2.3 Memoria virtual

En prácticamente todos los sistemas operativos modernos se usa la técnica de memoria virtual. En esta sección se analizarán los conceptos básicos de esta técnica.

El concepto básico de la memoria virtual es en principio el mismo que se aplica a los dispositivos físicos, el sistema operativo virtualiza el hardware y entre ellos se encuentra la memoria. De esta manera los programas tratan con objetos virtuales y no físicos realmente.

Este concepto es fundamental para sostener el principio de la concurrencia en donde dos o más procesos comparten la memoria con total independencia en el uso de las direcciones físicas.

Por lo tanto el sistema operativo es el responsable de virtualizar las direcciones de los procesos de tal manera de poder administrar las direcciones físicas del hardware, proporcionando a los procesos las direcciones físicas disponibles, independientemente de las direcciones lógicas solicitadas por estos.

En principio, ya vimos la necesidad de esta solución cuando incorporamos el concepto de *traducción de direcciones* que en un principio implementamos con la suma de un valor de desplazamiento y luego, mediante tablas de páginas o tablas de segmentos. En otras palabras, en todos los métodos de administración ya vistos existe un principio de virtualización de la memoria.

Hay una diferenciación entre memoria lógica que manejan los procesos y memoria física que administra el s.o. Sin embargo se le llama **administración de la memoria virtual**, cuando a estos conceptos se le agrega el

hecho de que es posible mantener en la memoria física, solamente un número limitado de páginas (ó segmentos) de un proceso (**conjunto residente** de páginas ó segmentos), siempre que estas sean las que se usen actualmente para permitirle ejecutarse normalmente (**conjunto de trabajo**).

En otras palabras el ***conjunto de trabajo*** es un número limitado de páginas (ó segmentos) que usa actualmente un proceso ejecutable, residentes en la memoria, a las que llamamos ***conjunto residente***.

De esta forma podemos poner en la memoria mayor cantidad de procesos que compartan al procesador. El resto de las páginas (ó segmentos) de los procesos residirán en la memoria secundaria (disco). Una ventaja adicional de esto, es que podemos ejecutar procesos que tengan un tamaño mayor que la memoria física.

Cuando el procesador al ejecutar una instrucción trata de acceder a una dirección cuyo contenido no se encuentra en el conjunto residente, el propio **procesador** detecta esta situación en base a un ***bit de residencia*** (bandera que indica si la página ó segmento se encuentra en la memoria ó no) del descriptor de página o descriptor de segmento (entrada de la tabla de página o tabla de segmento, que administra a cada página o segmento). Esto se conoce como *fallo de página o falla de segmento*.

El mismo procesador genera una *excepción* que permite ejecutar la rutina de *fallo de página* o f*allo de segmento*, que básicamente consiste en cargar la página o segmento faltante desde el disco a la memoria y luego re-ejecutar la instrucción que dio lugar al *fallo*. En esta oportunidad se accederá a la página o segmento en forma exitosa y el programa continuará normalmente su ejecución. Debido a esto, esta forma de trabajo se llama administración por demanda de página o demanda de segmento, ya que todas las

páginas o segmentos ingresan a la memoria por este mecanismo. El sistema operativo no sabe que páginas o segmentos necesita el proceso. Todo ingresa por demanda.

Nota: Sin embargo en la práctica esto es eficiente si lo que se intercambia entre el disco y la memoria tiene un tamaño fijo. Por lo tanto se usa memoria paginada o con segmentación paginada como se explicará oportunamente. De aquí en adelante nos referiremos a fallo de página solamente, ya que el conjunto residente se encontrará siempre en marcos de página.

La memoria en un sistema está organizada como una jerarquía de niveles de almacenamiento, entre los que se mueve la información dependiendo de la necesidad de la misma en un determinado instante.

