3ra Parte: Gestión de Memoria

2do Cuatrimestre – 2024

El apunte siempre es modificado y actualizado. Estudiar del apunte del año y cuatrimestre actual

Contenido

3. Gestión de Memoria

3.1. Introducción y Objetivos

Podemos entender la *memoria principal* como un inmenso casillero, donde cada casilla está numerada (*con una dirección*) y puede almacenar una *palabra*.

Una *palabra* es el conjunto de bits que la arquitectura de un computador puede manejar como un todo. Los tamaños de palabra más comunes son de 16, 32 ó 64 bits.

Como dijimos al hablar de procesos, para que un programa pueda ejecutarse, sus instrucciones y sus datos tendrán que estar presentes en la *memoria principal* del sistema, lo que conocemos como memoria *RAM*. Como hemos visto anteriormente, planificando el uso del procesador, mejoraremos el rendimiento general del sistema, pero esto implicará la necesidad de compartir la memoria principal entre varios procesos de forma simultánea. Por lo tanto, una buena administración de la memoria, repercutirá de forma inmediata en el comportamiento de todo el *sistema informático*.

La parte del sistema operativo que se ocupa de gestionar la memoria se le denomina *Gestor de Memoria*. Su cometido consiste en llevar la cuenta de las partes de memoria que se están utilizando y las que están libres, así como de gestionar el trasvase de información entre la memoria principal y la secundaria cuando la memoria RAM no sea suficientemente grande para acoger a todos los procesos.

Por lo tanto, el gestor de memoria deberá asignar la porción necesaria de memoria principal a cada proceso que lo necesite.

Para ejecutar una instrucción, habría que leerla desde la memoria a un registro del procesador y decodificarla (*averiguar qué significa*). A continuación, es posible que el sistema deba volver a la memoria para obtener los datos implicados en la operación. Finalmente, es común que los resultados obtenidos también haya que guardarlos en la memoria. Por todo ello, si la gestión de la memoria no es adecuada, el rendimiento general del sistema se verá inmediatamente disminuido.

Los sistemas operativos modernos son, casi siempre, *multiprogramados*. Es decir, ejecutan varios *procesos* de forma concurrente. Esto significa que la memoria debe dividirse para darles cabida.

Cuanto más eficaz sea ese reparto, más procesos podrán ejecutarse a la vez, lo que redundará en un mayor rendimiento (*no debemos olvidar que el procesador es el elemento más rápido del sistema y que el objetivo principal consiste en que siempre tenga instrucciones listas para ser ejecutadas*).

Además, la gestión de memoria deberá cumplir con las siguientes necesidades:

- ***Protección***: Debe evitarse que un proceso haga referencia a posiciones de memoria de un proceso diferente.

  Como el sistema operativo no puede anticiparse a todos los accesos que realizará un proceso durante su ejecución, es el *procesador* quien debe tener un mecanismo que intercepte los accesos no permitidos.

- ***Reubicación***: Como veremos más adelante, una forma de dar cabida a más procesos consiste en descargar a disco la totalidad o una parte de la memoria ocupada por un proceso (por ejemplo, mientras se encuentra

bloqueado). La *reubicación* consiste en que al volver a la memoria, no sea necesario que ocupe la posición original.

Para que esto sea posible las referencias que hagan los programas a direcciones de memoria deben ser *lógicas* y serán el *hardware* y el *sistema operativo* quienes colaboren para realizar la traducción.

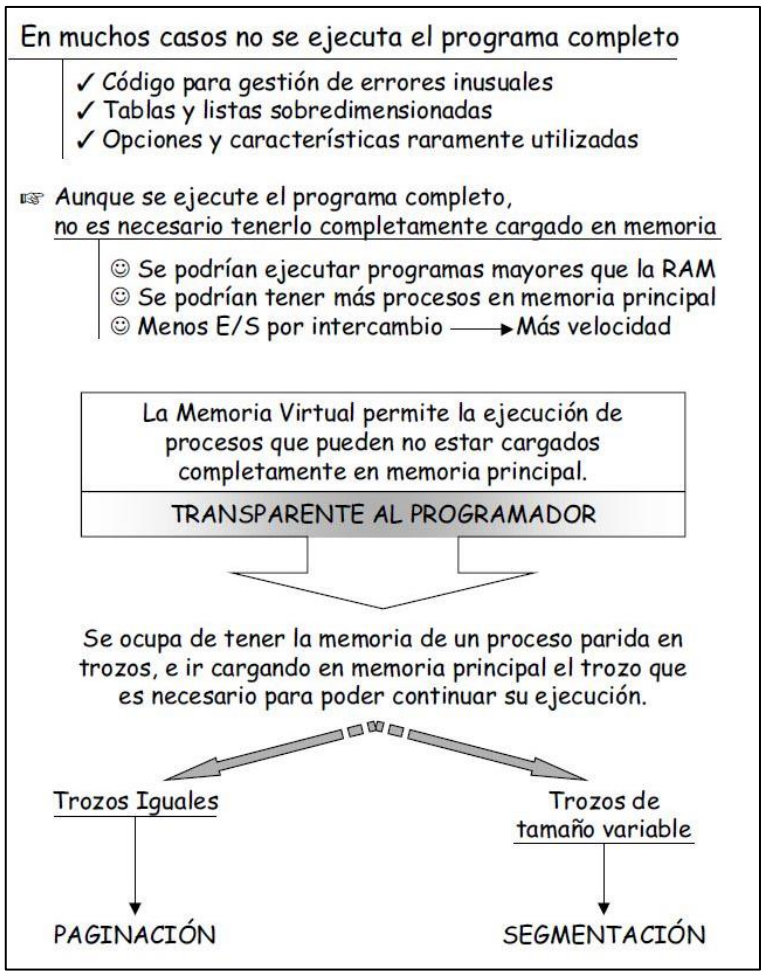
- ***Compartición***: Debe existir la posibilidad de que varios procesos compartan información a través de una zona de memoria compartida. Por ejemplo, es más eficiente que varios procesos accedan al mismo código de una *biblioteca de funciones compartida* (lo que en *Windows* conocemos como *DLL*), que mantener varias copias de ésta.

