Getión de Memoria

Se explican los distintos esquemas de memoria, memoria contigua (particiones estáticas y dinámicas) y memoria no contigua (paginación, segmentación y segmentación paginada).

Tabla de Contenidos

Autor: Lina García Cabrera

Copyright: Copyright by Lina García Cabrera

1 Introducción

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Este módulo aborda el problema de la gestión de la memoria. Hace un breve estudio preliminar de las posibles alternativas y variantes a la hora de **organizar y administrar el espacio de direcciones de un sistema**.

Comienza por el tipo de gestión más básico, el de los **sistemas de monoprogramación**, que apenas necesitan de ninguna organización. La irrupción de los sistemas **multiprogramados** hace necesario tomar decisiones sobre aspectos tan diversos como:

- cuánto espacio se dedica a cada proceso,
- de qué modo se le asigna, en qué lugar se ubica,
- durante cuánto tiempo permanece en memoria,
- qué sucede si no existe suficiente espacio o
- cómo se protege frente a accesos ajenos.

Todos estos factores se valoran para:

- **técnicas de asignación contigua** (**particiones estáticas** y **dinámicas**) y
- para métodos de **asignación no contigua** (**paginación**, **segmentación** y **segmentación paginada**).

También se discutirá el soporte hardware, y el grado de protección y compartición que es posible con cada uno de los esquemas.

En las estrategias de **asignación no continua** tendrán un particular interés los **esquemas de traducción de direcciones**, por su repercusión en el ***tiempo efectivo de acceso a memoria*** y, por tanto, en el rendimiento del sistema.

Mantener varios procesos en memoria implica el **compartir la memoria** entre ellos. La ***memoria es una matriz de palabras o bytes, cada uno con su propia dirección:***

- La CPU extrae instrucciones de la memoria de acuerdo con el valor del contador de programa.
- Estas instrucciones provocan la carga de datos en direcciones de memoria específicas.

Por tanto, un programa en ejecución genera una secuencia de direcciones de memoria. El sistema operativo tiene que saber **cómo acceder a estas direcciones y qué espacio está libre u ocupado**.

OBJETIVOS

- Proporcionar un descripción de las distintas formas de gestionar la memoria.
- Analizar diversas estrategias de gestión de memoria (particiones, paginación y segmentación).
- Saber cómo se traduce una dirección dinámica y su repercusión en el rendimiento.

2 Gestión de Memoria. Conceptos Generales

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Para que un **proceso pueda ejecutarse debe estar ubicado en la memoria principal** del ordenador. Una parte del sistema operativo se va a encargar de gestionar la memoria principal, de forma que los procesos puedan residir en la memoria sin conflictos.

La gestión de la memoria implica varias tareas:

- llevar un **registro de qué zonas están libres** (es decir, no están siendo utilizadas por ningún proceso),
- qué **zonas están ocupadas, y por qué procesos**.

En sistemas en los que no todos los procesos, o no todo el código y datos de un proceso, se ubican en la memoria principal, el sistema operativo:

- debe **pasar** parte, o la totalidad del **código y datos de un proceso, de memoria a disco**, o viceversa.

De esta forma se libera al usuario de realizar estas transferencias de información, de las cuales no es consciente.

Además de la carga de los programas de disco a memoria y está la protección. Desde el momento en que varios procesos deben compartir la memoria del ordenador surge el problema de la **protección**. En general, se pretende que **un proceso no pueda modificar las direcciones de memoria en las que no reside**. Esto es así porque en las direcciones de memoria donde no está ubicado el proceso pueden residir otros procesos, o código y/o estructuras de datos del S.O.

> Si un proceso puede modificar indiscriminadamente la memoria, podría, por ejemplo, cambiar el valor de una dirección de memoria donde residiera una variable de otro proceso, con la consecuente ejecución incorrecta del proceso propietario de la variable.

Algunos sistemas ni siquiera permiten que un proceso pueda leer las direcciones de memoria en las que no reside, con esto se consigue **privacidad sobre el código y datos de los procesos**.

Existen varias formas de **gestionar la memoria**. Por lo común, la forma de gestión **dependerá de la máquina virtual que se quiera proporcionar y del hardware subyacente**.

Hay que decidir **qué estrategias se deben utilizar para obtener un rendimiento óptimo** con independencia de la forma de gestión.

Las estrategias de gestión de la memoria especifican el comportamiento de una organización de memoria determinada cuando se siguen diferentes políticas:

- ¿cuándo se toma un nuevo programa para colocarlo en la memoria?,
- ¿toma el programa cuando el sistema lo necesita, o se intenta anticiparse a las peticiones del sistema?,
- ¿qué lugar de la memoria principal se coloca el siguiente programa por ejecutar?,
- ¿se colocan los programas lo más cerca posible unos de otros en los espacios disponibles de la memoria principal para reducir al mínimo el desperdicio de espacio, o se colocan lo más rápido posible para reducir el tiempo empleado en tomar la decisión?

Los sistemas actuales son de memoria virtual, muchas de la formas de gestión estudiadas en este tema tienen principalmente valor histórico, pero sientan las bases de los sistemas actuales.

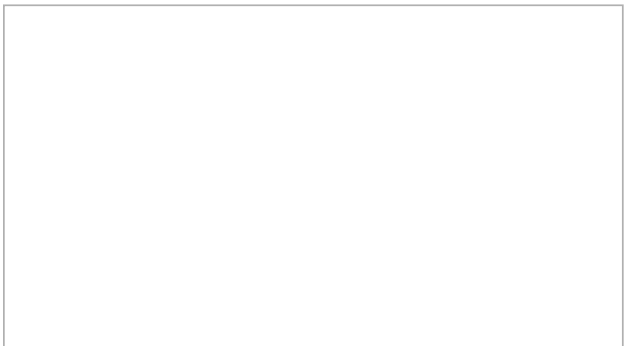
2.1 Jerarquía de Memoria

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Los programas y datos necesitan estar en la memoria principal para ser ejecutados, o para poder ser referenciados. Los programas o datos que no se necesitan de inmediato pueden guardarse en la memoria secundaria hasta que se necesiten, y en ese momento se transfieren a la memoria principal para ser ejecutados o referenciados. Los soportes de memoria secundaria, como cintas o discos, son en general menos caros que la memoria principal, y su capacidad es mucho mayor. Normalmente, es mucho más rápido el acceso a la memoria principal que a la secundaria.

En los sistemas con varios niveles de memoria hay muchas transferencias constantes de programas y datos entre los distintos niveles. Éstas consumen recursos del sistema, como tiempo de la CPU, que de otro modo podrían utilizarse provechosamente.

En los años sesenta se hizo evidente que la **jerarquía de la memoria** podía extenderse un nivel más, con una clara mejora del rendimiento. Este nivel adicional, la **memoria caché**, es una memoria de alta velocidad, mucho más rápida que la memoria principal. La memoria caché es extremadamente cara, si se compara con la principal, por lo que sólo se utilizan memorias caché relativamente pequeñas. La Figura 1 muestra la relación que existe entre la memoria caché, la principal y la secundaria.



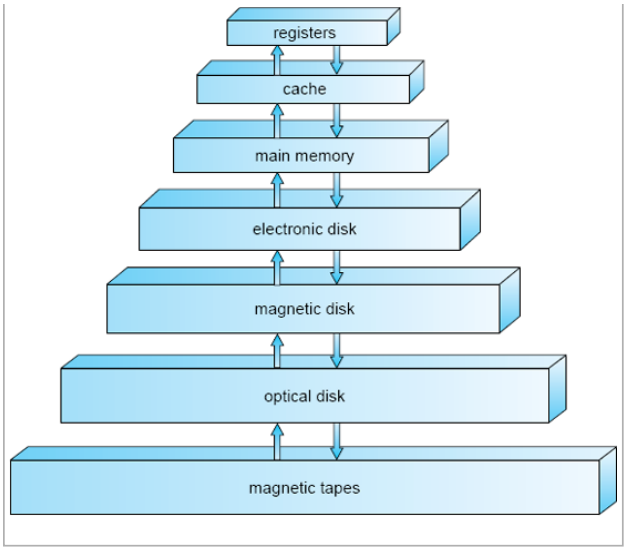


Figura 1. Jerarquía de Memoria [Silberschatz 2010].
La memoria caché introduce un nivel adicional de transferencia de información en el sistema. Los **programas en memoria principal se pasan a la memoria caché antes de ejecutarse**. En la memoria caché se pueden ejecutar mucho más rápido que en la principal. La esperanza de los diseñadores es que el trabajo extra requerido por la transferencia de programas sea mucho menor que el incremento del rendimiento obtenido por la ejecución más rápida en la caché.

3 Gestión Memoria: Monoprogramados

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
En los sistemas de **monoprogramación sólo existe un proceso de usuario**, que disfruta de todos los recursos del ordenador. Esto va a simplificar notablemente la gestión de la memoria, ya que ésta sólo debe ser compartida por los programas del sistema operativo, y por el único proceso de usuario existente.

Tal y como se muestra en la Figura 2. Dependiendo de detalles de diseño,

- el sistema operativo ocupará la parte baja de la memoria RAM, como se muestra en la Figura 2 (a);
- o la parte alta de la memoria ROM, como se muestra en la Figura 2 (b).
- El PC de IBM ubica parte del sistema operativo en RAM, y los gestores de dispositivos en ROM; a esta última parte se le llama BIOS (Basic Input/Output System, sistema básico de entrada/salida), esto último se ilustra en la Figura 2 (c).

Si el usuario conoce la ubicación en la memoria del sistema operativo, entonces puede escribir programas en términos de direcciones absolutas de memoria. Una **dirección absoluta** de memoria es una dirección física (es decir, real) de la memoria.

En contraposición se tienen las direcciones relativas. Un programa está escrito en término de **direcciones relativas** suponiendo que empieza a cargarse en la dirección cero de la memoria. Por lo general, los usuarios escriben programas en lenguajes de alto nivel, por lo que son los traductores los encargados de generar las direcciones que ocupan las variables, procedimientos, etc, en la memoria. Los compiladores no generan direcciones absolutas de memoria, pues no saben dónde se almacenarán los procesos.

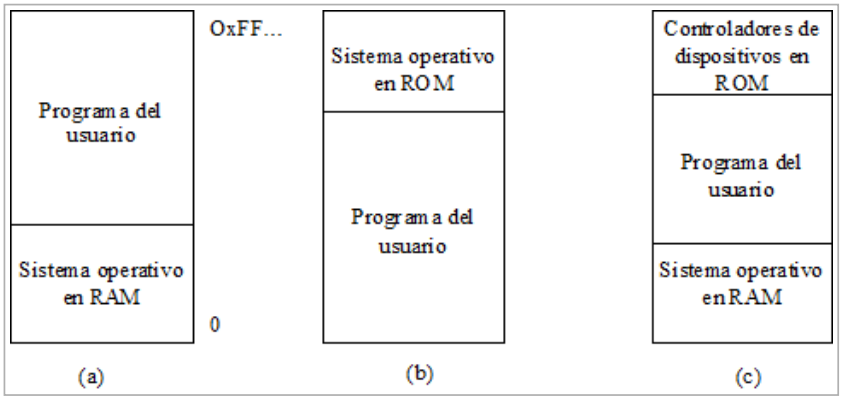


Figura 2. Tres formas de organización de la memoria, con un sistema operativo y un proceso de usuario.
Por lo común, los sistemas operativos monousuario de monoprogramación (muy comunes en las microcomputadoras) no tienen protección de la memoria. Por lo tanto, el único proceso de usuario que existe en la memoria, puede modificar posiciones de memoria pertenecientes al sistema operativo, esto provocaría errores al ejecutarse la zona modificada.

La protección se puede realizar mediante un registro de límite integrado en la CPU. Si se tiene un esquema como el de la Figura 2 (b) el **registro de límite** contendrá la dirección de inicio de carga del sistema operativo. El hardware, en tiempo de ejecución,

verifica que las direcciones generadas por el proceso de usuario no son superiores al valor del registro de límite. En caso de ser superior, el proceso de usuario intenta acceder al sistema operativo, esto provoca una interrupción hardware que gestiona el sistema operativo, normalmente eliminando al proceso.

4 Gestión de Memoria: Multiprogramados

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

En un sistema de **multiprogramación** la memoria debe ser compartida por varios procesos de cara a obtener una mayor utilización de los recursos del ordenador. Esto provoca que la gestión de la memoria se complique substancialmente.

