Universidad Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

MEMORIA

MEMORIA EXTENDIDA

Índice

Presentación

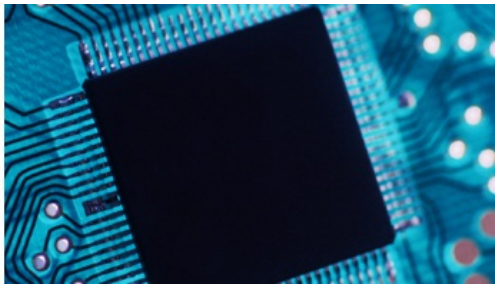
Para terminar con la memoria, solo nos queda una cosa, la **memoria extendida** o **memoria virtual**.

En este tema vamos a intentar explicar, al igual que hemos hecho con la caché, la necesidad de la misma, cómo se implementa y cuáles son sus beneficios.

No obstante, para ver la memoria extendida, nos tenemos que centrar en el sistema operativo. La memoria extendida, a diferencia de otras memorias como la RAM o la caché, que son hardware 100%, necesita del sistema operativo y de hardware. Por lo tanto, la memoria virtual está soportada a partes iguales por el microprocesador y el sistema operativo.

Los puntos sobre los que vamos a hacer hincapié en este tema son:

- La necesidad y ventajas de la memoria extendida.
- Los mecanismos para la existencia de la memoria extendida.
- Cómo se organizan los procesos: paginación o segmentación.
- Las características de la memoria virtual.
- El funcionamiento de la memoria virtual.
- El TLB (translation lookaside buffer, buffer de traducción lateral).



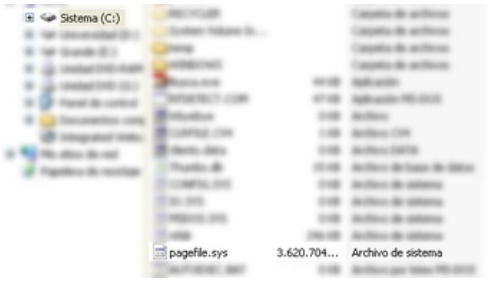
Necesidad

En la actualidad, la mayor parte de los sistemas operativos son multiusuario y multiproceso. Este hecho, trae como consecuencia, que en un sistema informático como el nuestro se estén ejecutando multitud de procesos simultáneamente. Algunos de ellos son aplicaciones que nosotros estamos utilizando, pero otros son procesos de otros usuarios o del propio sistema (sistema operativo).

La consecuencia es que la memoria RAM está compartida, por lo que a nosotros nos corresponde una porción de la misma, al igual que a otros usuarios les pasa lo mismo. Por tanto, ¿cómo podemos prevenir que nuestro proceso no altere datos de unos procesos cuando accedemos a las porciones de memoria de otros procesos?

Además, los procesos que podemos ejecutar pueden consumir muchos recursos de memoria si tenemos en cuenta las características de un ordenador. Por ejemplo, si nuestro sistema está ejecutando programas como un editor de video, un editor de imágenes, programas CAD, algún procesador de texto e incluso al mismo tiempo el navegador para consultar información por internet, podemos decir que cada uno de ellos cosume una media de 1Gb de memoria.

Por tanto, entre la memoria que retiene el propio sistema operativo (que es mucha) y lo que se llevan los procesos en ejecución, hace que la memoria RAM de nuestro ordenador sea insuficiente para darles servicio a todos.



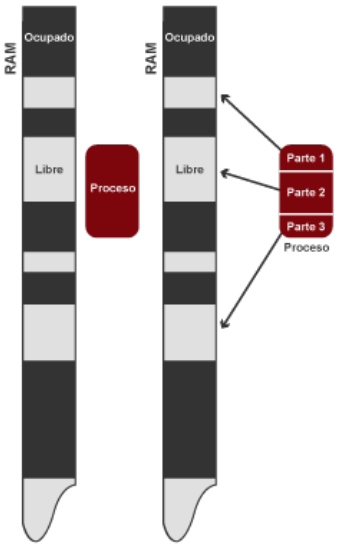
La solución la encontramos en la **memoria extendida** (o memoria virtual), que falsea los procesos en ejecución haciendo creer que estos disponen de toda la memoria que necesiten, cuando en realidad se les está dando una porción de RAM. La memoria restante, que no se encuentra en RAM, se ubica en la memoria extendida.