Igual puede ocurrir con datos que manejen diferentes procesos.

La gestión de memoria, cuenta con distintos mecanismos, en este a apunte analizaremos solo uno de ellos la ***paginación***;

Mecanismos

- Paginación.
- Segmentación.
- Segmentación Paginada.
- Tablas multinivel. *Modelo paginado con tabla de dos o más niveles*.



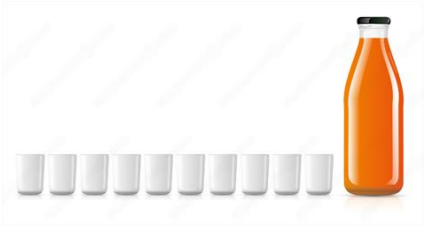
3.2. Paginación

En la gestión de memoria con la técnica de "*intercambio*", había que pasar un proceso a ejecución, era necesario traer el proceso entero de disco a memoria principal.

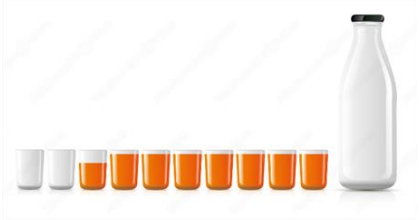
Con memoria virtual hemos dicho que no se trae todo el proceso, sino que cuando se hace referencia a una dirección de memoria virtual cuya correspondiente memoria física reside en disco, se trae el contenido de disco a RAM.

En un esquema de paginación, la memoria se divide en trozos del mismo tamaño que reciben el nombre de marcos de página (*o, en inglés, frames*). Del mismo modo, los procesos se dividen en fragmentos del mismo tamaño denominados *páginas*.

A modo de ejemplo, imagina que el proceso que queremos cargar en memoria lo representáramos con el *Jugo de frutas* contenido en la botella de la siguiente imagen. Por su parte, la memoria donde pretendemos almacenarlo estaría representada por los vasos. Cada vaso, sería el equivalente a un marco de página de la memoria.



Así, cuando carguemos el proceso en memoria, cada página (*cantidad de "Jugo de frutas" del tamaño de un vaso*) se ubicará en un marco de página diferente.



De este modo, cuando llegue un nuevo proceso, el único problema será encontrar la cantidad suficiente de marcos de página disponibles en la memoria principal, caso contario habrá *fallo de página* y se tendrá que decidir que marco liberar a partir de un *algoritmo de paginación*

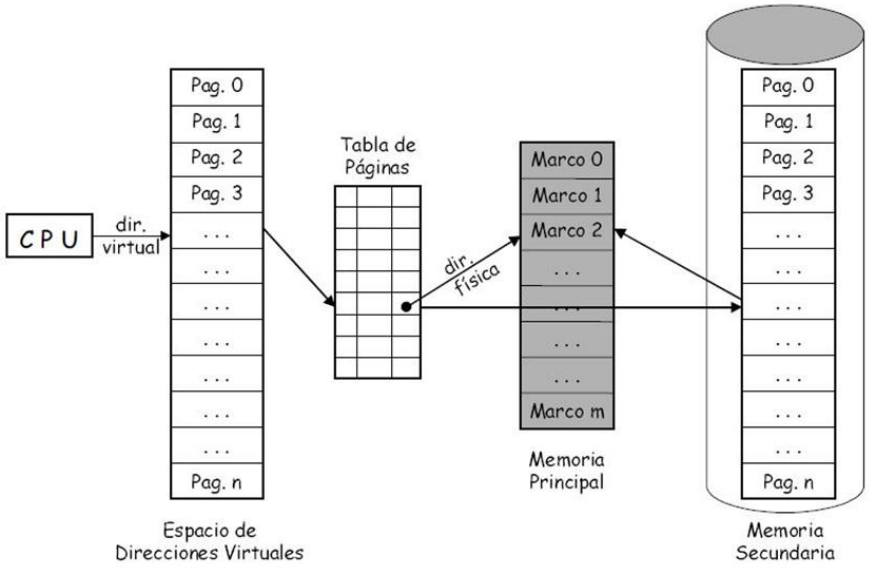
Gracias a este planteamiento, queda en evidencia la *fragmentación interna* quedará reducida al *último marco de página* asignado a cada proceso. Lo que equivaldría, en nuestro ejemplo, al último vaso. Cabe esperar que, por término medio, la mitad del último marco de página quede desocupado.