En primer lugar, hay que **llevar un recuento de las zonas de memoria ocupadas por los procesos**. Así, cuando un nuevo proceso entre en la memoria se le **asignará una zona que estaba libre**. Otro problema a resolver viene dado por el hecho de que **en el momento de escribir un programa no se sabe en qué zona de memoria se ubicará**, siendo posible que durante la vida de un proceso éste cambie varias veces de emplazamiento. Habrá que tener en cuenta, también, la **protección de las zonas de memoria** ocupadas por los procesos, máxime en sistemas multiusuario, donde los procesos pueden pertenecer a distintos usuarios.

Existen varias formas de gestión de la memoria utilizadas en sistemas de multiprogramación.

5 Asignación de Memoria Contigua

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
En un esquema de asignación de **memoria contigua** ***un proceso se ubica en su totalidad en posiciones consecutivas de memoria***.
En este apartado se estudian dos métodos de asignación contigua empleados históricamente en sistemas multiprogramados:

- Particiones Estáticas o Fijas.
- Particiones Dinámicas.

5.1 Particiones Estáticas o Fijas

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Esta forma de gestión consiste en ***dividir la memoria en varias zonas, pudiendo ser cada zona de un tamaño diferente*** (ver la Figura 3). El tamaño de las zonas podrá ser modificado eventualmente por algún usuario responsable de la administración del ordenador.

Los trabajos se traducían mediante compiladores y ensambladores absolutos, para ejecutarse en una partición específica. Una vez introducido un proceso en una partición, permanece en ella hasta su finalización. Si un trabajo se iniciaba, y la partición para la que estaba compilado estaba ocupada, tenía que esperar, aunque estuvieran libres otras particiones. Esto provoca una **pérdida de eficiencia**.

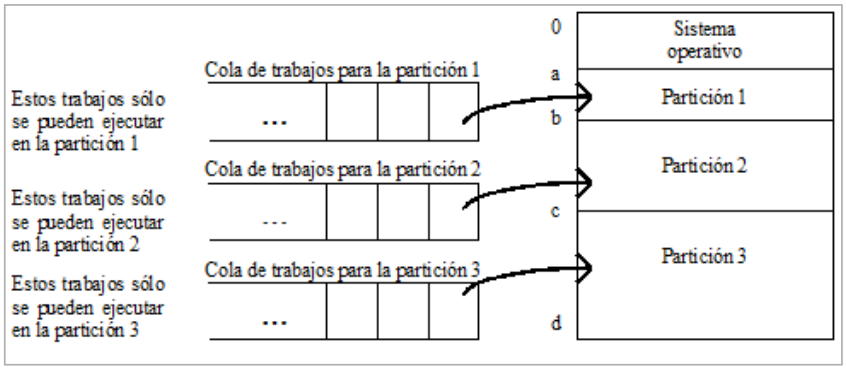


Figura 3. Multiprogramación con particiones estáticas, con traducción y carga absolutas.

De cara a ***mejorar el rendimiento es preciso que un proceso se pueda cargar en cualquier partición***. Para ello, los programas se escriben en términos de direcciones relativas a la dirección de comienzo de carga cero. Sin embargo, los programas no se cargan a partir de la dirección cero, esta circunstancia se debe resolver mediante la **reasignación** o **reubicación**, pero ¿cómo se realiza ésta?.

Una posible implantación viene proporcionada por el hardware. Existe un registro, denominado **registro base**, en el que el sistema operativo, dentro de la operación de cambio de proceso, escribe la dirección de memoria a partir de la cual se almacenó el

proceso. Esta dirección coincidirá con la de comienzo de la partición en que reside, y forma parte de su descriptor. Cuando la CPU genera una dirección de memoria, ésta es transformada por el hardware antes de ser introducida en el bus del sistema. La transformación consiste en **sumarle a la dirección el registro base**.

> Para clarificar esto, supóngase que la instrucción que actualmente ejecuta la CPU guarda el contenido del acumulador en la dirección relativa 100 de la memoria. Esta dirección, 100, (que podría, por ejemplo, guardar el contenido de una variable de otro proceso) es relativa a una dirección 0 de comienzo del programa. Si el programa se ha cargado en la posición 10000 de la memoria, el registro base contendrá el valor 10000. Cuando la CPU ejecuta la instrucción, genera la dirección 100 de memoria (la cual físicamente pertenece a otro proceso). Sin embargo, el hardware suma 10000 a este valor, introduciéndose 10100 en el bus del sistema, dirección que realmente ocupa la variable. Observe que, con este esquema, si se quiere reubicar el proceso a otro lugar de la memoria, sólo hay que desplazarlo de lugar, y modificar el registro base con la nueva dirección de comienzo de carga.

Una **solución software al problema de la reasignación** consiste en modificar las instrucciones cuando el programa se carga en la memoria. Para que esto ocurra es preciso que el enlazador (el programa que a partir de los ficheros objeto genera un único objeto) incluya en el programa binario una lista que indique qué partes del programa son direcciones a reasignar, y cuáles no (constantes, códigos de operadores u otros elementos que no deban ser reasignados). Con esta solución, cada reubicación en la memoria implica una reasignación de las direcciones de memoria del programa.

5.1.1 Protección

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
Si se tiene el esquema hardware del **registro base**, para lograr la protección de las zonas de memoria basta con añadir un nuevo registro, denominado **registro límite**. Este registro guarda la última dirección de la partición, y forma también parte del PCB del proceso. El hardware, después de sumar el registro base a la dirección relativa, comprueba que la dirección obtenida no supere el valor del registro límite. Si se supera el valor, se está intentando acceder a una zona que no corresponde al proceso; en esta situación, el hardware genera una interrupción. El sistema operativo sirve a la interrupción, lo normal es que mande una señal al proceso por violación de memoria. Si el proceso no tiene definido atrapar esa señal, lo cual es lo más probable, se eliminará al proceso.

Cuando un proceso quiera ejecutar código del sistema operativo, por ejemplo, para realizar una E/S, no tiene acceso directo a las rutinas que tiene el sistema operativo en memoria para implementar dicha función, sino que debe realizar una llamada al sistema para no violar la protección. Este esquema es necesario, pues los programas de usuario tienen que avisar al sistema operativo de que le solicitan servicios (al hacer una llamada al sistema), el sistema operativo atenderá las peticiones si son correctas, o si puede facilitarlas en dicho momento (por ejemplo, no se asignará una impresora que está siendo utilizada por otro proceso). Los procesos de usuario no pueden llamar directamente al sistema operativo gracias a la protección de memoria.

> Una última observación, al margen de la protección: cuando un proceso se introduce en una partición, lo más probable es que su tamaño no sea el mismo (es decir, sea algo menor) que el de la partición. Esto origina un problema de desperdicio de memoria conocido como **fragmentación interna**.

5.2 Particiones Dinámicas

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
En este método se va asignando la memoria dinámicamente a los procesos, conforme se introducen en la memoria. A cada proceso se le asigna exactamente la memoria que necesita.

> En la Figura 4 se ilustra cómo evoluciona la ocupación de la memoria en un sistema de este tipo. Al principio sólo se encuentra el proceso A en la memoria. Después, se insertan los procesos B y C . En la Figura 4 (d) A concluye. Luego, D entra y B sale. Por último E entra.

Con este método de gestión de la memoria se evita el problema de la fragmentación interna. Sin embargo, aparece el problema de la **fragmentación externa** entre particiones, el cual se aprecia en la Figura 4 (f).

El problema consiste en que se creen huecos libres demasiado pequeños como para que quepan procesos, aunque la unión de todos esos huecos produciría un hueco considerable, lo que acarrea el desperdicio de la memoria.

Una posible solución es la **compactación** de la memoria, que consiste en desplazar todos los procesos hacia la parte inferior de la memoria mientras sea posible. Como la compactación lleva mucho tiempo, a veces no se realiza, o se hace por la noche, en horas de poco uso del ordenador. Hay que tener en cuenta que **el sistema debe detener todas sus actividades mientras realiza la compactación**. Ello puede ocasionar tiempos de respuesta irregulares para usuarios interactivos, y podría ser devastador en un sistema de tiempo real. Además, con una combinación normal de trabajos que cambia rápidamente, es necesario **compactar a menudo**. En este caso, los recursos del sistema que se consumen quizá no justifiquen las ventajas de la compactación.

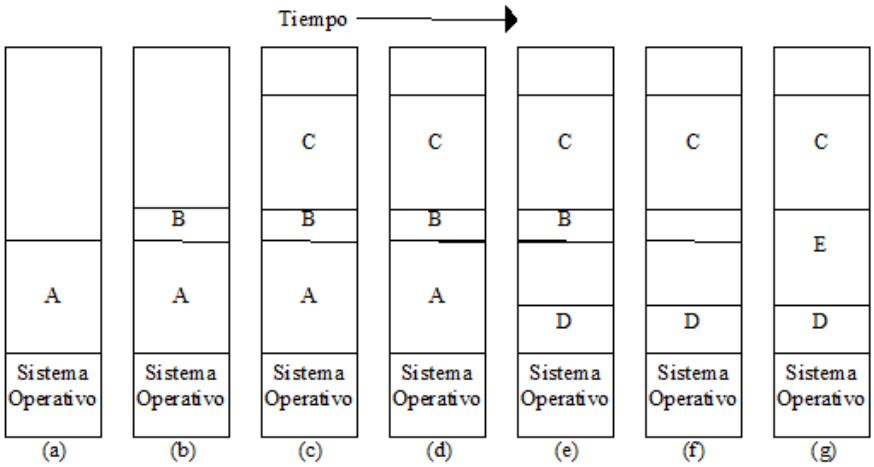


Figura 4. La asignación de memoria cambia cuando el pr oceso llega o sale de la memoria. Las r egiones sombr eadas son memoria libre.

El esquema de los **registros base y límite** sigue siendo válido para **la reasignación y la protección**. Otro tema a tener en cuenta es la cantidad de memoria por asignar a un proceso recién creado. Si los procesos se crean con un tamaño fijo invariante, la asignación es muy sencilla, se asigna exactamente lo que se necesite.

Si, por el contrario, los segmentos de datos de los procesos pueden crecer, como es el caso de la asignación dinámica de memoria a partir de una pila, que ocurre en muchos lenguajes de programación, aparece un problema cuando un proceso intenta crecer. Si hay un hueco adyacente al proceso, éste puede ser asignado, y el proceso podrá crecer hacia el hueco. Sin embargo, si el proceso es adyacente a otro proceso, el proceso de crecimiento deberá ser desplazado a un hueco de la memoria lo suficientemente grande; o bien, habrá que eliminarlo.

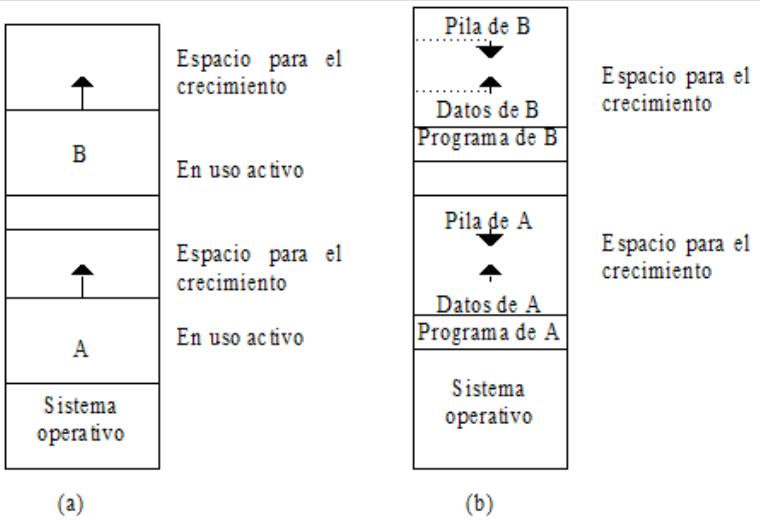


Figura 5. (a) Asignación de espacio a un segment o de datos que crece. (b) Asignación de espacio a una pila y un segment o de datos que crece.

Si es de esperar que la mayoría de los procesos crezcan conforme se ejecuten, sería una buena idea **asignar un poco de memoria adicional** siempre que un proceso pase a la memoria, con el fin de reducir el gasto excesivo asociado con el traslado de procesos que ya no tienen espacio suficiente en su memoria asignada. En la Figura 5 - 5 (a) vemos una configuración de la memoria en la que se asignó a dos procesos el espacio adicional para el crecimiento.