Pero, **¿dónde está la memoria extendida?** Depende del ordenador en concreto y del Sistema Operativo, pero generalmente, se encuentra en el **disco duro**. Para comprobarlo, si estamos ejecutando Windows, revisamos el directorio raíz del disco donde tenemos instalado el sistema operativo (normalmente C:/) y veremos que hay un archivo oculto que se llama **pagefile.sys**, de varios Gb de tamaño. Es es el fichero donde Windows guarda los datos que los procesos tienen almacenados en memoria virtual.

Mecanismos de la memoria extendida

¿Cómo es el mecanismo de memoria virtual? Existen varios sistemas, que veremos con más detalle, pero todos comparten una idea fundamental:

Cuando un proceso se inicia (comienza su ejecución) tiene que estar en memoria para que se pueda ejecutar, es decir, sus instrucciones y datos tienen que estar en memoria para que sean direccionables desde el microprocesador. Por lo tanto, hay que asignarle un hueco de memoria.



Debemos tener en cuenta que la cantidad de procesos que se están cargando y cerrando, provocan que la memoria esté ocupada a trozos, por lo que a veces, aún habiendo memoria libre suficiente, no hay un bloque lo bastante grande para dar cabida a este nuevo proceso. Esto se le conoce como **desfragmentación de la memoria**.

La solución es que el proceso se parte en trozos, y cada trozo puede ser asignado a huecos libres que, además, no tienen por qué ser contiguos. De esta manera, ya no es necesario encontrar un hueco lo suficientemente grande como para el proceso completo. Es más, incluso no habiendo espacio para todos los trozos de un proceso, no pasa nada, en memoria principal no hay por qué cargarlos todos, solo aquellos estrictamente necesarios para que el proceso comience la ejecución, asignando los demás en memoria virtual.

Debemos presta atención a que las partes de un proceso no cargadas en memoria principal, no son alcanzables desde el procesador al no tener una dirección de memoria real asignada.

Paginación/segmentación de procesos

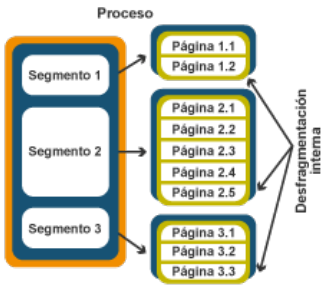
Para partir el proceso para que pueda ser cargado en memoria existen dos posibilidades, aunque ninguna perfecta:

En tamaños fijos	Llamados páginas. Es decir, tanto la memoria como los procesos los dividimos en bloques de tamaño fijo, a los que llamaremos páginas. Si analizamos bien, es justo lo que hace la caché, que divide la memoria en bloques de tamaño *conjunto de vecindad*. Con la memoria virtual, las páginas son mucho más grandes, del orden de 4kb, 16kb ó hasta 64kb. Depende mucho de la implementación.
En tamaños variables	Llamados segmentos. Es decir, no se asignan divisiones de tamaño fijo, sino de tamaño variable, que se asocia generalmente a unidades ejecutivas. Por lo tanto, un código tiene funciones, objetos, etc. Estas funciones no son del mismo tamaño. Entonces, ¿por qué hacer tamaño fijo si el código se organiza en bloques de tamaños diferentes?

Los problemas de estas dos formas de partición son los siguientes:

Segmentación interna	Las páginas provocan que un proceso pueda no tener una división exacta en páginas, sino que es probable que haya trozos de página que no estén totalmente ocupados por el proceso, por lo que puede provocar espacios de memoria asignados a una página que no estén totalmente utilizados. Se le conoce como **fragmentación interna**.
Segmentación externa	Los segmentos, por el contrario, no tienen fragmentación interna, pero no solucionamos el problema visto en el tema anterior, es decir, que existan espacios de memoria demasiado pequeños como para que puedan ser utilizados por un segmento. A esto se le conoce como **segmentación externa**.

Hoy día, en los procesadores actuales no se utiliza ningún método. Como siempre, la solución más eficaz es una mezcla de las dos, y se conoce como **memoria de segmentos paginados**.



Por un lado, la idea es aprovechar que los procesos se pueden dividir en bloques de tamaño variable, ya que así está implementado el código, y las funciones no son todas del mismo tamaño. Por otro lado, evitar la fragmentación externa, haciendo que la memoria esté paginada.

¿Cómo podemos hacer coincidir una memoria paginada con los procesos segmentados? Para ello, debemos paginar los segmentos de los procesos, tal como se ve muestra en la imagen de pantalla.