Como podrás suponer, este esquema necesita un método que traduzca las direcciones virtuales a direcciones físicas, teniendo en cuenta la ubicación real de cada marco de página. Este método se basa en la creación de una *tabla de páginas*, para cada proceso, en el momento de cargarlo en memoria. En ella se establecerá el paralelismo entre cada página y su marco de página correspondiente.

Por lo tanto, las *direcciones virtuales* constarán de un número de página y un desplazamiento dentro de ella. El número de página actuará como índice en la tabla de páginas.

Además, es frecuente que el sistema operativo mantenga una lista de marcos de página disponibles.

Las direcciones virtuales también suelen llamarse direcciones relativas y suelen asignarse durante la compilación del programa, siendo relativas al comienzo del mismo (*que será la dirección 0 de la página 0*).

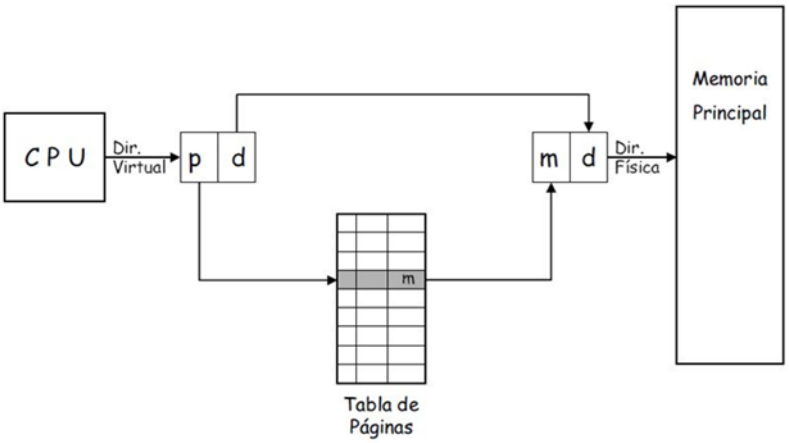


Así pues, veamos a grandes rasgos y con ayuda de la Figura, qué pasos se suceden a partir de una referencia generada por la CPU:

1. La CPU hace referencia a una dirección virtual de memoria.
2. Se calcula a qué página corresponde tal dirección.
3. Mediante la tabla de páginas se comprueba si la página se encuentra en algún marco de memoria física o en disco.
4. Si se encuentra en un marco, se traduce la dirección virtual a la dirección física que ocupa la página.
5. Si no se encuentra en un marco, hay que traer la página desde disco a un marco libre de memoria. Posteriormente habrá que traducir la dirección virtual a la dirección física, dependiendo del marco asignado a la página.
6. Por último, se realiza la operación de L/E sobre memoria principal solicitada por la CPU.

3.2.1. Traducción de direcciones virtuales en direcciones físicas

El espacio de direcciones virtuales está dividido en bloques del tamaño de una página. Por lo tanto, una dirección virtual está formada por una pagina p y un desplazamiento d dentro de la página, o sea por el par (*p,d*), mientras que el espacio de direcciones físicas o reales está dividido en m marcos del mismo tamaño de la página, por lo que las direcciones físicas estarán formadas por un marco m y un desplazamiento dentro del marco, es decir, por el par (*m,d*).



Por lo tanto, para realizar la traducción de direcciones virtuales a reales hay que construir la función de correspondencia (p,d) ⇒(m,d).

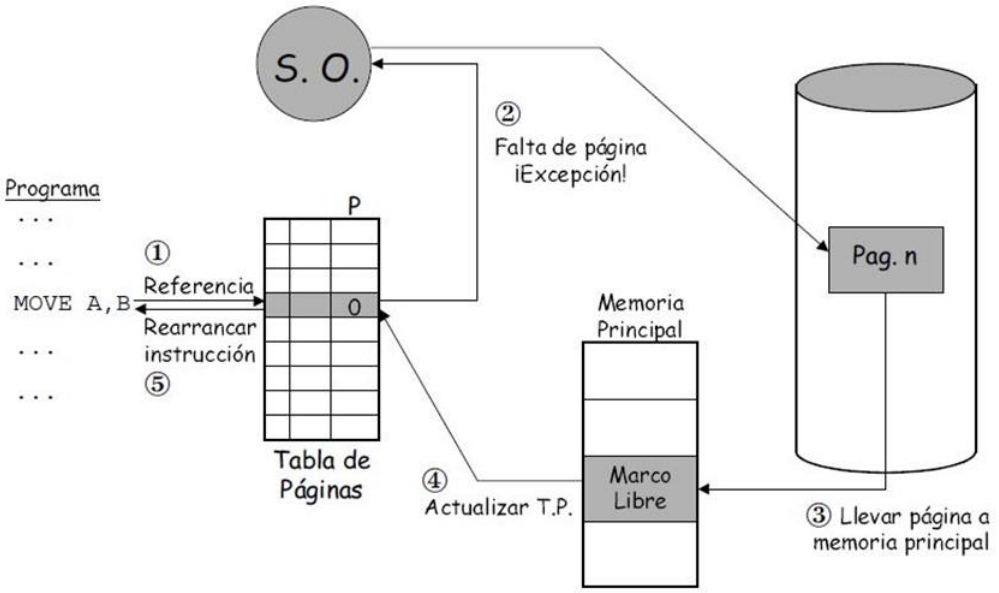
Sabemos que una página que se encuentre en memoria principal se encuentra en uno de los marcos. Puesto que el tamaño de las páginas y de los marcos es el mismo, parece obvio que para una dirección virtual (p,d) a la que le corresponde la dirección física (m,d), el desplazamiento d será el mismo, tanto dentro de la página como dentro del marco asignado. Así, únicamente será necesario realizar la traducción de página a marco (p ⇒ m), por lo que la tabla de traducción (tabla de páginas) solamente tendrá que tener tantas entradas como páginas, no tantas como direcciones virtuales.