Si los procesos pueden tener dos segmentos de crecimiento, como por ejemplo, el segmento de datos, que se utiliza como una pila, y el *stack*, se sugiere un método alternativo, el de la Figura 5 - 5 (b). En esta figura se puede ver que cada proceso tiene un *stack* de crecimiento hacia abajo, en la parte superior de la memoria asignada a él; además, tiene un segmento de datos justo encima del programa, el cual crece hacia arriba. La memoria entre ellos se puede utilizar para cualquiera de los segmentos. Si el espacio se agota, puede ocurrir que el proceso sea desplazado a un hueco con el espacio suficiente; o bien, ser aniquilado.

5.2.1 Registro Ocupación de la Memoria

En el sistema de **particiones estáticas** es sencillo llevar el registro de la ocupación de la memoria, basta con **guardar sobre cada partición si está libre, u ocupada y por qué proceso, así como sus direcciones de comienzo y fin de partición**.

Por contra, con las **particiones dinámicas**, el número de éstas varía con el tiempo, así como su tamaño. Una forma posible de registrar la ocupación de la memoria es utilizar una **lista enlazada** de los segmentos de la memoria asignados o libres.

La memoria de la Figura 6 - 6 (a) se presenta en la Figura 6 - 6 (c) como una lista enlazada de segmentos. Cada entrada de la lista especifica un hueco (H) o un proceso (P), la dirección donde comienza, su longitud, y un puntero a la siguiente entrada.

En este ejemplo, la lista de segmentos está ordenada por direcciones. Este orden tiene la ventaja de que al terminar un proceso la actualización de la lista es directa. Un proceso que termina, tiene dos vecinos (a menos que se encuentre en la parte superior o inferior de la memoria).

Estos pueden ser procesos o huecos, lo que produce las cuatro combinaciones de la Figura 6 - 7. En la Figura 6 - 7 (a), la actualización de la lista requiere el reemplazo de una P por una H. En la Figura 6 - 6 (b) y (c), dos entradas se funden en una, y la lista se acorta en una entrada. En la Figura 6 - 6 (d) tres entradas se fusionan en una. Puesto que en el descriptor del proceso que termina se guardará un puntero a la entrada de la lista enlazada que ocupa dicho proceso, sería conveniente que la lista fuera doblemente enlazada. Esta estructura facilita la búsqueda de la entrada anterior y, por tanto, la verificación de si es posible una fusión.

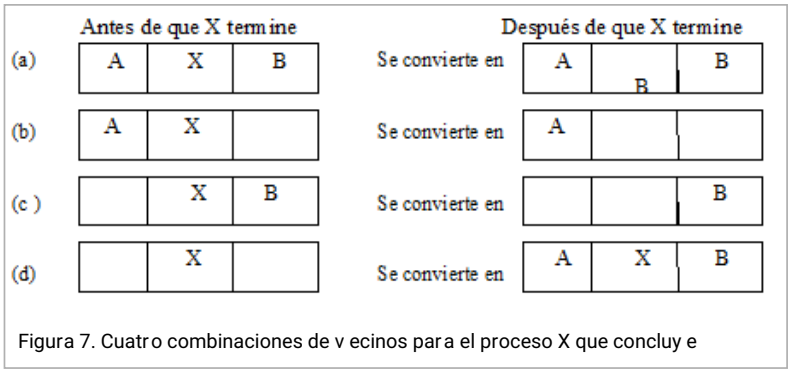


Figura 7. Cuatro combinaciones de v ecinos para el proceso X que concluy e

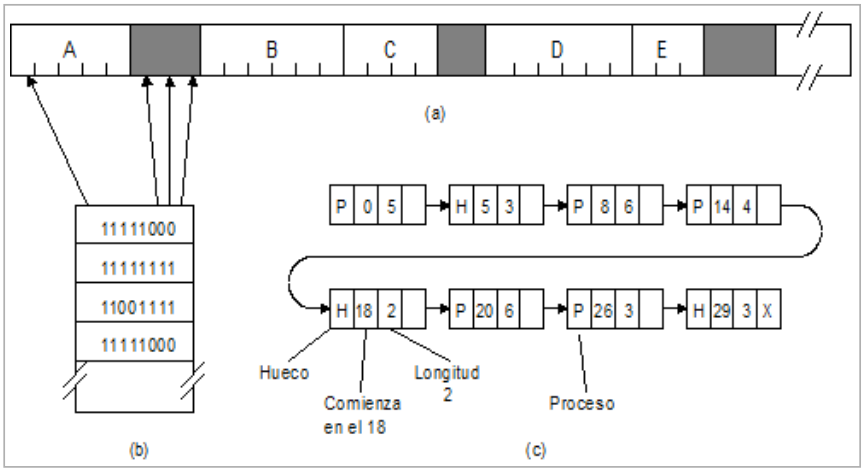


Figura 6. (a) Una parte de la memoria, con cinco procesos y 3 huecos. La marca muestra las unidades de asignación de la memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información como lista enlazada

5.2.2 Estrategias de Colocación

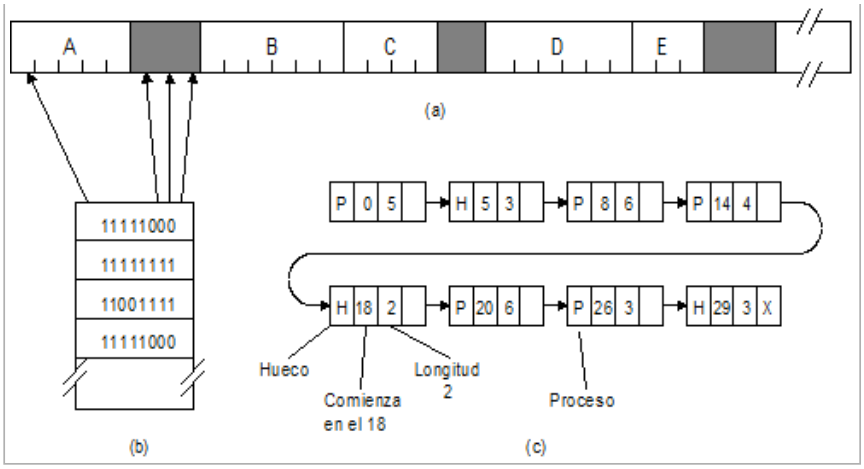
2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Cuando en un sistema de particiones dinámicas se debe asignar memoria principal para un nuevo proceso, y los procesos y huecos se mantienen en una lista ordenada por direcciones, se pueden utilizar diversos algoritmos para la elección del hueco de memoria donde ubicar al proceso. Supongamos que se conoce la cantidad de memoria por asignar.

El algoritmo más simple es **el primero en ajustarse** (*first fit*). Se revisa la lista de huecos hasta encontrar un espacio lo suficientemente grande. El espacio se divide entonces en dos partes, una para el proceso, y otra para la memoria no utilizada, excepto en el caso poco probable de un ajuste perfecto. Este algoritmo es rápido, ya que busca lo menos posible.

Otro algoritmo es el **mejor en ajustarse** (*best fit*), el cual busca en toda la lista, y elige el mínimo hueco suficientemente grande como para ubicar al proceso. Este algoritmo intenta que los huecos que se creen en la memoria sean lo más pequeños posible.

Como ejemplo de los algoritmos, retomemos la Figura 6. Si se necesita un bloque de tamaño 2, el primero en ajustarse asignará el espacio en 5, mientras que el mejor en ajustarse asignará el espacio en 18.



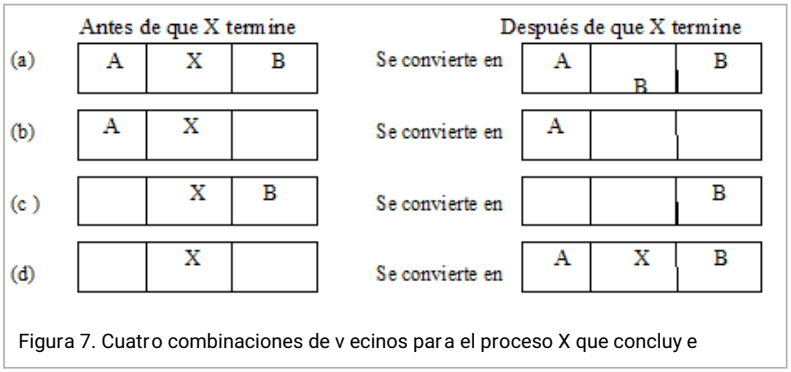


Figura 7. Cuatro combinaciones de v ecinos para el proceso X que concluy e

Realizando simulaciones se ha demostrado que ***el algoritmo del mejor ajuste desperdicia más la memoria***, pues al crear huecos demasiado pequeños, éstos no pueden ser utilizados por procesos.

Un algoritmo que enfrenta el problema de la manera contraria es el del **peor ajuste** (*worst fit*). En este algoritmo se elige el hueco más grande disponible. De esta forma aumenta la probabilidad de que el nuevo hueco creado sea lo suficientemente grande como para albergar un proceso.

5.2.3 Intercambio (Swapping)

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
En un sistema con **particiones estáticas** ***el número de procesos con posibilidades de estar en estado listo viene determinado por el número de particiones***, y en uno de **particiones dinámicas** ***por el tamaño de la memoria principal y el tamaño de los procesos***, ya que en ambos métodos un proceso permanece en una partición hasta que finaliza.

> Supongamos un sistema en que dicho número es cinco, es muy probable que en un momento dado los cinco procesos que ocupan las particiones estén bloqueados (por ejemplo, porque esperan la finalización de una operación de E/S). Mientras los cinco procesos permanezcan bloqueados se desperdicia la CPU.

Para remediar este inconveniente muchos sistemas operativos optaron por **permitir ejecutar concurrentemente más procesos de los que caben físicamente en la memoria principal** del ordenador. Para ello se utiliza la memoria secundaria (generalmente los discos) que, aunque más lenta, tiene mayor capacidad de almacenamiento que la principal. La solución consiste en tener en disco una copia de la parte de la memoria que ocupa todo proceso. En el disco se encuentran todos los procesos, en la memoria sólo unos cuantos. Para que un **proceso se pueda ejecutar debe residir en memoria principal**.

La razón por la que se aumenta el número de procesos con posibilidades de tener instrucciones en memoria principal es porque cuanto mayor sea este número, es menos probable que se dé la circunstancia de que todos estén bloqueados y, por lo tanto, es menor la posibilidad de que la CPU permanezca inactiva.

Algunos sistemas UNIX utilizaban el intercambio en un sistema de particiones dinámicas. El movimiento de procesos entre la memoria principal y el disco lo realizaba el **planificador de nivel medio o a medio plazo**, conocido como el intercambiador (*swapper*). El intercambio de la memoria principal al disco (*swapping out*) se iniciaba cuando el sistema operativo precisa memoria libre (y estaba toda ocupada) debido a alguno de los siguientes eventos:

1. Una llamada al sistema fork que necesitaba memoria para un proceso hijo.
2. Una llamada al sistema brk de solicitud de memoria dinámica en un proceso que no tiene suficiente memoria libre como para aceptar la petición.
3. Una pila que se agranda, y ocupa un espacio mayor al asignado.

Además, cuando había que recuperar un proceso presente en el disco (*swapping in*) desde hace mucho tiempo, con frecuencia se necesitaba sacar a otro proceso de memoria a disco, para disponer de espacio para el primero.

El intercambiador elegía una víctima al examinar los procesos bloqueados en espera de algo (por ejemplo, una entrada del terminal). Es **mejor sacar a un proceso bloqueado** (pasará a estado suspendido_bloqueado) que sacar a uno listo. Si existían varios procesos bloqueados ubicados en la memoria principal se elegía a uno cuya combinación de prioridad y tiempo de residencia en memoria principal fuera más desfavorable. Así, un buen candidato era un proceso que hubiera consumido mucho tiempo de CPU recientemente, al igual que uno que hubiera permanecido en la memoria durante mucho tiempo, aun cuando durante este tiempo hubiera realizado E/S. Si no se dispone de procesos bloqueados, entonces se elegía a un proceso listo en base a los mismos criterios.