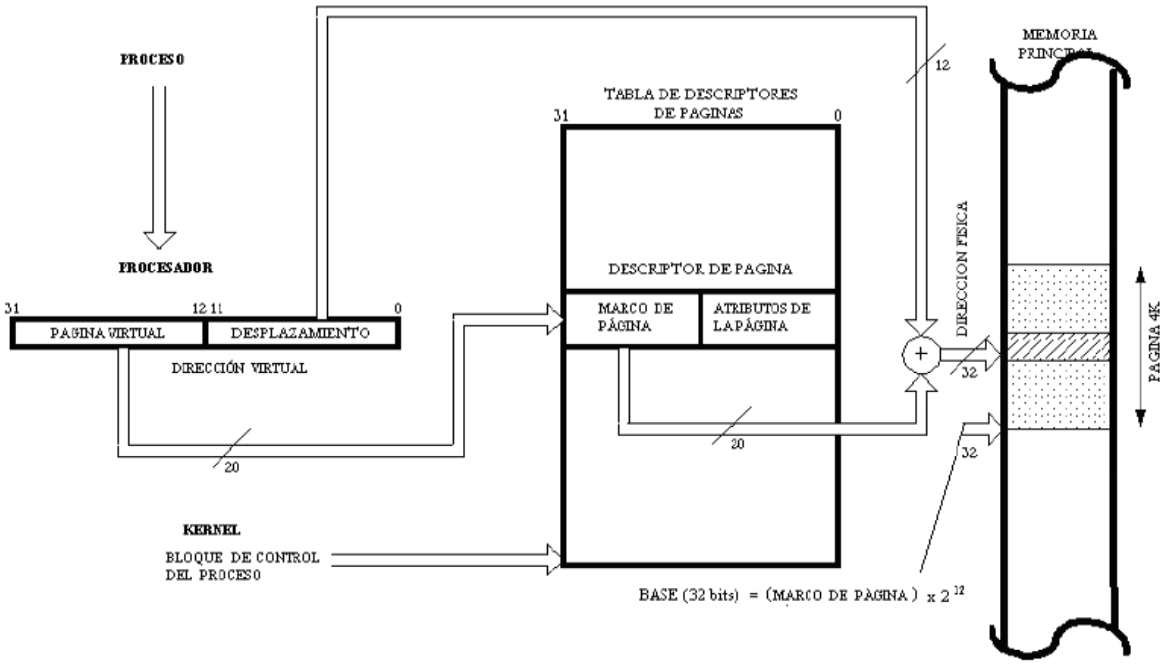
La técnica de memoria virtual se ocupa de la transferencia de información entre la memoria principal y la secundaria. La memoria secundaria está normalmente soportada en un disco (o