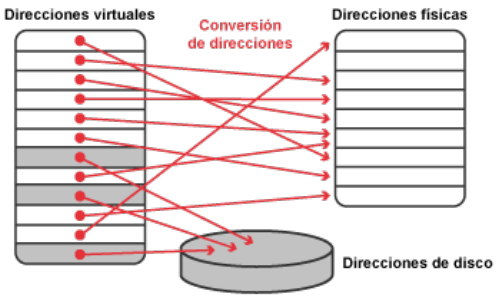
Características de la memoria virtual

Cada vez que un proceso es cargado en memoria, éste puede ser cargado en cualquier posición. Por tanto, las referencias a variables no siempre son las mismas direcciones a memoria. Por ejemplo, supongamos que el compilador reemplaza la variable **int a** por una dirección de memoria real. Pero, ¿qué dirección utilizar? Si cada vez que un proceso es cargado en memoria, éste ocupa una posición distinta, ¿cómo sabremos la dirección de esa variable?

Por otro lado, como programadores, crearemos programas (procesos) muy variados, unos amplios y otros más amplios. Pero, ¿de cuánta memoria podemos disponer si no sabemos en qué ordenador se ejecutará? A continuación, vamos a ver cómo responde a estas reguntas la memoria virtual.

Para empezar, los procesos creen que tienen disponibilidad máxima de toda la memoria que necesiten, y además, cada vez que se ejecutan, siempre *piensan* que se han situado en la dirección cero de toda la memoria. Es decir, dan por hecho que no hay problemas de memoria. Además, si continuamente *creen* que, por casualidad, a dicho proceso se le asigna siempre un bloque de memoria contigua y que empieza en la dirección cero, es muy fácil que el compilador pueda sustituir las variables simbólicas por direcciones reales de memoria. Es decir, el compilador (que es quien realiza esta tarea) sabe calcular qué posición ocuparía una variable en memoria al analizar el código de nuestro programa.

Esta apreciación es solo una configuración falsa que el proceso asume. El encargado de mantener esta falsedad sobre todos los procesos es el MMU, que es uno de los componentes hardware que se encarga de administrar la memoria virtual.



Fuente: Patterson y Hennessy (2011)

El procedimiento para conseguir esta *mentira* consiste en el empleo de direcciones virtuales por parte del proceso y la conversión de éstas a direcciones Reales por parte del MMU. En un principio, esta era la principal ventaja de la memoria virtual, pero hoy en día, consiste en la protección de la memoria de los procesos. Es decir, prevenir que un proceso solo tenga acceso a los bloques de memoria propios, protegiendo la memoria de otros procesos.

Bloque de memoria contigua y comienzo en la dirección cero

Esto lo *creen* todos los procesos que se están ejecutando en un ordenador en un momento determinado. Todas sus direcciones empiezan en la posición cero y se les asigna un bloque contiguo.

MMU

MMU son las siglas de Unidad de Manejo de Memoria.

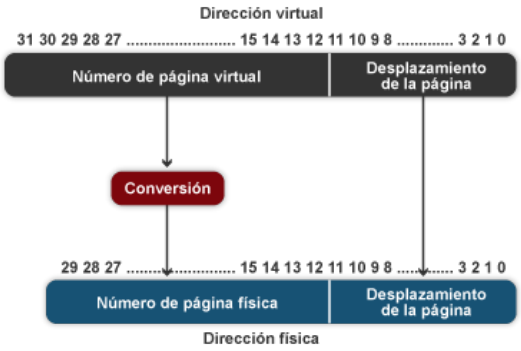
Funcionamiento de la memoria virtual

Los procesos organizan, mediante direcciones virtuales, las direcciones de memoria tanto para instrucciones como para datos. Por su parte, cuando el sistema operativo carga un proceso para dejarlo en el estado de disponible, para ser ejecutado, no todo el proceso se carga en memoria, sino solo un conjunto de páginas suficiente para que el proceso pueda empezar a funcionar.

Esto provoca, al igual que pasaba con la caché, que por algún salto en el código o, simplemente, la necesidad de utilizar una variable lejana, el proceso intente utilizar un dato o instrucción de una página que no estaba previamente cargada en memoria. A esto se le conoce como **fallo de página**. Como sucedía en la caché, se produce una carga de esa página en memoria para que luego pueda ser referenciada.

En la figura, podemos ver una instantánea de un proceso cualquiera que está en un momento dado en ejecución. Podemos observar cómo algunas páginas, de todas las que forman un proceso, están cargadas en memoria física y otras en memoria virtual (disco duro).