Cada pocos segundos el intercambiador examinaba la lista de procesos intercambiados para ver si alguno estaba suspendido_listo. En caso de que existiera alguno, se seleccionaba a aquel que hubiese permanecido en el disco durante mucho tiempo. A continuación el intercambiador verificaba si **el intercambio sería fácil o difícil**. Un intercambio fácil era aquel en el que existiera la suficiente memoria libre, de forma que no había necesidad de sacar a un proceso para hacer espacio para el nuevo. Un intercambio difícil precisaba la eliminación de uno o más procesos. La implantación de un intercambio fácil se llevaba a cabo al traer el proceso a la memoria. Un intercambio difícil se implantaba liberando primero la memoria suficiente sacando a disco a uno o más procesos, y después, cargando en la memoria el proceso deseado.

Este algoritmo se repetía hasta que se cumpliera alguna de estas condiciones:

1. ningún proceso en el disco está suspendido_listo; o
2. la memoria está tan ocupada por procesos recién traídos que no hay espacio para más.

Para evitar un trasiego excesivo entre memoria y disco que pudiera afectar al rendimiento no se intercambiaba hacia el disco a un proceso hasta que hubiera permanecido en la memoria durante 2 segundos.

El espacio libre en la memoria principal y en el dispositivo de intercambio se registraba mediante una lista enlazada de huecos. Si se necesitaba espacio en alguno de los dos, el algoritmo del primero en ajustarse leía la lista adecuada de huecos, y devolvía el primer hueco que encontrara y que fuese lo bastante grande, a la vez que eliminaba este espacio de la lista de huecos.

6 Asignación de Memoria No Continua

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
Los esquemas de administración de la memoria pueden basarse en **asignaciones No continua de la memoria**. Un ***proceso puede dividirse en bloques, y estos bloques pueden situarse en posiciones no contiguas de memoria principal***. Es más, **no es preciso que se encuentren en la memoria todos los bloques de un proceso** para que se pueda ejecutar, basta con que se encuentren los bloques que contienen código o datos actualmente referenciados, el resto puede permanecer en memoria secundaria.

6.1 La Memoria Virtual

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

La clave del concepto de memoria virtual es la disociación de las direcciones a las que hace referencia un proceso en ejecución de las direcciones disponibles en la memoria principal.

Las direcciones a las que hace referencia un proceso en ejecución, en este esquema, se llaman **direcciones virtuales**. El **intervalo de direcciones virtuales a las que puede hacer referencia un proceso en ejecución** se llama **espacio de direcciones virtuales**, V, del proceso. El **intervalo de direcciones reales de la memoria principal** de un ordenador concreto se llama **espacio de direcciones reales**, R. El número de direcciones de V se denota |V|, y el número de direcciones de R, |R|. En los sistemas de almacenamiento virtual ya implantados lo normal es que |V| >> |R|, aunque se han construido sistemas en los que |V| < |R|.

La **memoria virtual** es una técnica de gestión de la memoria que ***posibilita que el espacio de direcciones virtuales sea mayor al espacio de direcciones reales***. En otras palabras, se permite **hacer programas de tamaño mayor al de la memoria principal**.

Para lograr esto, el sistema operativo se encarga de mantener en la memoria principal solamente aquellas partes del espacio de direcciones del proceso que actualmente están siendo referenciadas, el resto permanece en disco.

La memoria virtual se basa en el hecho de que muchos programas presentan un comportamiento conocido como **operación en contexto o localidad**, según el cual, en cualquier intervalo pequeño de tiempo un programa tiende a operar dentro de un módulo lógico en particular, sacando sus instrucciones de una sola rutina y sus datos de una sola zona de datos. De esta forma, **las referencias a memoria de los programas tienden a agruparse en pequeñas zonas del espacio de direcciones**. La localidad de estas referencias viene reforzada por la frecuente existencia de bucles: cuanto más pequeño sea el bucle, menor será la dispersión de las referencias.

> La observación de este comportamiento conduce al postulado (Denning, 1970) del llamado **principio de localidad**: "Las referencias de un programa tienden a agruparse en pequeñas zonas del espacio de direcciones. Estas zonas, además, tienden a cambiar sólo de forma intermitente".

La validez del principio de localidad varía de un programa a otro programa: será, por ejemplo, más válido en programas que lleven a cabo accesos secuenciales a vectores que en programas que accedan a estructuras complejas de datos.

La memoria virtual se compagina con la multiprogramación. Al no tener que almacenar los procesos enteros en la memoria, caben más en la memoria principal, con lo que es más probable que siempre exista un proceso en estado listo o preparado. Por otro lado, cuando un proceso espera a que se cargue en la memoria principal parte de su código o datos, se inicia una E/S con el disco. Mientras dura dicha E/S la CPU puede ejecutar otro proceso.

Aunque los procesos sólo hacen referencia a direcciones virtuales, deben ejecutarse en la memoria real. Por lo tanto, durante la ejecución de un proceso es preciso establecer la **correspondencia entre las direcciones virtuales y las reales**. Como se verá más adelante esta correspondencia debe realizarse de una manera rápida, pues si no, se ralentizaría demasiado el tiempo de ejecución de los procesos.

Se han desarrollado varios métodos para asociar las direcciones virtuales con las reales. Los **mecanismos de traducción dinámica de direcciones convierten las direcciones virtuales en direcciones reales en tiempo de ejecución**. Todos estos sistemas tienen la propiedad de que las direcciones contiguas en el espacio de direcciones virtuales de un proceso no son necesariamente contiguas en la memoria principal. Esto se conoce como contigüidad artificial (Figura 6 - 8). Debe quedar claro que toda esta correspondencia es transparente al programador, que escribe sus programas en términos de direcciones consecutivas de memoria virtual.

6.2 Esquema General de Traducción

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Los **mecanismos de traducción dinámica de direcciones deben mantener mapas de correspondencia** de traducción de direcciones que indiquen qué localidades de la memoria virtual están en memoria principal en un momento dado y dónde se encuentran.

Existen dudas en cuanto a si los bloques en que se dividen los procesos deben ser del mismo tamaño o de tamaños diferentes. Cuando **los bloques son del mismo tamaño**, se llaman **páginas**, y la organización de la memoria virtual correspondiente se conoce como **paginación**. Cuando **los bloques pueden tener tamaños diferentes** se llaman **segmentos**, y la organización de la memoria virtual correspondiente se llama **segmentación**. Algunos sistemas combinan ambas técnicas, con segmentos, que son entidades de tamaño variable, compuestos de páginas de tamaño fijo, **segmentación paginada**.

Las direcciones en un sistema de correspondencia son bidimensionales. Para referirse a un elemento en particular, el programa especifica el bloque en el que se encuentra el elemento, y su desplazamiento a partir del inicio del bloque. Una dirección virtual, v, se denota por un par ordenado (b,d), donde b es el número de bloque en el que se encuentra el elemento al que se hace referencia, y d es el desplazamiento a partir del inicio del bloque.

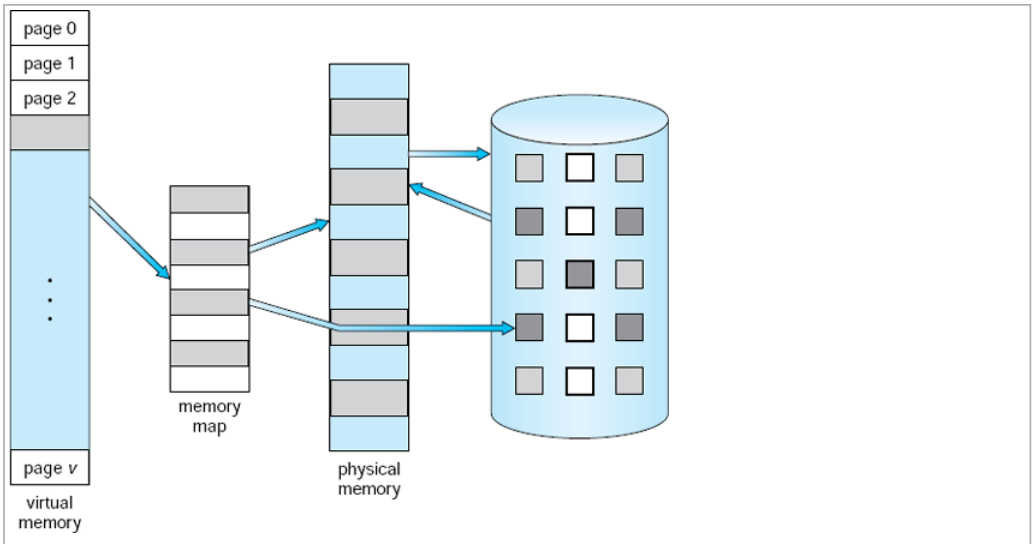


Figura 8. Correspondencia de elementos del espacio de direcciones virtuales al espacio de direcciones reales.

La traducción de una dirección virtual `v = (b,d)` a una dirección real, `r`, se lleva a cabo de la siguiente forma (Figura 9):

- Cada proceso tiene su propia tabla de correspondencia de bloques, mantenida por el sistema operativo dentro de la memoria principal.
- Un registro especial dentro de la CPU, denominado registro de origen de la tabla de correspondencia de bloques, se carga con la dirección real, a, de la tabla de correspondencia de bloques del proceso durante el cambio de proceso. La tabla contiene una entrada por cada bloque del proceso, y las entradas siguen un orden secuencial para el bloque 0, el bloque 1, etcétera.
- Ahora se suma el número de bloque, b, a la dirección base, a, de la tabla de bloques, para formar la dirección real de la entrada del bloque b en la tabla de correspondencia de bloques. Esta entrada contiene la dirección real, b', de inicio del

bloque b. El desplazamiento, d, se suma a la dirección de inicio del bloque, b', para formar la dirección real deseada, r = b' + d.

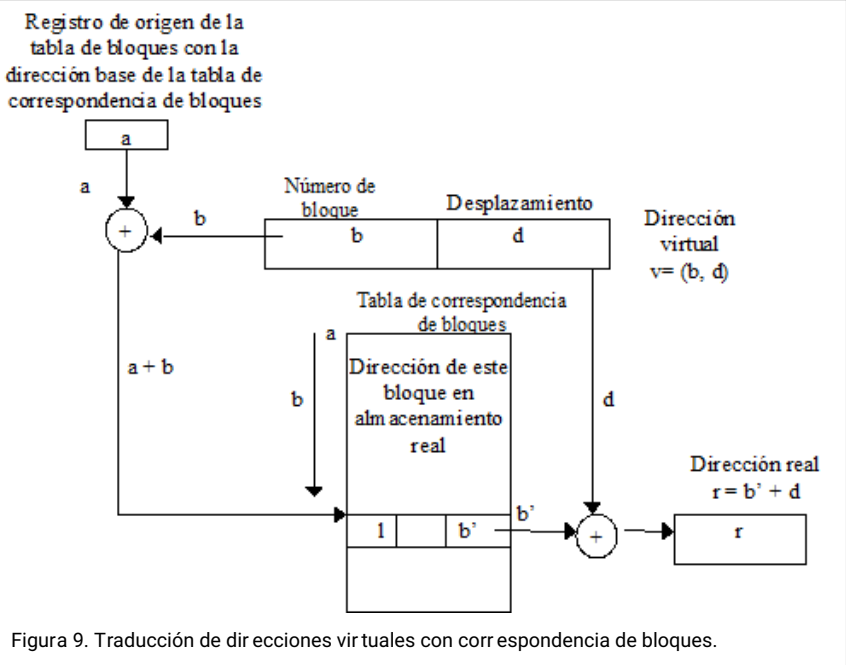


Figura 9. Traducción de direcciones virtuales con correspondencia de bloques.

Todas las técnicas de correspondencia de bloques empleadas en los sistemas de segmentación, paginación, y paginación y segmentación combinada son similares a la correspondencia mostrada en la Figura 9.

Es importante señalar que **la traducción de una dirección virtual a real la realiza una unidad hardware**, que transforma todas las direcciones generadas por la CPU antes de que pasen al bus del sistema. Es esencial que esta transformación la realice el hardware, y no el sistema operativo, pues muchas instrucciones máquina incluyen referencias a memoria, y la correspondencia debe realizarse rápidamente, para no ralentizar en exceso el tiempo de ejecución de los procesos. Por ejemplo, las dos sumas indicadas en la Figura 9 deben ser más rápidas que las sumas convencionales del lenguaje máquina.

Aunque el hardware consulta las tablas de correspondencia de bloques para la transformación de direcciones, es **el sistema operativo el encargado de rellenar y gestionar dichas tablas**.