Pues bien, a partir de una dirección virtual (p,d), se comprueba en la tabla de páginas si la página p se encuentra en alguno de los marcos de memoria. Si es así, se obtiene de la tabla el marco m que contiene la página. La dirección física es entonces (m,d).

Obsérvese que, dado que el número de páginas puede ser mayor que el número de marcos de memoria física, el número de bits de la dirección virtual puede ser mayor que el de la dirección física.

Conviene saber que, en el bus de direcciones, que se dirige desde la CPU a la memoria principal, se encuentra interceptado por el hardware que nos ayuda a realizar el proceso de traducción, o sea, la Unidad de Gestión de Memoria (*Memory Management Unit o MMU*). Ésta recibe las direcciones virtuales y con ayuda de la tabla de páginas genera las direcciones físicas correspondientes, sin necesidad de software adicional. (*No obstante, hoy día la MMU suele estar integrada dentro de la propia CPU*).

3.2.2. Falta de página



Cuando en la referencia a una dirección virtual, la página correspondiente no se encuentra en memoria principal, entonces decimos que se produce una **falta de página**. En este caso, las acciones a realizar son, como se indica en la Figura, las siguientes:

1. La CPU genera una dirección virtual.
2. La MMU consulta el bit de presencia de la tabla de páginas y se comprueba que la página de la dirección referenciada no se encuentra en memoria principal.
3. La MMU pone la CPU en espera y genera una interrupción (que suele ser Error de Bus) que es capturada por el sistema operativo.
4. Las rutinas de carga de páginas del sistema operativo se encargan de localizar la página referenciada en el disco y cargarla en un marco libre de la memoria principal.
5. Se actualiza la tabla de páginas indicando que está presente y el marco en el que ha sido cargada.
6. Termina el tratamiento de la excepción.
7. La MMU indica a la CPU que debe rearrancar la instrucción que provocó la falta de página.
8. Se continua la ejecución de la instrucción a partir del direccionamiento que produjo la falta de página. Esta vez ya no se producirá la falta de página.

Obsérvese que en el intercambio de memoria se traían y llevaban procesos enteros si había un área de memoria contigua suficientemente grande; con paginación basta con que se disponga de la memoria necesaria, aunque no sea contigua, pues el proceso puede estar repartido entre varias páginas.

En principio, parece que cuando se produce una *falta de página*, son necesarias dos transferencias entre memoria y disco: una para llevar la víctima al área de intercambio y otra para traer la página referenciada al marco que se acaba de liberar. Esto duplica el tiempo de servicio de una falta de página, incrementando, por lo tanto,

el tiempo medio de acceso a memoria. Esta sobrecarga en tiempo puede aliviarse utilizando el bit de ensuciado o bit de página modificada. En cada entrada de la tabla de páginas hay un **bit de ensuciado** que se pone a "falso" cuando se carga la página en memoria. Este bit permanece así hasta que se produzca una escritura en la página, en cuyo momento el hardware lo pone a "cierto", indicando que la página ha sido modificada desde que se trajo de disco. De este modo, si en algún momento esta página es elegida como víctima en una sustitución, se consultará el bit de ensuciado; si la página no ha sido modificada quiere decir que la copia en memoria es idéntica a la copia en disco, luego no hay motivo para escribirla de nuevo en el disco. Si por el contrario ha sido modificada, habrá que actualizarla en el área de intercambio. Con esta técnica, el tiempo de sustitución de páginas se reduce a la mitad en el caso de que la víctima no haya sido modificada.

3.2.3. Estructura de la Tabla de Páginas

Ya dijimos que la función de correspondencia entre las direcciones virtuales y las físicas o reales se realiza mediante una tabla de páginas. Esta tabla tiene tantas entradas como páginas existentes y cada entrada contiene información que, aunque varía de unos sistemas a otros, suele estar compuesta de los siguientes campos:

- **Protección**. Expresa los permisos de acceso del proceso. En caso de tener permiso para la operación de lectura/escritura que se va a realizar, se consulta el resto de los campos.
- **Bit de presencia**. Indica si la página está presente en memoria principal o no. Si se encuentra en RAM, también tienen sentido los siguientes campos.
- **Marco ocupado**. Si la página se encuentra en memoria principal, este campo expresa el marco que la contiene.
- **Modificada (bit de ensuciado).** Este campo indica si el contenido de la página ha sido modificado desde que se trajo de la memoria secundaria.
- **Referenciada**. Este campo booleano se pone a cierto cada vez que se hace referencia a cualquier dirección de la página. Lo utiliza el sistema operativo para ayudar a los algoritmos de sustitución de páginas.

