden estar en modo de RO.

En detalle - Snooping para write back

Snooping write back

Supongamos un procesador multinúcleo con tres núcleos, con su correspondiente caché asociada a cada núcleo y una memoria compartida.

Supongamos que estos núcleos acceden de manera compartida a una dirección de memoria que llamaremos X, según la secuencia:

	Acción	Memoria compartida	Caché procesador 1	Caché procesador 2	Caché procesador 3
1º	V. Iniciales	X = 3	X=?, estado=INV	X=?, estado=INV	X=?, estado=INV
2º	P1: Lee X	X=3	X=3, estado=RO	X=?, estado=INV	X=?, estado=INV
3º	P2: Lee X	X=3	X=3, estado=V	X=3, estado=RO	X=?, estado=INV
4º	P3: Lee X	X=3	X=3, estado=RO	X=3, estado=RO	X=3, estado=RO
5º	P1: Escribe X=9	X=3	X=9, estado=RW	X=3, estado=INV	X=3, estado=INV

procesadores replican dicho valor en sus respectivas cachés. Como todos los accesos son de lectura el valor de X queda en estado RO en todas ellas. En la etapa 5º, el procesador P1, accede a dicho valor en escritura. Como es una caché write back, esta actualización no se refleja en la memoria principal, pero provoca que todas las cachés detecten el cambio de estado de este dato, por lo que ponen sus respectivas líneas en INV. En la fase 6º, es el procesador P3 el que quiere actualizar el valor de X, al tenerlo en modo INV, antes genera un fallo de caché, para que la caché que se encuentre en estado RW actualice la memoria y el procesador P3 pueda disponer del valor correcto.

Una vez la memoria se ha actualizado, el procesador P3 actualiza su copia de caché, por lo que su estado pasa a RW, pero las cachés que tienen el dato en RO o RW lo pasan a INV. Por último, en la fase 7º, el procesador P2 lee el contenido de la variable X. Su caché detecta que ese dato está INV, por lo que genera un fallo de caché, posteriormente notifica de que se desea el valor del dato, lo que provoca que la caché en estado RW actualice la memoria principal lo que provoca que se traiga de nuevo el dato desde memoria y se pasa el dato a RO. Fíjese que en este caso, al ser una acceso en lectura, las cachés en estado RW pasan a RO no ha INV.

Coherencia basada en directorios

Los protocolos de coherencia de cachés basados en directorios se utilizan en los sistemas multiprocesadores lejanos, construidos utilizando redes punto a punto o redes multietapa.

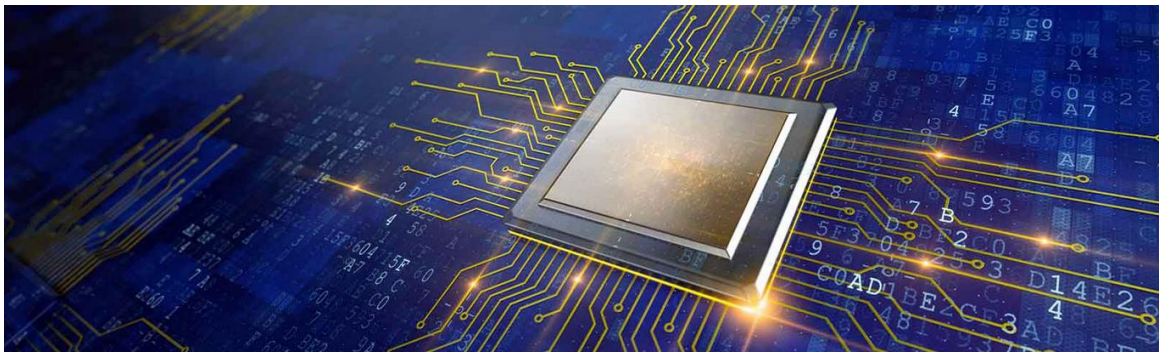
En tales arquitecturas, los procesadores están **comunicados por redes**, por lo que no existe un mecanismo adecuado de observación de todas las transacciones con la memoria y además no hay capacidad de **broadcast**, por lo que sería imposible implementar un **protocolo snooping**. Además, en estas redes, los procedimientos de comunicación a todos los nodos son **muy caros de implementar** y mantener, por lo que los comandos de consistencia deberán ser solo enviados a aquellas cachés que guarden una copia del bloque. Ello conduce a la necesidad de que exista **información almacenada** en determinados nodos sobre qué es lo que contienen otros nodos y esos almacenes son los directorios de información caché que utilizan estos protocolos y que les dan su nombre.