Un proceso no tiene por qué tener todos sus bloques en memoria principal, recuérdese que el espacio de direcciones virtuales puede ser muy superior al espacio de direcciones reales, esto hace que a veces **un proceso referencie a una dirección de un bloque que no se encuentra en la memoria principal**. Para detectar esto, las tablas de correspondencias tienen un "**bit de presencia**" por entrada (cada entrada representa un bloque), que indica si el bloque se encuentra presente en la memoria principal o no.

El **hardware de traducción debe verificar este bit en cada referencia a memoria**.

1. Si el bloque no está en memoria principal, el hardware produce una interrupción.
2. Esta interrupción provoca que el control pase al software (sistema operativo), para que éste inicie la transferencia del bloque que falta desde la memoria secundaria a la memoria principal, y actualice de acuerdo con ello la tabla de correspondencias.
3. El proceso en ejecución pasará a bloqueado hasta que se haya completado esta transferencia. La posición de los bloques en la memoria secundaria puede guardarse en la misma tabla de correspondencias.

6.3 Paginación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

La paginación es un esquema de gestión de memoria que permite cargar en memoria un proceso (su espacio de direcciones físicas) en **posiciones no contiguas**.

Ahora el espacio de direcciones virtuales de **un proceso se divide** en trozos llamados **páginas del mismo tamaño**. La **memoria principal se divide también en marcos** o páginas físicas del mismo tamaño. Estos marcos son compartidos entre los distintos procesos que haya en el sistema, de forma que en cualquier momento un proceso dado tendrá unas cuantas páginas residentes en la memoria principal (sus páginas activas) y el resto en la memoria secundaria (sus páginas inactivas). El mecanismo de paginación cumple dos funciones:

1. llevar a cabo la **transformación de una dirección virtual a física**, o sea, la determinación de la página a la que corresponde una determinada dirección de un programa, así como del marco, si lo hay, que ocupa esta página;

2. **transferir**, cuando haga falta, **páginas de la memoria secundaria a la memoria principal**, y de la memoria principal a la memoria secundaria cuando ya no sean necesarias (memoria virtual).

6.3.1 Transformar una dirección virtual a física en Paginación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

La transformación de una dirección virtual a física necesita cierto respaldo hardware. Toda dirección generada por la CPU está dividida en dos partes: *un número de página* (p) y un *desplazamiento* (d) u offset. El número de página se utiliza como índice en la tabla de páginas. La tabla de páginas contiene el número de marco en memoria física.

El tamaño de página que igual al del marco está definido por el hardware, es una potencia de 2, variando entre 512 bytes y 16 MB.

Tener una potencia de 2 como tamaño de página hace que la traducción de una dirección lógica a un número de página y a un desplazamiento resulte fácil.

Si el tamaño de direcciones es 2^m y el tamaño de página es 2^n bytes o palabras, entonces los $m - n$ **bits de mayor peso** se interpretan como el **número de página** y los n **bits de menor peso** como el **desplazamiento dentro de la página**.

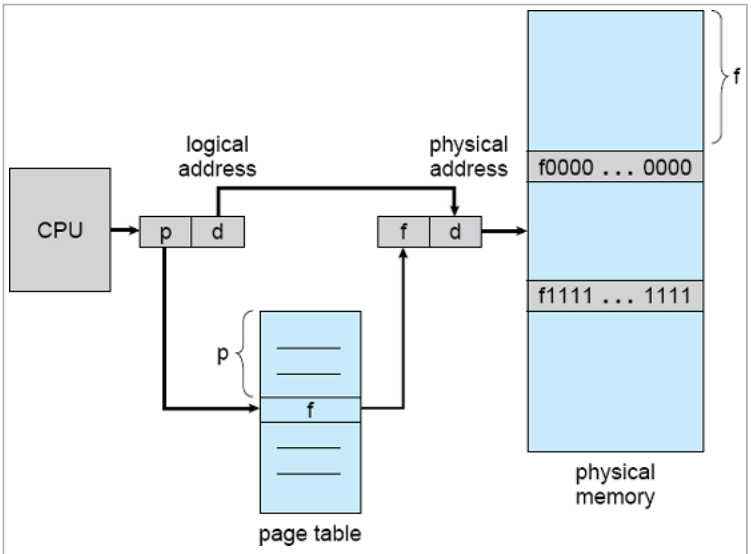


Figura 10. Transformar una dir ección lógica en física.

	Cache de memoria principal	Memoria virtual (Paginación)	Cache de disco
Proporción típica entre tiempos de acceso	5:1	10^6:1	10^6:1
Sistema de gestión de memoria	Implementado por un hardware especial	Combinación de hardware y software de sistema	Software de sistema
Tamaño típico de bloque	4 a 128 bytes	64 a 4096 bytes	64 a 4096 bytes
Acceso del procesador al segundo nivel	Acceso directo	Acceso indirecto	Acceso indirecto

Tenemos un tamaño de página de $4bytes$ y una memoria física de $32bytes$ (8 páginas) y tenemos la dirección lógica 0 (página 0, desplazamiento 0). Indexamos en la tabla de páginas, y vemos que la página 0 está en el marco 5. Por tanto:

- La dirección lógica 0 (página 0, desplazamiento 0) se corresponde con la dirección física $20 = ((5x4) + 0)$.
- La dirección lógica 3 (página 0, desplazamiento 3) con la dirección física 23 = ((5x4) + 3).
- La dirección lógica 4 (página 1, desplazamiento 0), la página 1 está en el marco 6, con la dirección física $24 = ((6x4) + 0)$.
- La dirección lógica 13 (página 3, desplazamiento 1), la página 3 está en el marco 2, con la dirección física $9 = ((2x4) + 1)$.

La transformación de número de página y de palabra en la dirección física de memoria se realiza a través de una tabla de páginas, cuyo $p-$ésimo elemento contiene la posición p' del marco que contiene a la página p (la posibilidad de que la $p-$ésima página no se encuentre en la memoria principal se abordará dentro de un momento). El número de palabra, d, multiplicada por el $tamañodepágina$, se suma a p' para obtener la dirección buscada (Figura 10).

La transformación de direcciones consiste, pues, en:
$f(a) = f(p,d) = p' * tamañopágina + d$
donde la dirección de programa, a, el número de página, p, y el número de palabra, d, están relacionados con el tamaño de página Z a través de:
$p =$parte entera de (a/Z)
$d =$ resto de (a/Z)

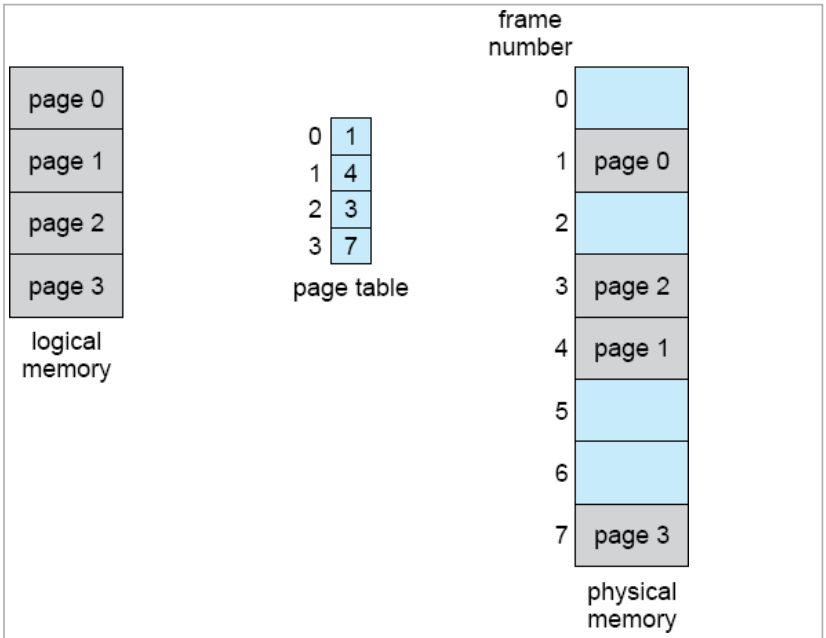


Figura 11. Gestión Memoria P aginado. Corr espondencia entr e memoria lógica y vir tual.

La **división de la dirección en número de palabra y número de página es tarea del hardware**, siendo transparente al programador: por lo que a él concierne, está programando en un espacio secuencial de direcciones muy grande.

Así pues, cada vez que la CPU genere una dirección de memoria ésta es transformada por una unidad hardware, de forma que en el bus del sistema se introduce la dirección física correspondiente. Es importante observar que la paginación es en sí misma una forma de reubicación dinámica. Cada dirección lógica es transformada en alguna dirección física por el hardware de paginación. Observe también que si el tamaño de página (como es lo usual) es una potencia de 2, el hardware no precisa realizar ninguna división, simplemente sabe que los últimos $nbits$, si el tamaño de página es de $2n$, representan el desplazamiento, y los primeros bits la página.

Cada proceso debe tener su propia tabla de páginas, y su dirección de comienzo en la memoria principal forma parte de la porción del PCB utilizada para realizar un cambio de proceso.

Como el número de marcos (cantidad de memoria real) asignados a un proceso será normalmente menor que el número de páginas que éste utiliza, es muy posible que una dirección del programa haga referencia a una página que no se encuentre en aquel momento en la memoria principal.

En este caso el elemento correspondiente de la tabla de páginas estará vacío, provocando el hardware una interrupción de "**fallo de página**" si se intenta acceder a ella. Esta interrupción provoca que el control pase al software (al sistema operativo), para que éste inicie la transferencia de la página que falta desde la memoria secundaria a la memoria principal, y actualice de acuerdo con ello la tabla de páginas. El proceso en ejecución pasará a bloqueado hasta que se haya completado esta transferencia.

La posición de las páginas en la memoria secundaria puede guardarse en una tabla separada o en la misma tabla de páginas. En este último caso, es necesario un "**bit de presencia**" en cada elemento de la tabla de páginas, para indicar si la página se encuentra presente o no en la memoria principal, y si el campo de direcciones debe interpretarse como una dirección de marco, o bien como una dirección de la memoria secundaria.

Si **no existe ningún marco vacío** en el momento en que ocurre un fallo de página, hay que guardar en la memoria secundaria alguna otra página con el fin de hacer sitio a la nueva. La elección de la página que habrá que sacar es el resultado de un algoritmo de reemplazo de página, del cual veremos varios ejemplos en el tema siguiente. Por el momento, vamos a destacar tan sólo el hecho de que la información que necesita el algoritmo de cambio de página, puede estar contenida en algunos bits adicionales que se añaden a cada elemento de la tabla de páginas.

Quizás habría que aclarar, que toda la operación de transformaciones de direcciones la lleva a cabo el hardware, excepto en el caso en que haya que traer una página de la memoria secundaria. En este caso, la aplicación del algoritmo de cambio de página, así como la actualización de la tabla de páginas, las lleva a cabo el software.

Esta es la visión general de cómo funciona la paginación. Para que la implementación sea viable, la transformación de dirección virtual a física debe ser rápida.

6.3.2 Respaldo Hardware

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
La mayoría de los sistemas operativos asigna **una tabla de páginas para cada proceso**. En el bloque de control de proceso se guarda un puntero a esta tabla (junto a otros valores de los registros). Cuando se despacha el proceso se deben cargar los registros de usuario y los valores correctos de la tabla de páginas *hardware* a partir de la tabla almacenada en la páginas de usuario.
La implementación *hardware* de la tabla de páginas puede hacer de varias formas:

- Mediante **registros**, la carga de estos registros la hace de despachador y se hace de forma muy rápida. El uso de registros para la tabla de páginas es posible si su tamaño es muy pequeño, imposible para tablas con millones de entradas (lo usual en los actuales computadores), demasiado caro.
- En **memoria principal**, un registro base de la tabla de páginas (PTBR, *page-table base register*) apunta a la tabla de páginas. Ahora sólo hay que cambiar este registro. Pero ahora el tiempo por cada referencia a memoria se dobla porque antes se accede a la tabla de páginas.
- Utilizar una **memoria asociativa**, una caché hardware especial de pequeño tamaño y con acceso rápido, un búfer de consulta de traducción (TLB, *translation look-aside buffer*). Cada entrada de TLB está formada por: una clave y un valor. Cuando se busca una dirección, se compara a la vez con todas las claves. Si se encuentra, se devuelve el valor (el marco de página). Este hardware es caro, el número de entradas de TLB es pequeño (entre 64 y 1024). Si no encuentra la página en TLB, hay que consultar la tabla de páginas.
- La memoria asociativa debe contener los números de las páginas a las que haya mayores posibilidades de acceder. No existe ningún algoritmo general que nos asegure que así suceda. En la práctica se cargan cíclicamente en la memoria asociativa las direcciones de las páginas a las que se ha hecho referencia con más frecuencia recientemente. Este algoritmo, más bien primitivo, es, de hecho, bastante eficaz.