En caso de que la *página referenciada no esté cargada en memoria*, sabemos que se encuentra en el *área de intercambio del disco*, pero ¿en qué lugar concreto?

La dirección concreta de cada página virtual en el disco (cilindro, pista, sector), no se indica explícitamente en ningún campo de la tabla de páginas. En su lugar, *lo que se hace es calcular la dirección.*

Esto se puede realizar fácilmente ya que el *área de intercambio* ocupa una porción contigua del disco, o sea, que no está fragmentada, bien sea por estar

- ***en una partición reservada*** para ello (caso de Unix y Linux)
- ***en un archivo*** creado en por el sistema operativo en la generación del sistema (*caso de Windows: Pagefile.sys*) y que tampoco está fragmentado.

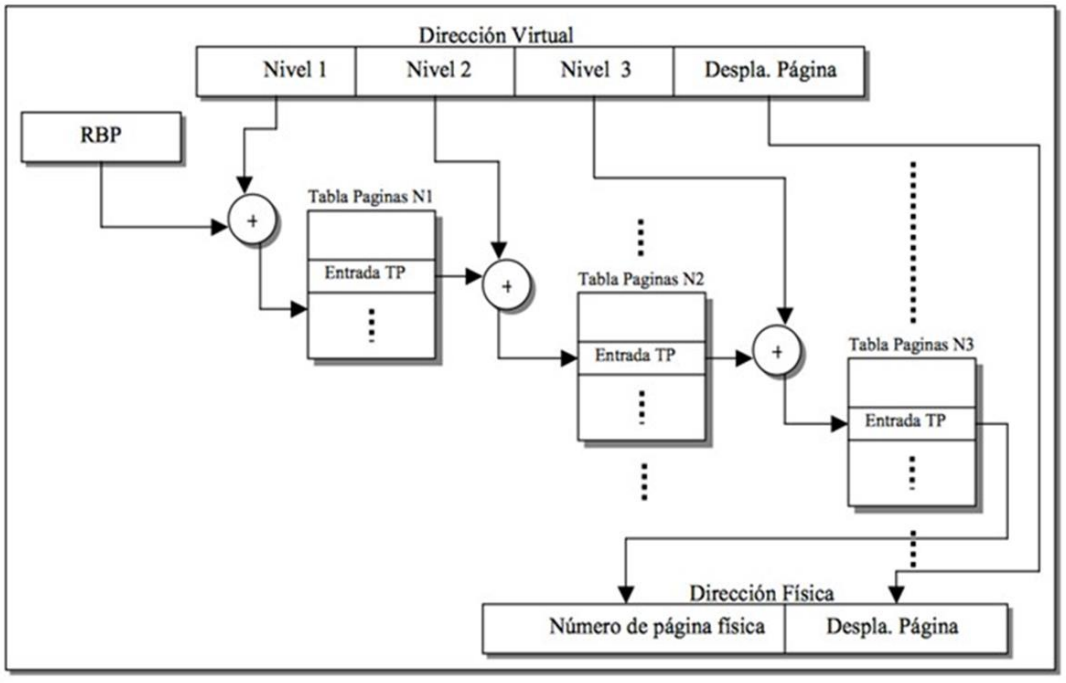
Conociendo la dirección de comienzo del área de intercambio y el tamaño de la página, el cálculo de la dirección de cada página es inmediato. De esta forma se ahorra mucho espacio en la tabla de páginas.

En los sistemas de hoy día, dado su amplio espectro de direccionamiento, las tablas de páginas pueden tener miles o millones de entradas, por lo que normalmente se tendrá cargada en memoria principal solamente una parte de una tabla de páginas (*la correspondiente al proceso en ejecución*), el resto se mantendrá en disco. El cambio de contexto tendrá que incluir por tanto la carga y actualización de la tabla de páginas del proceso que pasa a ejecución, lo que viene a incrementar el tiempo necesitado para realizar dicho cambio.

Dado el enorme rango de direccionamiento del que se dispone en las computadoras actuales, y teniendo en cuenta que los procesos suelen referenciar sólo unas pocas páginas durante su ejecución, normalmente se utilizan unas pocas de los miles o millones de entradas que contiene la tabla de páginas, con el consiguiente desperdicio del espacio de memoria utilizado por las entradas de las páginas no correspondientes al proceso.

Para evitar esto, otra posible organizar la información de control de las páginas consiste en estructurarla en varios niveles de jerarquía, en lugar de tener una única tabla muy grande, dando lugar a las llamadas **tablas multinivel**.

Algunos procesadores hacen uso de un esquema de dos o más niveles para organizar las tablas de páginas. En este esquema, hay una página de directorio de nivel 1 en la que cada elemento apunta a una tabla de páginas de nivel 2, y así sucesivamente. En la siguiente figura se muestra una TP organizada en tres niveles: N1, N2 y N3. Típicamente, la longitud máxima de una tabla de páginas se restringe al tamaño de una página.