partición). Dado que la memoria virtual se implementa sobre un esquema de paginación, a este dispositivo se le denomina ***dispositivo de paginación***. También se usa el término dispositivo de ***swaping***. Aunque este término no es muy adecuado, ya que proviene de la técnica del intercambio (de procesos completos), pero por tradición
se usa frecuentemente y se utilizará indistintamente en esta exposición.
Es importante recordar en este punto que el buen rendimiento del sistema de memoria virtual está basado en que los procesos presentan la propiedad de ***proximidad de referencias*** (ó ***principio de localidad***) que permite mantener en un espacio de memoria reducido tanto las instrucciones como los datos que actualmente se estén utilizando. Esto también se conoce generalmente como ***cache de páginas***. Esta propiedad permite que un proceso genere muy pocos fallos aunque tenga en memoria principal sólo una
parte de su imagen de memoria (conjunto residente). Un proceso por lo tanto mantiene en memoria solamente su cache de páginas.
El objetivo del sistema de memoria virtual es intentar que la información que está usando un proceso en un determinado momento (conjunto de trabajo) esté residente en memoria principal. O sea, que el **conjunto residente del proceso contenga a su conjunto de trabajo**.
Hay que resaltar que el objetivo de la memoria virtual no es acelerar la ejecución de un programa. En algunos casos, puede hacerlo, especialmente, en situaciones donde el proceso no accede a todo su código o a todos sus datos durante su ejecución, o que el conjunto residente sea lo suficientemente grande, no siendo necesario por tanto, leerlos del disco.
Sin embargo, en otras ocasiones, puede incluso ralentizar la ejecución, debido a la sobrecarga asociada a las transferencias entre la memoria principal y la secundaria (demanda de página del disco a la memoria).
Por eso, esta técnica no es la mejor para **sistemas de tiempo real**.
Hay que recordar que la memoria virtual se construye generalmente sobre un esquema de paginación, ya sea **paginación simple** o **segmentación paginada**. Por tanto en la práctica, las unidades de información que se transfieren entre la memoria principal y la secundaria son páginas y no segmentos como también podría suponerse.
Cuando un proceso necesita acceder a una página que no está en memoria principal (a lo que se denominó fallo de página), el sistema operativo se encarga de transferirla desde la memoria secundaria. Si al intentar traer la página desde memoria secundaria, se detecta que no hay espacio en la memoria principal (no hay marcos libres o el conjunto residente excede cierto valor límite), será necesario expulsar una página de la memoria principal y transferirla a la secundaria.
Por tanto, las transferencias desde la memoria principal hacia la secundaria se realizan normalmente por expulsión.
El algoritmo para elegir qué página debe ser expulsada (el contenido de la página, de ser necesario se actualiza en el disco y se sobrescribe con el nuevo valor) se denomina ***algoritmo de reemplazo*** y se analizará más adelante. Sin embargo cabe destacar que en los sistemas operativos actuales, por una cuestión de eficiencia y velocidad, continuamente se desechan páginas no útiles del conjunto residente de
los procesos, mediante alguna estrategia comúnmente conocida como **envejecimiento de página**. Por lo tanto cuando se debe incorporar una nueva página siempre existe un marco libre.
Nota: El problema es elegir entre todos los marcos libres cual es el más adecuado, ya que algunos podrían no estar "totalmente libres" como por ejemplo aquellos que tienen grabación diferida y que deberá actualizarse previo a la sobrescritura. También por una cuestión de eficiencia es deseable que los marcos estén físicamente contiguos, ya que esto facilita la traducción de direcciones a través del cache TLB como se discutirá en temas posteriores.
Dado que se está usando la paginación para construir un esquema de memoria virtual, se puede usar indistintamente el término de dirección lógica y el de dirección virtual para referirse a las direcciones que genera un programa.
Para construir un esquema de memoria virtual soportado por hardware (sobre un procesador que ofrezca paginación), se utiliza el ***bit residente*** de la entrada de la tabla de páginas que indica si la

página es válida (el marco que indica la entrada de la tabla de página efectivamente contiene a la página solicitada por la dirección virtual).
Estarán marcadas como inválidas todas las entradas correspondientes a las páginas que no están residentes en memoria principal en ese instante.
Nota: En el caso de que se utilice el bit residente para marcar la ausencia de una página y este mismo bit también se usa para indicar que una página es realmente inválida (una página que corresponde a un hueco en el mapa), es necesario que el sistema operativo almacene información asociada a la página para distinguir entre esos dos casos.
En caso de que la página sea válida pero no residente, el sistema operativo también deberá guardar información de identificación del **bloque de la memoria secundaria** donde está almacenada la página.
De esta forma, cuando se produzca un acceso a una de estas páginas, se producirá una excepción (fallo de página) que activará al sistema operativo, que será el encargado de traerla desde la memoria secundaria.
Nota: Ya vimos que cada región del proceso tiene normalmente un soporte en memoria secundaria: archivo ejecutable en el sistema de archivos, archivo (del sistema de archivos) proyectado en memoria, o archivo de paginación (en Linux, área de swaping) del sistema.