El porcentaje de veces que se encuentra un número de página en el TLB se llama tasa de acierto. Un tasa de acierto del 80% significa que el 80% de las veces encontramos el número de página deseado entre los registros asociativos.

> Si se tarda 20 nanosegundos en consultar el TLB y 100 nanosegundos en acceder a memoria principal, un acceso a memoria mediante TLB será 120 nanosegundos. Si no se encuentra en memoria asociativa (20 nanosegundos), hay que acudir a la tabla de páginas (100 nanosegundos) y luego acceder al byte o palabra deseada en memoria (100 nanosegundos), por tanto, 220 nanosegundos. El **tiempo efectivo de acceso a memoria**, es una media ponderada:
> **tiempo efectivo de acceso a memoria = 0,80 x 120 + 0,20 x 220 = 140 nanosegundos**
> En este ejemplo, sufrimos un 40% de retardo en el tiempo de acceso a memoria (de 100 a 140 nanosegundos) pero no del 100% (200 nanosegundos).

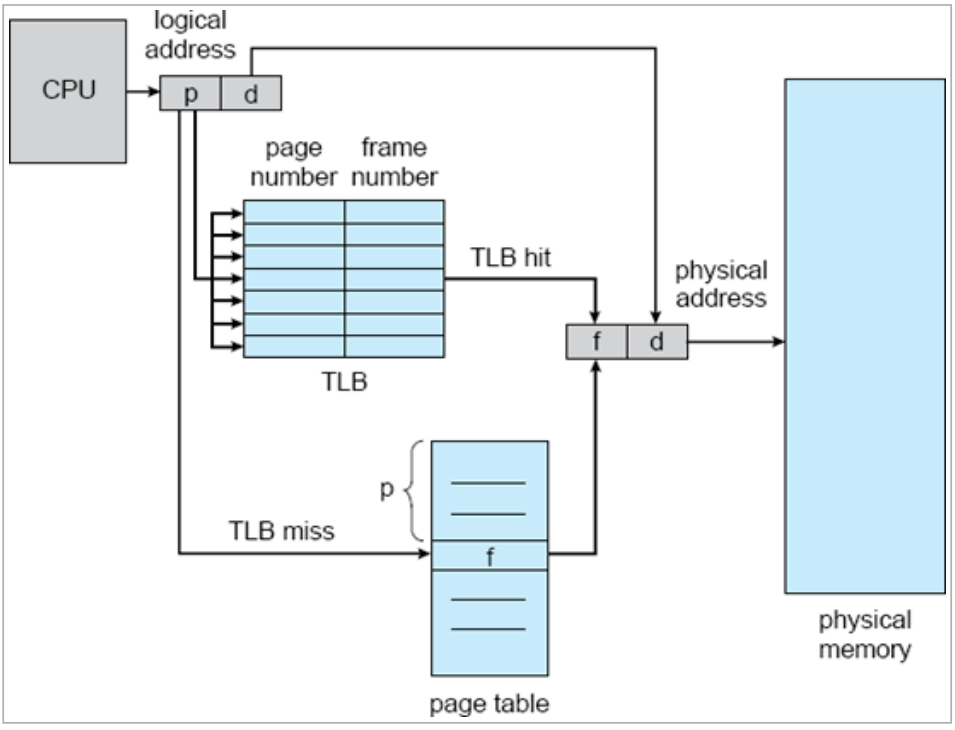


Figura 12. Paginación con memoria asociativa TLB.
Puesto que el sistema operativo gestiona memoria física, tiene que estar al tanto de la memoria física: qué marcos están asignados, qué marcos están disponibles, cuántos marcos hay en total, etc. Esta información se mantiene generalmente en una

estructura denominada **tabla de marcos**. La tabla de marcos tiene una entrada para cada marco, que indica si está libre o asignado y, si está asignado, a qué página de qué proceso. La asignación de marcos a páginas es trivial.

> Un proceso se divide en páginas de igual tamaño. El número de páginas de un proceso depende de su tamaño. Lo normal, es que este tamaño NO sea un múltiplo del tamaño de página. Por tanto, **la última página de cada proceso estará parcialmente libre/ocupada**. En media, se despercia la mitad de una página por proceso (**fragmentación interna**).

Además, el sistema operativo tiene que conocer qué procesos de usuario operan en el espacio de usuario, y tiene que transformar todas las direcciones lógicas para generar direcciones físicas. Si un usuario realiza una llamada al sistema (para realizar E/S) y da una dirección como parámetro (por ejemplo, un buffer), esa dirección tiene que ser traducida para generar la dirección física correcta. El sistema operativo puede utilizar la dirección de la tabla de páginas del proceso (que se guarda en su descriptor) para traducir las direcciones lógicas en físicas, siempre que tenga que realizar por él mismo la operación.

6.3.3 Protección

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
La **protección de la memoria** en un entorno paginado se consigue por medio de unos **bits de protección asociados a cada página**. Normalmente **estos bits se mantienen en la tabla de páginas**. Un bit puede definir que una página sea de lectura/escritura o de sólo lectura. Cada referencia a memoria pasa a través de la tabla de páginas para encontrar el número de marco correcto. Al tiempo que se calcula la dirección física, pueden verificarse los bits de protección para asegurar que no se escribe sobre una página de sólo lectura. Una tentativa de escribir sobre una página de sólo lectura ocasiona una excepción *hardware* al sistema operativo (por violación de memoria).
Esta concepción de la protección puede ser extendida fácilmente para obtener una protección más detallada. Podemos disponer de hardware que ofrezca protección de sólo lectura, lectura-escritura o sólo ejecución. O bien, por medio de bits de protección independientes para cada tipo de acceso, puede permitirse cualquier combinación de estos accesos, al tiempo que las tentativas ilegales generan una excepción al sistema operativo.

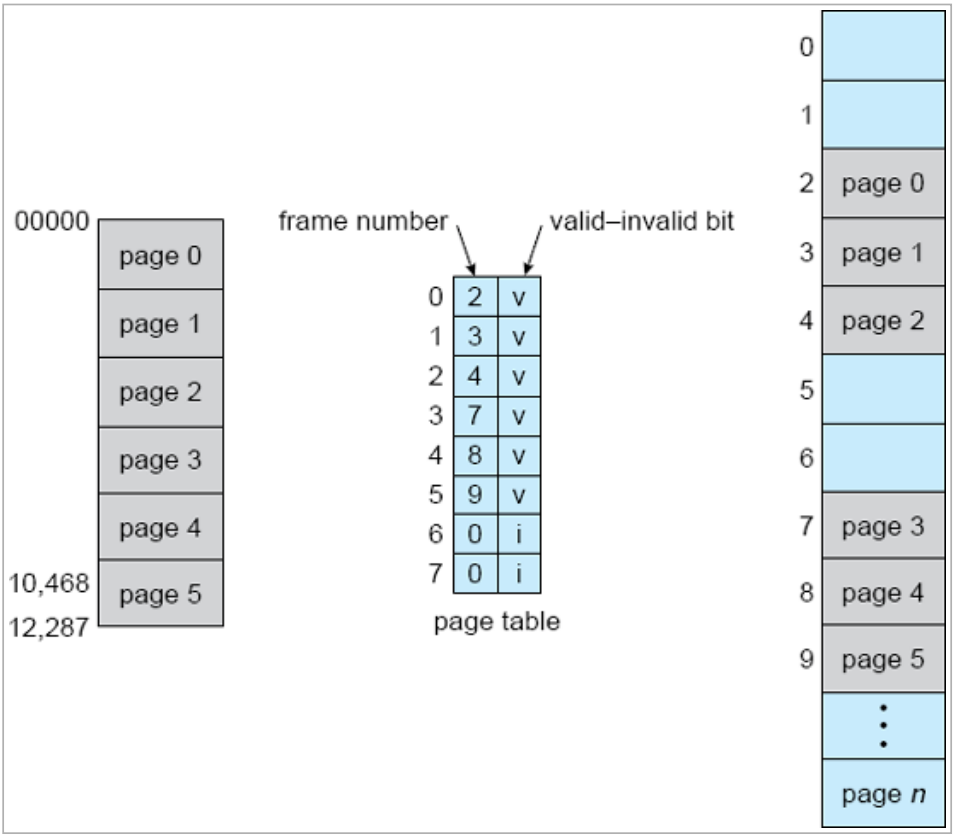


Figura 13. Protección en la paginación.

6.3.4 Compartir Páginas

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Otra ventaja de la paginación es la posibilidad de compartir programas de uso corriente. Esto es particularmente importante en un entorno de tiempo compartido.

Consideremos un sistema que soporta 40 usuarios, cada uno de los cuales ejecuta un editor de textos. Si el editor de textos consta de 150K de código y 50K de espacio para datos, necesitaríamos 8000K para permitir a los 40 usuarios.
No obstante, si el **programa es reentrante**, podría compartirse como se muestra en la Figura 13. Aquí vemos un editor de tres páginas que es compartido por tres procesos. Cada proceso tiene su propia página de datos.

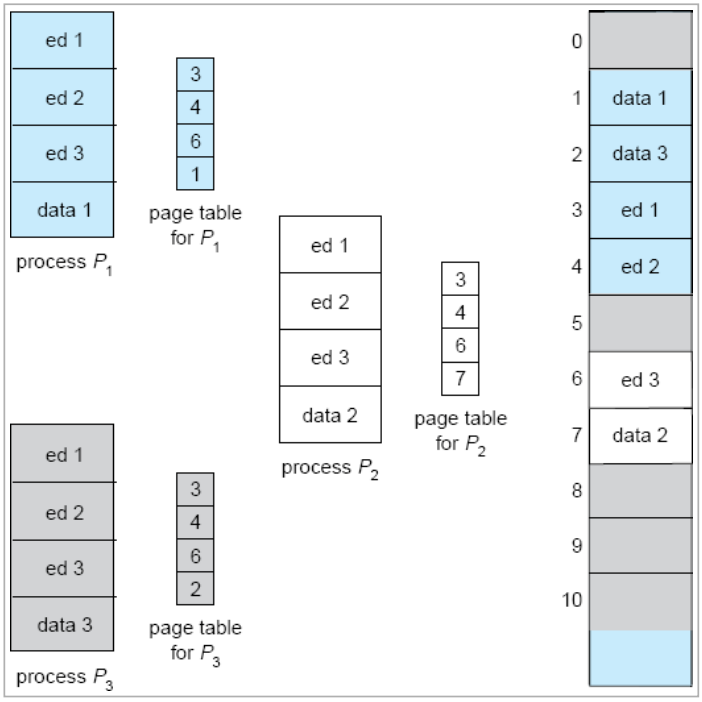


Figura 14. Compartir en un esquema de paginación.
El **código reentrante** (también llamado código puro) es un código no automodificable. Si el código es reentrante, entonces nunca cambia durante la ejecución. Así, dos o más procesos pueden ejecutar el mismo código al mismo tiempo. Cada proceso, para su ejecución, tiene su PCB y su memoria para mantener los datos. Por supuesto, los datos de todos esos procesos diferentes varían para cada uno de ellos.
Tan sólo hace falta mantener una copia del editor en la memoria física. Cada tabla de páginas de usuario-proceso hace referencia a la misma copia física del editor, pero las páginas de datos lo hacen a marcos diferentes.
Así, para permitir 40 usuarios, precisamos solamente una copia del editor, 150K, más 40 copias del espacio de 50K por usuario. El espacio total requerido es ahora de 2150K, en lugar de 8000K, un ahorro significativo.
También pueden compartirse otros programas muy utilizados: compiladores, ensambladores, sistemas de bases de datos, etc.
Para que sea compartible, el código tiene que ser reentrante (no automodificable). Este término significa que nunca debería darse una tentativa de almacenar algo en el código, que es de sólo búsqueda o sólo lectura. Obviamente, es crucial que las páginas compartidas sean inamovibles. Si un usuario cambiara una posición, cambiaría para todos los usuarios. La naturaleza de sólo lectura del código compartido no debería dejarse a merced de la corrección del código. El sistema operativo debe reforzar esa propiedad.