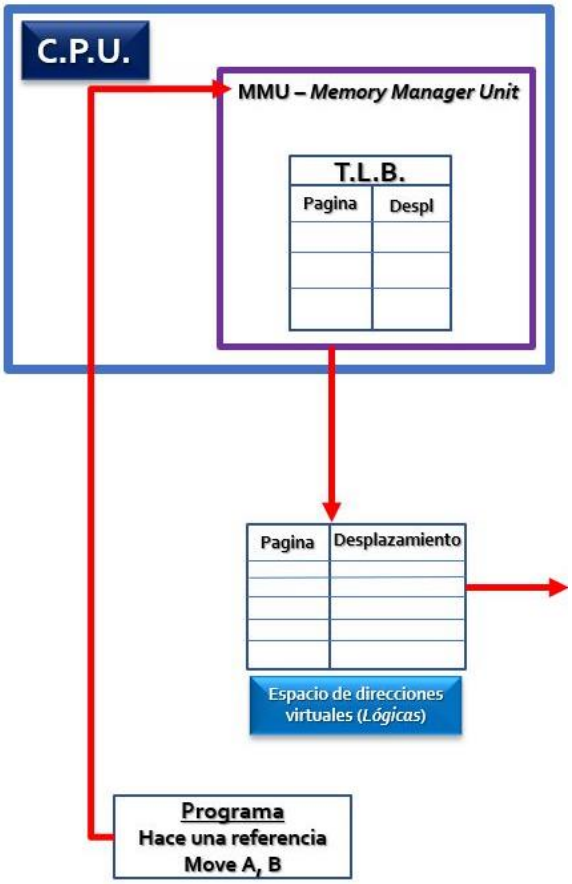
3.2.4. Soporte Hardware – TLB - MMU

Como acabamos de ver, uno de los soportes para la tabla de páginas puede ser un conjunto de registros dedicados a los que la CPU puede acceder con la misma rapidez que a los registros generales. En los cambios de contexto, el *dispatcher* tendría que actualizar estos registros como el resto de los registros generales.

Como ya vimos anteriormente, para realizar una referencia a la página n hay que acceder a la enésima entrada de la tabla, obtener el marco m y formar la dirección como ya sabemos. Pero según esto, para cada referencia de la CPU a una dirección de RAM se requieren *dos accesos a memoria*, uno para calcular la dirección real y otro para acceder a la dirección calculada y realizar la operación de lectura/escritura. Claramente, esto ralentiza los accesos a memoria en un factor de 2, lo cual puede resultar intolerable en ciertas circunstancias.

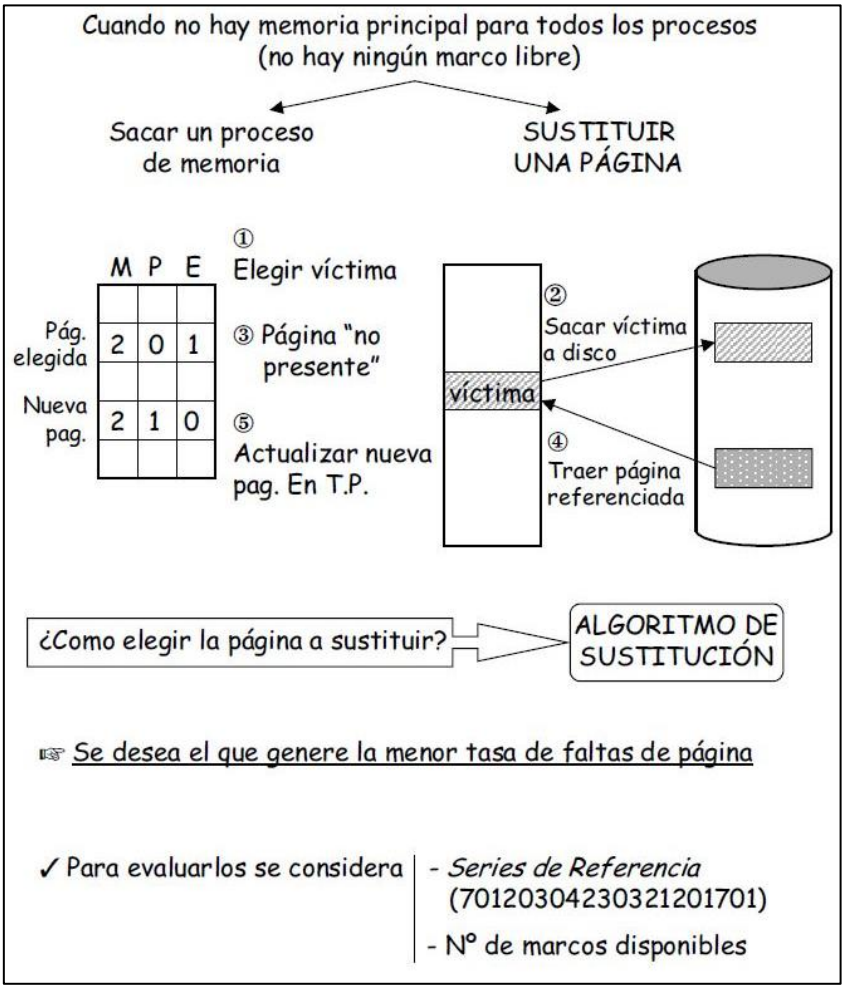
La solución habitual a este problema consiste en utilizar una pequeña memoria asociativa de acceso muy rápido conocida como **TLB** (*Translation Lookaside Buffer*) que contiene las entradas de los últimos accesos a la tabla (de 8 a 2048). De este modo se consigue un tiempo medio de acceso sólo ligeramente mayor que en los accesos directos a memoria.