Memoria virtual con paginación pura

Veremos a continuación detalles del modelo de paginación en que se basa la administración de memoria virtual.
Como se ha analizado previamente, los sistemas de gestión de memoria basados en asignación contigua presentan numerosas restricciones a la hora de satisfacer los requisitos que debe cumplir el gestor de memoria del sistema operativo. La paginación surge como un intento de paliar estos problemas sofisticando apreciablemente el hardware de gestión de memoria del procesador (Memory Management Unit) y aumentando considerablemente la cantidad de información de traducción que se almacena por cada proceso.
Como su nombre indica, la unidad básica de asignación, de este tipo de esquema, es la página. La página se corresponde con una zona de memoria contigua de un determinado tamaño (marco de página). Por motivos de eficiencia en la traducción, este tamaño debe ser potencia de 2 (un tamaño de página de 4 KB es un valor bastante típico).
El mapa de memoria de cada proceso se considera dividido en páginas. A su vez, la memoria principal del sistema se considera dividida en zonas del mismo tamaño que se denominan marcos de página. Un marco de página podrá contener en un determinado instante una página de memoria de un proceso.
La estructura de datos que relaciona cada página con el marco donde está almacenada es la **tabla de páginas**.
El hardware de gestión de memoria (MMU) usa esta tabla para traducir todas las direcciones que genera un programa. Esta traducción consiste en detectar a qué página del mapa corresponde una dirección lógica y acceder a la entrada de la tabla de páginas para obtener el número de marco donde está almacenada dicha página. La dirección física tendrá un desplazamiento con respecto al principio del marco igual que el que tiene la dirección lógica con respecto al principio de la página.
La siguiente figura muestra cómo es este esquema de traducción. A cada entrada de la tabla de página la llamamos descriptor de páginas ya que sería la ficha administrativa de cada página. Por lo tanto también podemos llamarle ***tabla de descriptores de página***



Cada entrada de la tabla de páginas, además del número de marco que corresponde con esa página, contiene información adicional que genéricamente llamamos **atributos de la página**. Dichos atributos
dependen del microprocesador (atributos manejados por la MMU y el sistema operativo) y del sistema
operativo (atributos manejados por software), y genéricamente son los siguientes:

- **Información de protección.** Un conjunto de bits que especifican qué tipo de accesos están permitidos. Típicamente, se controla el acceso de lectura, de ejecución y de escritura.
- **Indicación de página válida o página presente.** Un bit que especifica si esa página es válida, o sea, tiene una traducción asociada. Observe que, en el caso de que no lo sea, la información del número de marco es irrelevante. Como se analizará más adelante, este bit también se utilizará en los esquemas de memoria virtual para indicar que la página no está residente en memoria principal. El hardware consulta esta información cada vez que realiza una traducción para verificar si el acceso es correcto. En caso contrario, genera una excepción (interrupción iniciada por el propio procesador) que activa al sistema operativo.
- **Indicación de página accedida.** La MMU activa este bit indicador cuando se accede a una dirección lógica que pertenece a esa página. Indica que la página fue accedida recientemente (hace algunos ms).
- **Indicación de página modificada.** La MMU activa este bit indicador cuando se escribe en una dirección lógica que pertenece a esa página. Indica que la página contiene una modificación, y que fue diferida su actualización en el disco. Llegado el caso deberá actualizarse en el bloque que soporta a dicha página en la memoria secundaria.
- **Desactivación de cache.** Este bit indica que no debe usarse la cache de la memoria principal para acelerar el acceso a las direcciones de esta página. En sistemas con mapa común de memoria y de entrada/salida, este bit se activaría en las páginas que contienen direcciones asociadas a dispositivos de entrada/salida, ya que para estas direcciones no se debe usar la cache sino acceder directamente al dispositivo.

El sistema operativo mantiene una única tabla de páginas para sí mismo. De esta forma, todos los procesos comparten el sistema operativo. Cuando el sistema operativo está ejecutando una llamada al sistema invocada por un proceso, puede acceder directamente a su propio mapa y al del proceso.
Además de las tablas de páginas, el sistema operativo debe usar una estructura para almacenar el estado de ocupación de la memoria principal. Se trata de la tabla de marcos de página que permite conocer qué marcos están libres y cuáles están ocupados.