6.4 Segmentación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
Un aspecto importante de la gestión de la memoria que la paginación convierte en inevitable es la separación de la visión que el usuario tiene de la memoria de la memoria física real. La visión del usuario no coincide con la memoria física real. La visión del usuario se transforma en la memoria física. La traducción de direcciones permite esta diferencia entre la memoria lógica y la física.
¿Cuál es la visión de la memoria que tiene el usuario? ¿concibe el usuario la memoria como una tabla lineal de palabras, algunas de las cuales contienen instrucciones mientras que otras contienen datos, o bien se prefiere alguna otra visión de la memoria?
Hay un acuerdo general en que el usuario o programador de un sistema no piensa en la memoria como una tabla lineal de palabras. Más bien prefiere concebirla como una colección de segmentos de longitud variable, no necesariamente ordenados (Figura 15).

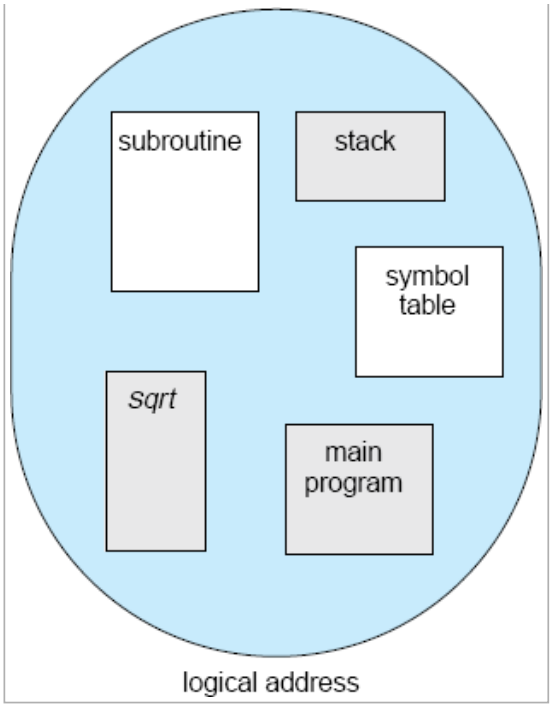


Figura 15. Espacio de direcciones lógicas.
Piensa en un programa cuando lo estás escribiendo: un programa principal, con un conjunto de subrutinas, procedimientos, funciones o módulos. También puede haber diversas estructuras de datos: tablas, matrices, pilas, variables, etc. Cada uno de estos módulos o elementos de datos se referencian por un nombre. Se habla de la "tabla de símbolos", "la función Sqrt", "el programa principal", sin tener en cuenta qué direcciones de memoria ocupan estos elementos. No se sabe si la tabla de símbolos se almacena antes o después de la función Sqrt. Cada uno de estos elementos es de longitud variable; la longitud está definida intrínsecamente por el propósito del segmento en el programa. Los elementos dentro de un segmento están identificados por su desplazamiento desde el principio del segmento: la primera instrucción del programa, la decimoséptima entrada de la tabla de símbolos, la quinta función Sqrt, etc.
La **segmentación** es un esquema de administración de la memoria que permite la visión que el usuario tiene de la misma. Un espacio de direcciones lógicas es una colección de segmentos. Cada segmento tiene un nombre (un número) y una longitud. Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro del segmento. Por lo tanto, el usuario especifica cada dirección mediante dos cantidades: un nombre de segmento y un desplazamiento.
En el esquema paginado, el usuario especificaba solamente una única dirección, que el hardware particionaba en número de página y desplazamiento, siendo todo ello invisible al programador.
Por simplicidad de implementación, los segmentos están numerados y se referencian por un número de segmento en lugar de por un nombre. Normalmente el programa de usuario se ensambla (o compila), y el ensamblador (o el compilador) construye automáticamente segmentos que reflejan el programa de entrada.

> Un compilador de C podría crear segmentos separados para:
>
> 1. El código.
> 2. Las variables globales.
> 3. Las variables locales.
> 4. Las pilas para cada hebra, de llamada a procedimientos, para almacenar parámetros y devolver direcciones.
> 5. La biblioteca C estándar.
>
> También segmentos separados para la bibliotecas que se montan en tiempo de compilación. El cargador tomará todos esos segmentos y les asignará números de segmento.

6.4.1 Hardware

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
Aunque el usuario ahora puede referenciar los objetos del programa por medio de una dirección de dos dimensiones, la memoria física real es todavía, por supuesto, una secuencia unidimensional de palabras. La transformación se efectúa por medio de una tabla de segmentos.
El empleo de una tabla de segmentos se muestra en la Figura 15. Una dirección lógica consta de dos partes: **un número de segmento s** y un **desplazamiento dentro de ese segmento, d**. El número de segmento se utiliza como un índice en la tabla de segmentos. Cada entrada de la tabla de segmentos tiene una **base de segmento** y un **límite**. El desplazamiento d de la dirección lógica tiene que estar comprendido entre 0 y el límite de segmento. En caso contrario se produce una excepción al sistema operativo (tentativa de direccionamiento lógico más allá del fin del segmento). Si este desplazamiento es legal, se añade a la base para producir la dirección de la tabla deseada en la memoria física. La tabla de segmentos es así esencialmente una matriz de pares registros base/límite.

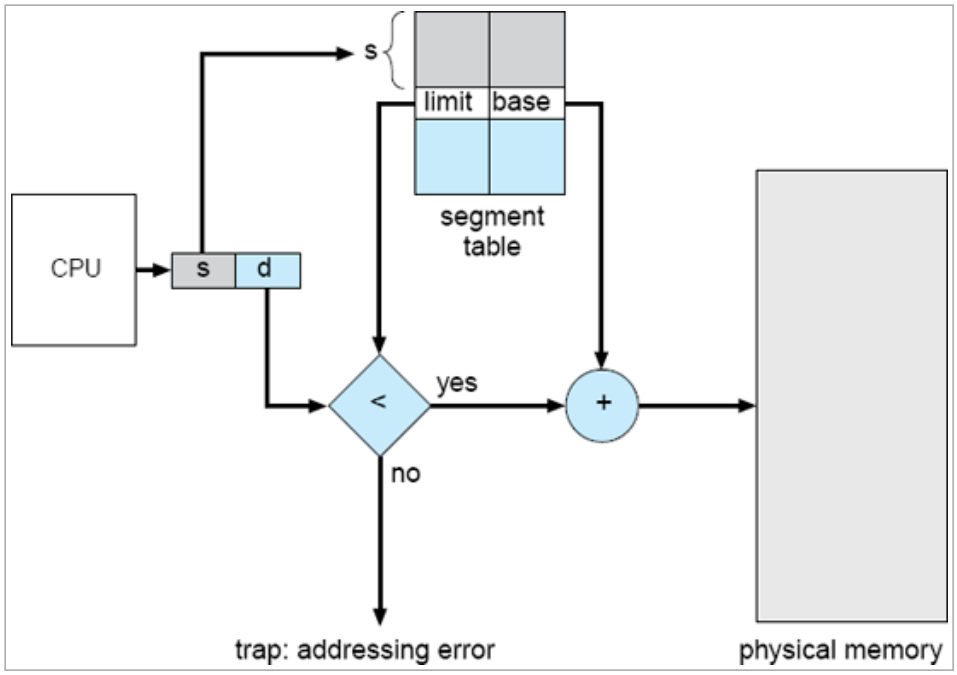


Figura 15. Hardware de segmentación

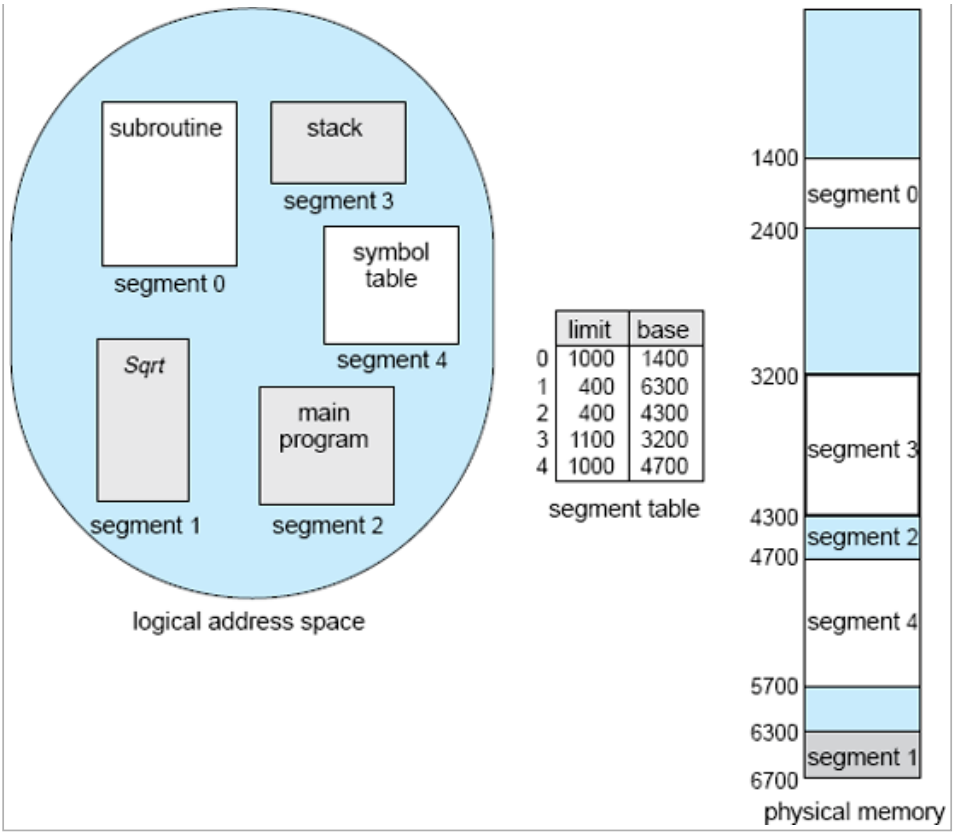
Al igual que la tabla de páginas, la tabla de segmentos puede situarse bien en registros rápidos o bien en memoria. Una tabla de segmentos mantenida en registros puede ser referenciada muy rápidamente: la adición a la base y la comparación con el límite pueden realizarse simultáneamente para ahorrar tiempo.

El PDP-11/45 utiliza este método; tiene 8 registros de segmento. Una dirección de 16 bits se forma a partir de un número de segmento de 3 bits y de un desplazamiento de 13 bits. Esta disposición permite hasta 8 segmentos; cada segmento puede ser de hasta 8 K-bytes.

Cada entrada en la tabla de segmentos tiene una dirección base, una longitud y un conjunto de bits de control de acceso que especifican acceso denegado, acceso de sólo lectura, o acceso de lectura/escritura al segmento.

Con muchos segmentos no es factible mantener la tabla de segmentos en registros, de modo que tiene que mantenerse en memoria. Un registro de base de tabla de segmentos (STBR) apunta a la tabla de segmentos. Puesto que el número de segmentos utilizado por un programa puede variar ampliamente, también se utiliza un registro de longitud de tabla de segmentos (STLR). En el caso de una dirección lógica (s, d) verificamos primero que el número de segmento s es legal (s < STLR), Entonces, añadimos el número de segmento al STBR resultando la dirección en memoria de la entrada de la tabla de segmentos (STBR + s). Esta entrada se lee en la memoria y actuamos igual que antes: se verifica el desplazamiento frente a la longitud de segmento, y se calcula la dirección física de la palabra deseada como la suma de la base del segmento y el desplazamiento.

Igual que con la paginación, esta transformación requiere dos referencias a memoria por dirección lógica, el ordenador disminuirá su velocidad en un factor de 2, a menos que se haga algo para evitarlo. La solución normal consiste en utilizar un conjunto de registros asociativos para mantener las entradas utilizadas más recientemente en la tabla de segmentos. Un conjunto de registros asociativos relativamente pequeño (8 / 16) puede reducir generalmente el retardo a los accesos a memoria hasta no más allá de un 10% o 15% más lentos que los accesos a memoria “mapeada”.



6.4.2 Compartir y Proteger

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

Una ventaja importante de la segmentación es la asociación de la **protección con los segmentos**. Puesto que los segmentos representan una **porción del programa definida semánticamente**, es probable que todas las entradas en el segmento se utilicen de la misma manera.

De ahí que tengamos algunos segmentos que son instrucciones, mientras que otros son datos. En una arquitectura moderna las instrucciones son no automodificables, de modo que los segmentos de instrucciones pueden definirse como de sólo lectura o sólo ejecución.