Se utilizan directorios caché para almacenar información sobre **dónde residen copias de los bloques de memoria.** Los diferentes protocolos de este tipo difieren en qué información contienen esos

En primer lugar, podemos distinguir **dos tipos** de esquemas:

Sistemas basados en directorio centralizado
Este contiene toda la información necesaria para mantener la consistencia. Es sencillo de implementar pero tiene el inconveniente de que el directorio central actúa como cuello de botella para los accesos.
Sistemas basados en directorios distribuidos
Cada módulo de memoria contiene un directorio separado que informa sobre dónde hay copias de los bloques de memoria que contiene, y sobre cuál es el estado de esas copias.

Los directorios de caché están formados por entradas. La memoria está dividida en bloques y el bloque es lo que se puede compartir entre procesos. Independientemente de la información almacenada en los directorios caché, en cada caché se almacena información sobre el **estado de los bloques** en ella depositados.



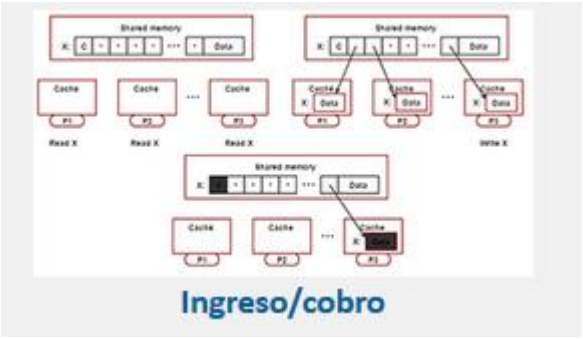
Modelos de directorios

Según cómo se organizan internamente los directorios, hay **tres grupos de protocolos** basados en directorio: los directorios full-map, los directorios limitados y los directorios encadenados.

El presupuesto de tesorería muestra los **cobros** y los **pagos** que se producirán debido a las operaciones que se han planteado en el presupuesto de operaciones y el de inversiones.

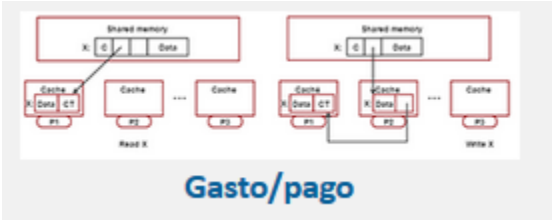
En el presupuesto de las operaciones se calculan los **gastos** e **ingresos** que se espera tendrán lugar en la compañía durante el ejercicio y para un determinado nivel de actividad. Esos gastos e ingresos se pagarán y cobrarán en un determinado momento a lo largo del periodo presupuestado o en el próximo periodo. Las entradas y salidas de efectivo del periodo al que se refiere el presupuesto es lo que muestra un presupuesto de tesorería.

A continuación, se analizarán las **diferencias** entre los conceptos de ingreso/cobro y gasto/pago.



Los directorios full-map se caracterizan porque cada entrada de directorio contiene **N bits**, siendo N el **número de procesadores del sistema**, y un bit más de sucio. Así, para cada entrada, hay N bits de presencia en procesador, uno por cada procesador del sistema dónde se indica en qué procesadores hay copias válidas del bloque.

Si el bit de sucio está activo, solo un bit de procesador estará en "presente" y quiere decir que la posición de memoria no contiene información actualizada. Obsérvese que la cantidad de memoria consumida por el directorio es **proporcional** al tamaño de la memoria y al tamaño de cada entrada.

	Puede surgir el problema de que se tengan **más copias que bits** para un bloque, y en ese caso habrá que invalidar alguna de las copias, simplemente porque no se puede reflejar su presencia en el directorio. Para proceder a ese reemplazamiento se establece una estrategia para determinar qué bloque debe ser candidato al desplazamiento.
	Los directorios encadenados mantienen información para un **número no limitado de copias por bloque** pero la mayor parte del directorio se distribuye dentro de las propias **cachés locales de los procesadores**. Las entradas de directorio se organizan en forma de lista enlazada. La primera entrada de esta lista se encuentra en memoria principal y contiene un puntero que indica en qué caché está la primera copia. Esa copia a su vez contiene un puntero que indica en qué caché está la segunda copia, y así sucesivamente.