Por último, será necesario que el sistema operativo almacene por cada proceso su **tabla de regiones** que contenga las características de cada región, especificando qué rango de páginas pertenecen a la misma.
Observe que el esquema de paginación requiere que el sistema operativo tenga una serie de estructuras de datos de tamaño considerable para poder mantener la información requerida por el mismo. Este gasto es mucho mayor que en el esquema de asignación contigua. Evidentemente, es el precio que hay que pagar para obtener una funcionalidad mucho mayor.

Translation Lookaside Buffer (TLB)

Para hacer que un sistema de paginación sea
un volcado de la TLB a la tabla de páginas, ya que la información de los bits de página accedida o modificada se actualiza directamente en la TLB, pero no en la aplicable en la práctica es necesario que la mayoría de los accesos a memoria no impliquen una consulta a la tabla de páginas, sino que únicamente requieran el acceso a la posición solicitada. De esta forma, el rendimiento será similar al de un sistema sin paginación.
Como se comentó previamente, esto se logra mediante el uso de la TLB. Se trata de una pequeña memoria asociativa interna a la MMU que mantiene información sobre las últimas páginas accedidas. Cada entrada en la TLB es similar a la de la tabla de páginas (número de marco, protección, bit de referencia, etc.), pero incluye también el número de la página para permitir realizar una búsqueda asociativa. Existen dos alternativas en el diseño de una TLB dependiendo de si se almacenan identificadores de proceso o no.

TLB sin identificadores de proceso. La MMU accede a la TLB sólo con el número de página. Por tanto, cada vez que hay un cambio de proceso el sistema operativo debe invalidar la TLB ya que cada proceso tiene su propio mapa, y existe la superposición entre los mapas de distintos procesos.
TLB con identificadores de proceso. La MMU accede a la TLB con el número de página y un identificador de proceso. En cada entrada de la TLB, por tanto, se almacena también este identificador. La MMU obtiene el identificador de un registro del procesador (registro de tarea). El sistema operativo debe encargarse de asignarle un identificador a cada proceso y de rellenar este registro en cada cambio de proceso. De esta forma, no es necesario que el sistema operativo invalide la TLB en cada cambio de proceso, pudiendo existir en la TLB entradas correspondientes a varios procesos.
Tradicionalmente, la TLB ha sido gestionada directamente por la MMU sin intervención del sistema operativo. La MMU consulta la TLB y, si se produce un fallo debido a que la traducción de esa página no está presente, la propia MMU se encarga de buscar la traducción en la tabla de páginas e insertarla en la TLB. De hecho, la TLB es casi transparente al sistema operativo, que sólo debe encargarse en cada cambio de proceso de solicitar a la MMU su volcado y, en caso de que no use identificadores de proceso, su invalidación. Observe que es necesario realizar tabla de páginas. En realidad el bit modificado sería el más interesante ya que se trata de un proceso que no se ejecuta actualmente (pero podría ejecutarse nuevamente muy rápido y ahí si interesa el bit de accedido.)
Algunos procesadores modernos (como, por ejemplo, MIPS o Alpha) tienen un diseño alternativo en el que se traspasa parte de la gestión de la TLB al sistema operativo. A este esquema se le denomina **TLB gestionada por software**. La MMU se encarga de buscar la traducción en la TLB, pero si no la encuentra produce una excepción que activa al sistema operativo. Éste se debe encargar de buscar «a mano» en la tabla de páginas e insertar en la TLB la traducción. Observe que, con este esquema, la MMU se simplifica considerablemente, ya que no tiene que saber nada de las tablas de páginas. Además, proporciona más flexibilidad, ya que el sistema operativo puede definir las tablas de página a su conveniencia, sin ninguna restricción impuesta por el hardware. Como contrapartida, el sistema será menos eficiente, ya que parte del proceso de traducción se realiza por software.

Estructura de la TLB

Con 32 entradas residentes en la TLB se controlan 32 páginas, que suponen un espacio total de 128 KB de memoria, que, en muchas ocasiones, es suficiente para contener el área de trabajo de un proceso. Se ha comprobado experimentalmente que, para programas de propósito general, una TLB de 32 entradas proporciona "ACIERTO" en más del 97% de los accesos a la memoria.