El **hardware verificará los bits de protección** asociados a cada entrada en la tabla de segmentos para impedir accesos ilegales a memoria, tales como tentativas de escribir en un segmento de sólo lectura o de utilizar un segmento de sólo ejecución como datos. Situando una tabla en un segmento propio, el hardware verificará automáticamente que toda indexación en la tabla es legal, y no sobrepasa los límites de la misma. Así, muchos errores frecuentes en programas serán detectados por hardware antes de que puedan ocasionar un daño serio.

Otra ventaja de la segmentación está relacionada con la **compartición** de código y datos. Los segmentos se comparten cuando las entradas en las tablas de segmentos de dos procesos diferentes apuntan a las mismas posiciones físicas.
La compartición se produce a nivel de segmento. Por lo tanto, cualquier información puede compartirse definiéndole un segmento. Pueden compartirse varios segmentos, de modo que es posible compartir un programa compuesto de más de un segmento.

> Por ejemplo, consideremos el uso de un editor de textos en un sistema de tiempo compartido. Un editor completo podría resultar bastante largo, y formado por muchos segmentos. Estos segmentos pueden compartirse entre todos los usuarios, limitando la memoria física necesaria para permitir las tareas de edición. En lugar de necesitar n copias del editor, precisamos solamente una. Aún necesitamos segmentos únicos e independientes para almacenar las variables locales de cada usuario. Estos segmentos, por supuesto, no deben ser compartidos.

> También es posible compartir solo partes de programas. Por ejemplo, subrutinas de uso frecuente pueden compartirse entre muchos usuarios definiéndolas como segmentos de sólo lectura compartibles. Por ejemplo, dos programas Fortran pueden utilizar la misma subrutina Sqrt, pero sólo será precisa una copia física de la rutina Sqrt.

Aunque esta compartición parece ser bastante sencilla, tiene algunas sutilezas. Típicamente, los segmentos de código tienen referencias a sí mismos. Por ejemplo, un salto condicional tiene normalmente una dirección de transferencia. La dirección de transferencia es un nombre de segmento y un desplazamiento. El número de segmento de la dirección de transferencia será el del segmento de código. Si tratamos de compartir este segmento, todos los procesos que lo compartan tienen que definir el segmento de código compartido con el mismo número de segmento.

> Por ejemplo, si queremos compartir la rutina Sqrt y un proceso quiere definirla como segmento 4 y otro lo hace como segmento 17, ¿cómo podría la subrutina Sqrt referenciarse a sí misma? Puesto que solamente hay una copia física de Sqrt, tiene que referenciarse a sí misma de la misma manera para ambos usuarios: tiene que tener un número de segmento único. A medida que crece el número de usuarios que comparten el segmento, también crece la dificultad de encontrar un número de segmento aceptable.

Los segmentos de datos de sólo lectura (sin punteros) pueden compartirse aún usando números de segmento diferentes; lo mismo puede hacerse con segmentos de código que no se referencian directamente a sí mismos, sino sólo indirectamente. Por ejemplo, la bifurcación condicional que especifica la dirección de desplazamiento a partir del valor actual del contador de programa o respecto a un registro que contiene el número de segmento actual, permite que el código no tenga que realizar una referencia al número de segmento actual.
El ordenador GE 645 utilizado con Multics tenía 4 registros que contenían los números de segmento del segmento actual, del segmento de pila, del segmento de enlace y de un segmento de datos. Los programas pocas veces hacen referencia directamente a un número de segmento, sino siempre indirectamente a través de estos cuatro registros de segmento. Esto permite que el código pueda compartirse libremente.

6.4.3 Fragmentación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

El sistema operativo tiene que encontrar y asignar memoria para todos los segmentos de un programa de usuario. Esta situación es similar a la paginación, excepto en el hecho de que los segmentos son de longitud variable; las páginas son todas del mismo tamaño. Por tanto, como en el caso de las particiones dinámicas, la asignación de memoria es un problema de **asignación dinámica de almacenamiento**, resuelto probablemente mediante un **algoritmo del mejor o primer ajuste**.

La segmentación puede ocasionar entonces **fragmentación externa**, cuando todos los bloques libres de memoria son demasiado pequeños para acomodar a un segmento. En este caso, el proceso puede simplemente verse obligado a esperar hasta que haya disponible más memoria (o al menos huecos más grandes), o puede utilizarse la compactación para crear huecos mayores. Puesto que la segmentación es por naturaleza un algoritmo de reubicación dinámica, podemos compactar la memoria siempre que queramos.
¿En qué medida es mala la fragmentación externa en un esquema de segmentación? La respuesta a estas preguntas depende principalmente del tamaño medio de segmento.

- En un extremo, se podría definir cada proceso como un segmento; este esquema es el de las particiones dinámicas.
- En el otro extremo, cada palabra podría situarse en su propio segmento y reubicarse por separado. Esta disposición elimina la fragmentación externa. Si el tamaño medio de segmento es pequeño, la fragmentación externa también será pequeña.

(Por analogía, consideremos la colocación de las maletas en el maletero de un coche; parece que nunca encajan bien. Sin embargo, si se abren las maletas y se colocan en el maletero los objetos sueltos, todo encaja). Puesto que los segmentos individuales son más pequeños que el proceso en conjunto, es más probable que encajen en los bloques de memoria disponibles.

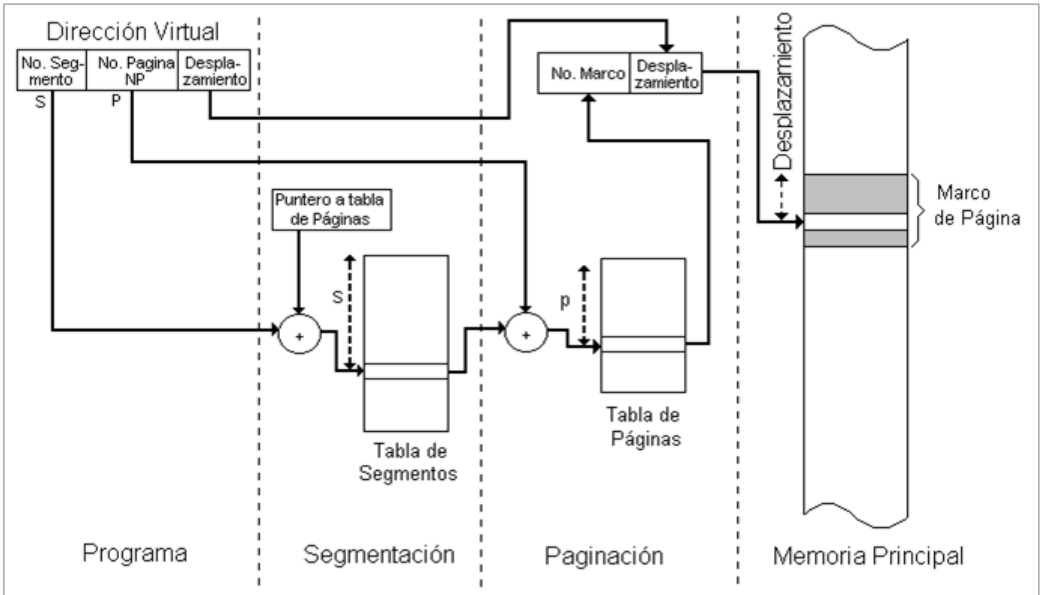
6.5 Segmentación Paginada

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria
Tanto la paginación como la segmentación tienen sus ventajas y desventajas. También es posible combinar estos dos esquemas para mejorar ambos.

Veamos como ejemplo el esquema del ordenador GE 645 con el sistema operativo Multics. Las direcciones lógicas estaban formadas a partir de un número de segmento de 18 bits y un desplazamiento de 16 bits. Aunque este esquema crea un espacio de direcciones correspondiente a una dirección de 34 bits, la tabla de segmentos tiene un tamaño tolerable, puesto que el número variable de segmentos conduce naturalmente al uso de un Registro de Longitud de Tabla de Segmentos. Necesitamos tan solo el mismo número de entradas en la tabla de segmentos que segmentos; no tenemos por qué tener entradas vacías en la tabla de segmentos.

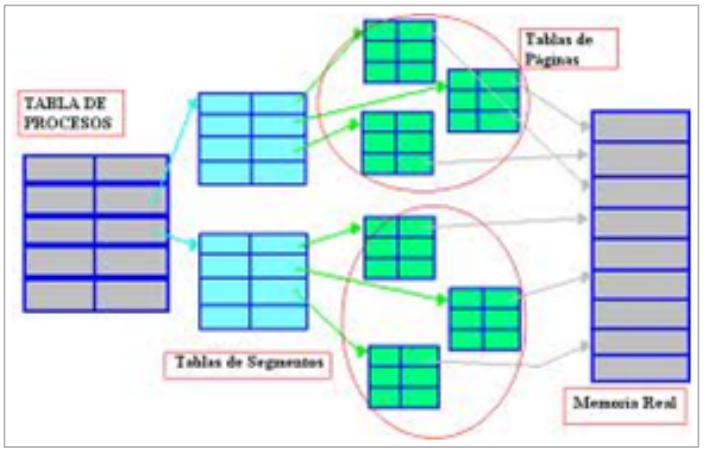
No obstante, con segmentos de 64 K-palabras, el tamaño medio de segmento podría resultar bastante grande y la fragmentación externa constituir un problema. Incluso si la fragmentación externa no es significativa, el tiempo de búsqueda para asignar un segmento, utilizando un primer o mejor ajuste, podría ser grande. De esta manera se podría desperdiciar memoria a causa de la fragmentación externa o bien desperdiciar tiempo debido a la búsqueda larga, o bien ambas cosas.
La solución adoptada fue paginar los segmentos.

- La paginación **elimina la fragmentación externa** y convierte en **trivial el problema de la asignación**: cualquier marco vacío puede utilizarse para una página.
- Obsérvese que la diferencia entre esta solución y la segmentación pura es que la entrada en la tabla de segmentos no contiene la dirección de la base del segmento, sino la dirección de la base de una tabla de páginas para ese segmento.
- El desplazamiento del segmento se fragmenta entonces en un número de página de 6 bits y un desplazamiento de página de 10 bits. El número de página indexa en la tabla de páginas para dar el número de marco. Finalmente, el número de marco se combina con el desplazamiento de página para formar la dirección física.



Traducción de dir ección en la Segmentación P aginada.

Ahora debemos tener **una tabla de páginas independiente para cada segmento** de cada proceso. No obstante, puesto que cada segmento tiene una longitud limitada por su entrada en la tabla de segmentos, la tabla de páginas no tiene por qué tener su tamaño máximo. Sólo precisa tantas entradas como se necesiten realmente. Además, generalmente la última página de cada segmento no estará totalmente llena. De este modo tendremos, por término medio, **media página de fragmentación interna por segmento** del proceso. Consecuentemente, aunque hemos eliminado la fragmentación externa, hemos introducido fragmentación interna e incrementado la sobrecarga de espacio de la tabla.



A decir verdad, incluso la visión de paginación segmentada de Multics que acabamos de presentar es simplista. Puesto que el número de segmento es una cantidad de 18 bits, podríamos tener 262144 segmentos, con lo que precisaríamos una tabla de segmentos muy larga. Para simplificar este problema, Multics pagina la tabla de segmentos. De esta manera, en general, una dirección en Multics utiliza un número de segmento para definir un índice de página en una tabla de páginas para la tabla de segmentos. A partir de esta entrada, localiza la parte de la tabla de segmentos que tiene la entrada para ese segmento. La entrada en la tabla de segmentos apunta a una tabla de páginas para ese segmento, que especifica el marco que contiene la palabra deseada.

6.6 Memoria Comprimida

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
Bloque Memoria: Gestión de Memoria

La Memoria Comprimida es una nueva estrategia que intenta evitar el traslado de los procesos a disco, el uso de la memoria virtual. Es, por tanto, una buena alternativa a la memoria virtual.

Para ello, comprime las páginas de los procesos que están bloqueados o que llevan mucho tiempo sin ejecutarse.

Por ejemplo, supongamos la siguiente situación, se tiene un proceso 1 bloqueado y un proceso 2 listo, de forma que entre ambos ocupan toda la memoria disponible. Si llegase un nuevo proceso 3, no habría espacio suficiente para ejecutarlo.

La solución clásica consistiría en mover algunas de las páginas del proceso 1 al disco, para dejar espacio libre para el nuevo proceso.