Para finalizar con la ayuda que se recibe del hardware en toda la operación de traducción, solamente recordar que todo el proceso de conversión de direcciones virtuales en reales se realiza bajo la dirección no de la CPU, sino de un procesador especializado en este trabajo denominado **MMU** (*Memory Manager Unit*), que se sitúa entre la CPU y la memoria principal, recibiendo como entrada la dirección virtual que viene por el bus de direcciones y generando a su vez, como salida, una dirección real que va dirigida a la memoria. La utilización de memoria virtual es tan común hoy día, que los procesadores actuales suelen incluir la MMU en la misma pastilla de la CPU.



3.2.5. Sustitución de Páginas

Según hemos visto hasta ahora, el tratamiento de una falta de página es como el descrito en la Figura antecesora, esto es, simplemente hay que traer la página referenciada desde disco a un marco libre de la memoria principal y rearrancar la instrucción que originó la falta de página. Pero ¿qué pasa si no hay ningún marco libre?



Esta situación no es extraña; pensemos que la memoria puede estar completamente ocupada por dos motivos:

- Se utilizan muchas páginas del proceso en ejecución.
- Se ha ido incrementando el grado de multiprogramación hasta ocupar toda la memoria principal, por lo que ésta se encuentra ocupada por páginas de diversos procesos.

En este último caso, se podría optar por sacar alguno de los procesos completamente a disco, es decir, todas sus páginas; pero puede que no sea una buena idea, porque a lo mejor en la siguiente porción de tiempo hay que volver a cargar todas esas páginas. Puesto que lo único que se necesita en este momento es espacio para una página, la mejor salida para cuando se produce una falta de página suele ser la sustitución de alguna de las páginas que actualmente residen en memoria principal por la que acaba de ser referenciada. El proceso de sustitución tiene los siguientes pasos:

1. Se elige la "víctima", es decir, una página cargada en memoria principal, para llevarla a disco y dejar un marco libre. La elección se realizará dependiendo de la política de sustitución de páginas.
2. Se lleva la página elegida al área de intercambio del disco.
3. En la entrada de la tabla de páginas correspondiente a la víctima, se indica que "no está presente".
4. Se trae la página referenciada al marco que se acaba de dejar libre.
5. Se actualiza la entrada de la tabla de páginas correspondiente a la página que se acaba de traer a memoria principal, indicando que está presente y actualizando tanto el número de marco que ocupa como los campos de control que sean necesarios en cada caso.

Nos queda un problema por resolver: ¿cómo elegir la página a sustituir cuando no hay ningún marco libre? Veamos ahora las distintas políticas de sustitución de páginas.

3.2.6. Algoritmos de Sustitución de Páginas

Hay muchos algoritmos de sustitución; posiblemente cada sistema operativo tiene el suyo particular; pero en cualquier caso todos persiguen lo mismo: seleccionar páginas que causen la menor tasa posible de faltas de página.

- Primera en Entrar - Primera en Salir (FIFO)
- La Menos Recientemente Utilizada (LRU)
- Algoritmo de la Segunda Oportunidad (del reloj)

La eficacia de los algoritmos de sustitución se evalúa ejecutándolos sobre una serie concreta de referencias a memoria y contabilizando el número de faltas de página que se producen. A la serie de referencias a memoria se la denomina **serie de referencia** o cadena de referencia. Las series de referencia se pueden generar artificialmente (*mediante un generador de números aleatorios*) o capturando todas las referencias a memoria que se realizan en la ejecución de un programa dado.

3.2.7. Cuestiones de Diseño

Otro aspecto a tener en cuenta será el tamaño de los *marcos de página*:

- Con ***marcos de página pequeños***, tendremos poca *fragmentación interna* y *tablas de páginas* grandes.
- Con ***marcos de página grandes***, tendremos más *fragmentación interna* y *tablas de páginas* pequeñas.

Usando la *paginación*, se pueden seguir aplicando técnicas de *intercambio*, para mover a memoria secundaria, los procesos que se encuentran bloqueados en espera de un suceso.