También hay que considerar que en cada cambio de contexto, habría que limpiar la TLB, lo que puede ser barato, pero hay que considerar un costo indirecto, pues si los cambios de contexto son muy frecuentes, la tasa de aciertos se puede reducir. Al comienzo de la ejecución del proceso los aciertos son nulos o muy escasos. Recién el porcentaje es bueno cuando se ejecutan varias instrucciones.

Las 32 entradas de la TLB están organizadas en cuatro grupos de ocho entradas cada uno, que operan en paralelo. La gran velocidad en este tipo de memorias se alcanza por su método de acceso, que es por contenido (CAM: Memoria de Acceso por Contenido).

Cada entrada en una CAM se compone de una etiqueta y un dato asociado. Cuando se quiere obtener una información se suministra un valor, que se compara con los campos de etiqueta de todas las posiciones.

La comparación se hace en paralelo y a gran velocidad. En el caso de que el circuito comparador hardware detecte ACIERTO o PRESENCIA, es decir, que el valor suministrado a la CAM coincida con alguna etiqueta, la información asociada a la misma se obtiene como salida. En la TLB, se proporciona como entrada la dirección lineal y, si la contiene, el resultado es la dirección física correspondiente.

Si el comparador no encuentra una etiqueta igual a la información suministrada, la CAM no contiene la traducción que se busca y señaliza AUSENCIA o FALLO.

Como la TLB está estructurada en cuatro grupos de ocho entradas cada uno, la comparación con el campo etiqueta se hace de la siguiente forma: Con los bits 12, 13 y 14 de la dirección lineal a traducir, se selecciona una de las ocho entradas en los cuatro grupos, simultáneamente.

Dichas entradas o etiquetas constan de 21 bits: los 17 bits de más peso de la dirección lineal, un bit de VALIDEZ y tres más de atributos (D: Sucio, U: Usuario y W: Escritura). Después, mediante cuatro comparadores que trabajan en paralelo, se comparan las cuatro entradas o etiquetas seleccionadas con los bits 15 al 31 de la dirección lineal. En caso de que algún comparador detecte PRESENCIA (HIT), significa que la dirección lineal está traducida en la TLB.

Entonces la CPU procede a extraer la información ligada con la etiqueta del acierto y saca los 20 bits de más peso de la dirección física que carga en la líneas 12-31 del bus de direcciones.