A pesar de que los principios de funcionamiento de la memoria virtual y la caché son parecidos, tienen algunas peculiaridades:



Fuente: Patterson y Hennessy (2011)

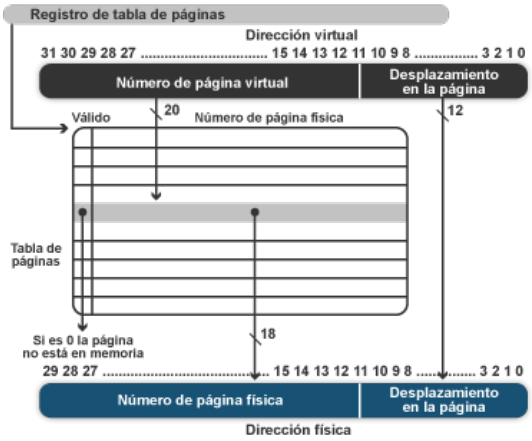
- Las direcciones de memoria entre memoria física y virtual no coinciden, por lo que tiene que haber conversión de dirección.
- Los tiempos de movimiento de datos entre memoria virtual y real son enormes, de millones de ciclos de reloj (segundos en tiempo real).
- La caché es autónoma, si la sacamos de un procesador, no ocurre nada (a excepción de la reducción de eficiencia), mientras que para la memoria virtual, el procesador y el sistema operativo tienen que estar preparados.

Emplazamiento de una página

Cuando hay un fallo de página, el proceso que ha producido el error, entra en suspensión, El sistema operativo coge el control del procesador para hacer el movimiento de página y, una vez encontrado un hueco para la nueva página, esta es movida. A continuación, se le devuelve el control al proceso suspendido para que sea reanimado.

Para situar la nueva página, y lo más importante, cómo encontrarla cuando sea necesario utilizar sus datos, utilizamos un método muy simple: mediante una tabla de conversión. Es decir, cada proceso tiene una tabla de asignación de páginas. Esta tabla tiene información que relaciona la página del proceso con la ubicación de la misma en memoria principal, además de otra información como si esa pagina esta o no en memoria o virtualizada.

Como se puede ver en la figura, una memoria virtual tiene un número de bits dependiente de la máquina (32 en nuestro ejemplo), mientras que las direcciones físicas, pueden tener otro número de bits, ya que no hay ningún problema de compatibilidad.



Fuente: Patterson y Hennessy (2011)

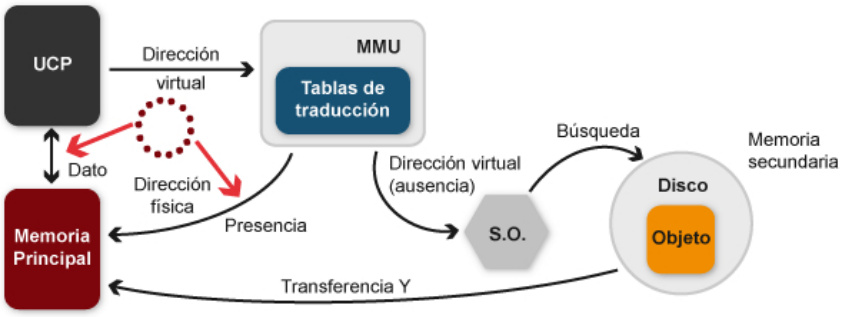
Una dirección virtual es *truncada* para separar los bits de la dirección que pertenecen a un desplazamiento de página y cuáles bits son de página. Este truncamiento viene determinado por el tamaño de página. En el ejemplo se trunca 12 bits, por lo tanto, el tamaño de pagina utilizada es 2^{12}=4Kb. La parte de dirección perteneciente a número de página se utiliza como selector de fila en la tabla de conversiones, reemplazando la de este campo, en la dirección virtual, por el dato almacenado en la tabla para obtener la dirección física.

Fijémonos que la tabla también tiene un bit asociado a cada fila, el bit de válido, que identifica qué pagina está cargada un memoria y cual virtualizada.

Fallo de página

A continuación, vamos a ver que el proceso de petición de datos por parte del procesador es un poco más complejo que antes. Por tanto, además de la caché, la RAM y los discos, ahora también tenemos la memoria virtual.

Como podemos observar en la imagen de pantalla, metemos un componente más en el esquema, el MMU y las tablas de conversión. En la figura, ¿dónde colocaríamos la memoria caché?