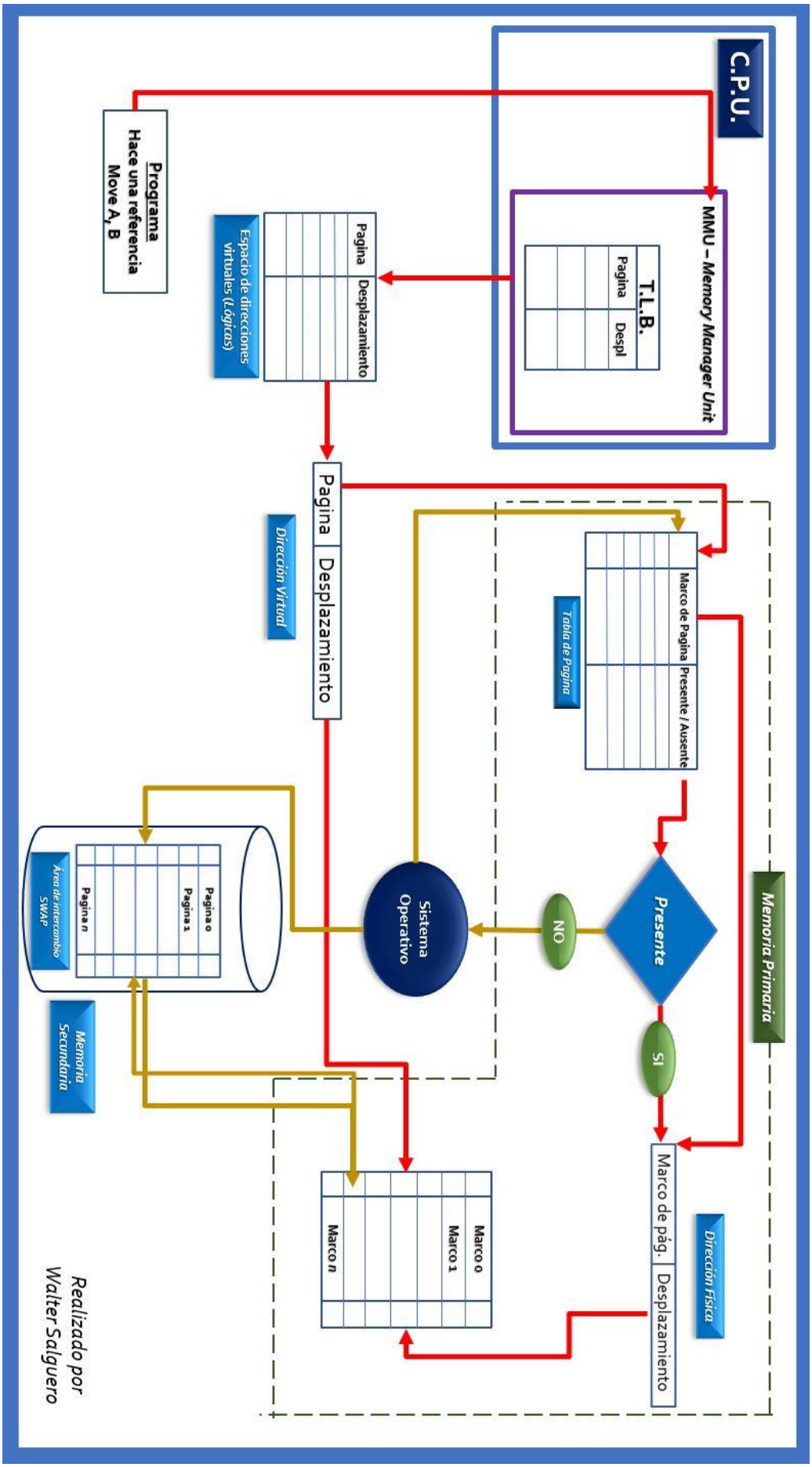
3.2.7.1. Otras consideraciones a tener en cuenta

Cuando se utilizan *tablas de páginas* de gran tamaño, puede que buena parte de las tablas de los procesos activos se encuentren descargados en *memoria secundaria*, perjudicando su rendimiento.

Además, el diseño de la *memoria secundaria* suele estar orientado al manejo de bloques grandes, que se adecuan mejor al uso de páginas de mayor tamaño.

Sin embargo, si utilizamos páginas pequeñas, cuando un proceso lleve un tiempo ejecutándose, todas sus páginas de memoria tendrán referencias con una alta probabilidad de ser utilizadas, lo que reducirá la *tasa de fallos de página*.

En cualquier caso, el tamaño de página deberá estar relacionado con la cantidad de *memoria principal* y el uso que vayamos a hacer de ella. Por ejemplo, las técnicas de *programación orientada a objetos* producen programas con bloques de código pequeños que generan referencias a posiciones de memoria dispersas en breves periodos de tiempo. Por su parte, las *aplicaciones multihilo* suelen modificar bruscamente el flujo de referencias a instrucciones y datos.


C.P.U.
MMU – Memory Manager Unit
T.L.B.
Pagina
Despl
Programa
Hace una referencia
Move A, B
Espacio de direcciones virtuales (Lógicas)
Pagina
Desplazamiento
Pagina
Desplazamiento
Dirección Virtual
Tabla de Pagina
Marco de Pagina
Presente / Ausente
Memoria Primaria
Presente
NO
SI
Sistema Operativo
Área de intercambio SWAP
Pagina 0
Pagina 1
Pagina n
Memoria Secundaria
Marco 0
Marco 1
Marco n
Marco de pág.
Desplazamiento
Dirección Física
Realizado por
Walter Salguero

Bibliografía

- Francisco Aylagas Romero, Elvira Martínez de Icaya Gómez, "*Sistemas Operativos1 - Ingeniería de Computadores*" - Universidad Politécnica de Madrid (UPM) y Escuela Universitaria de Informática (EUI)
- A.S. Tanenbaum: Modern Operating Systems (3rd edition). Prentice-Hall, 2008.
- Salguero Walter, Sistemas operativos1, UNPAZ