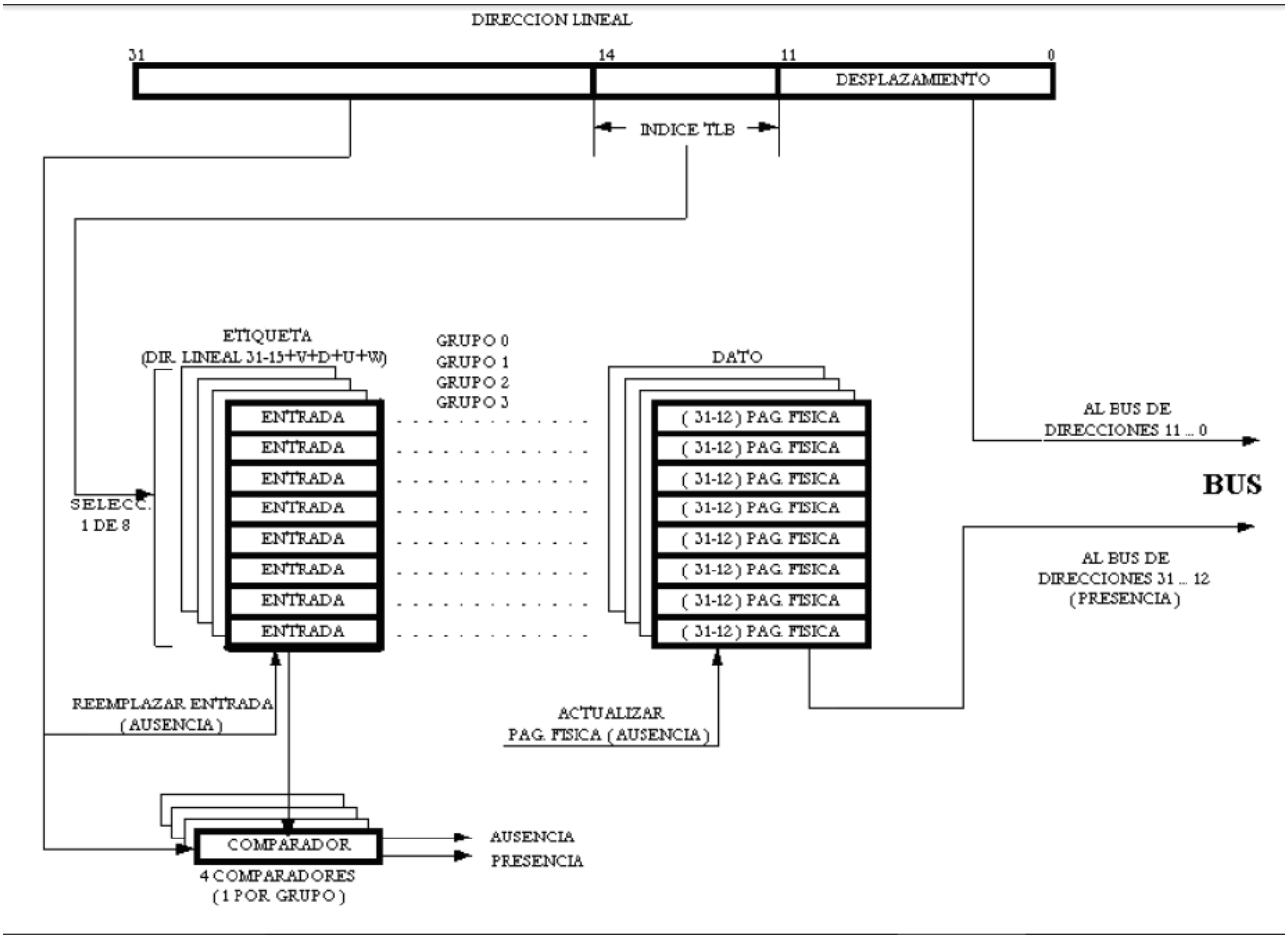
Añadiendo los 12 bits de menos peso de la dirección lineal a las líneas 0-11 del bus de direcciones, se obtiene la dirección física completa.

Si los circuitos comparadores señalan AUSENCIA (MISS), se pone en marcha el mecanismo de paginación y, tras acceder al Directorio y a una Tabla de Páginas, se obtiene los 20 bits de más peso de la dirección física, que se cargan en una de las entradas da la TLB. Después, la CPU **reintenta** el acceso por la TLB, que dará ACIERTO, procediendo a su acceso.

Cada vez que se modifican las Tablas de Páginas al cambiar de Directorio como consecuencia de modificarse el contenido del registro CR3. Habría que borrar la TLB, puesto que el i386 no hace esta operación automáticamente. El procesador asume la traducción de dirección lineal a física, sin considerar las conmutaciones de tarea.

El sistema operativo debe encargarse de:

- Inicializar las tablas de paginación
- Interpretar ciertos bits activados por la MMU
- Soportar las rutinas de fallos de páginas
- Borrar la TLB
- Re-inicializar la TLB Después de toda modificación de las tablas de páginas.
- Cargar la TLB (gestión por software)



El Pentium tiene dos TLB independientes, una para la caché de instrucciones y otra para la de datos.
La caché de datos tiene 8 entradas exclusivas para páginas de 4MB y 64 para las de 4 KB.
Las TLB son invisibles para todos los programas con excepción de los del Sistema Operativo con nivel de prioridad PL = 0.