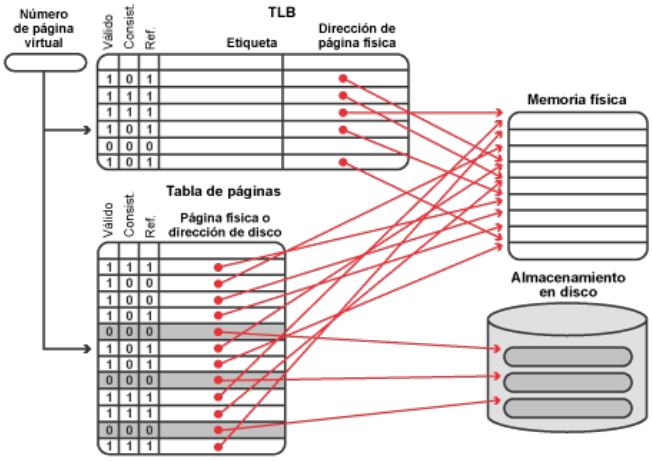
Aunque no es fácil determinar dónde colocaríamos la caché, nuestra primera idea podría consistir en cortar los dos buses: el que va desde el MMU a la memoria, y el bus de datos que va de UCP a la memoria (marcados con la flechas rojas). Dado que la caché trabaja con direcciones físicas, debe situarse después de las conversiones. Pero el problema viene cuando analizamos y vemos que las tablas de conversión residen en memoria. Eso quiere decir que para cada petición de datos por parte del microprocesador, antes hay que hacer otra petición de datos a memoria para convertir la dirección de virtual a real. Esto resulta, como mínimo, muy poco eficiente, ya que un acceso a memoria origina otro. El descenso de eficiencia es inaceptable.

La solución consiste en una caché de la tabla de traducción de tablas, la **TLB**. No obstante, antes de pasar a ver el TLB, debemos definir cómo es el método para determinar qué página se quita de memoria física cuando hay que meter una nueva página en memoria. Para ello, una vez más, utilizamos el algoritmo de reemplazo LRU; una implementación muy parecida a la que se emplea para cachés de dos vías, añadiendo un bit más a la tabla de conversión, que indica cuál de las páginas es la menos utilizada y, por tanto, la más propensa a ser reemplazada.

Algoritmo de reemplazo LRU

Debemos saber que, a pesar de existir una tabla de traducciones privada para cada proceso, el algoritmo de reemplazo funciona sobre la totalidad de las páginas almacenadas en memoria. Es decir, que al proceso **PX** necesite cargar una nueva página en RAM física, no significa que haya que eliminar una página de PX, sino que esa página eliminada puede ser de cualquier otro proceso. Este sistema balancea mejor la carga, ya que los procesos que están mucho tiempo sin ser utilizados, se van quedando con menos presencia en RAM en beneficio de procesos más *vivos*.

El TLB



Fuente: Patterson y Hennessy (2011)

Una dirección virtual tiene que ser traducida a una dirección física por medio de una consulta a unas tablas de traducción que residen en memoria. Esto es un círculo vicioso que ralentiza mucho los accesos a los datos por parte del procesador. Por tanto, la solución pasa por tener una porción de la tabla de traducción en una caché especial que reside dentro de la porción del MMU, que se implementa en la propia CPU. La CPU tiene guardadas las conversiones más habituales, con el fin de agilizar las conversiones. El comportamiento de la tabla TLB es similar al de la caché estudiada en temas anteriores.

En la imagen, es importante observar que las tablas de conversión incluyen ahora tres bits más asociados a cada fila, y que se describen a continuación.

Válido	El bit que determina si la página está en memoria física o virtualizado.
Consistencia	Se utiliza para determinar cuándo una página ha sido accedida en escritura, por lo que debe ser escrita en memoria virtual antes de ser eliminado. Es el bit utilizado para las políticas de escritura.
Referenciado	Es el bit de uso, utilizado por el algoritmo LRU de remplazo.

Caché especial

Esta caché no es la que hemos estudiado en el tema anterior.

TLB

TLB son las siglas de Translation Lookaside buffer, Buffer de Traducción Lateral.

Resumen

En referencia a la memoria virtual, faltan todavía muchos contenidos por estudiar para ampliar nuestros conocimientos, pero quedan pendientes fuera del alcance de esta materia.

Por ejemplo, una materia que no hemos abordado, y que en muy importante, consiste en la reducción de las tablas de traducción, ya que es posible que en algún momento nos hayamos puesto a hacer números.

Si existe una tabla por proceso, se utilizan 20 bits para referenciar el índice de tabla, y la tabla almacena un dato de 20 bits, que junto con los bit se estado, forman unos 32 bits. La tabla de conversión ocupa en memoria la cantidad de 2^{20}filas * 4bytes (32bits) y constituyen 4Mb de memoria ocupada por cada tabla.

Estos 4Mb, son reducidos por técnicas **Hash** de reducción, permitiendo la paganización de la tabla de páginas, etc. Estos métodos está fuera del alcance de este tema.