Resumen

En este recurso, hemos visto cuáles son los problemas de la jerarquía de memoria en sistemas multiprocesador o multinúcleo.

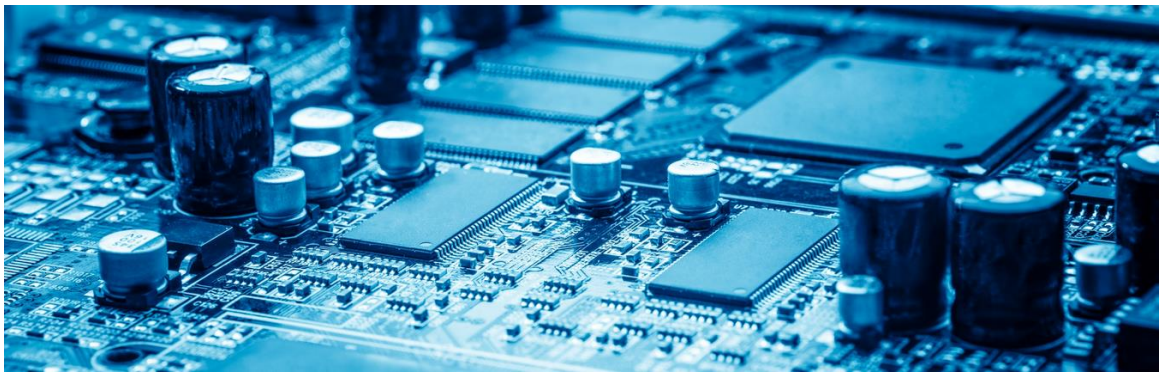
Hemos empezado exponiendo los problemas, para terminar revisando y estudiando cómo funcionan los algoritmos más utilizados para asegurar la coherencia de caché.

En los sistemas multinúcleo, normalmente se utiliza un bus digital de datos como sistema de comunicación entre las memorias cachés y la memoria principal. Para este tipo de sistemas, mantenemos la coherencia de los datos en la caché mediante el algoritmo snooping. Dependiendo de la política de escritura, tendremos dos tipos de memorias cachés:

- Protocolo **snooping write through:** para cachés write through que usan escritura directa en memoria principal.
- Protocolo **snooping write back:** para cachés write back que utilizan la escritura retardada.

Además, hemos esbozado métodos para arquitecturas multiprocesador con mapa de memoria distribuida conocidos como protocolos basados en directorios. Existen tres métodos diferentes:

- **Directorios full-map:** Necesitan mucha memoria.
- **Directorios limitados:** Están muy limitados en cuanto al número de copias de un dato.
- **Directorios encadenados:** Son los más óptimos.



Referencias bibliográficas

- AMD (2016). Procesadores AMD Phenom™ II. Acceso web: http://www.amd.com/es-xl/products/processors/desktop/phenom-ii
- Intel (2016). © Intel Corporation. Acceso web: www.intel.com. La Coruña. Editorial Netbiblo.
- Legit Reviews (2016). Intel Core i7-3960X Sandy Bridge-E Processor Review. Acceso web: http://www.legitreviews.com/intel-core-i7-3960x-sandy-bridge-e-processor-review_1773
- Marcellus D.H (1984). *Systems programming for Computers.* Prentice Hall.
- Newell, A. and Simon, H. 19, 3 (Mar. 1976), pp. 113-126. *Computer science as empirical inquiry: symbols and search. Communications* of the ACM.
- Patterson D., Hennessy J. (2011). *Estructura y Diseño de Computadores: Interfaz Hardware/Software.* Editorial Reverté. Cuarta Edición. Traducido y adaptado de: Patterson & Hennessy, Computer Organization and Design: Hardware/Software Interface. Forth Edition. 2009. Ed. Elsevier.
- Tanembaum, A.S. (1999). *Structured Computer Organization* (4th. ed.). Englewood Cliffs N.J. Prentice-Hall.