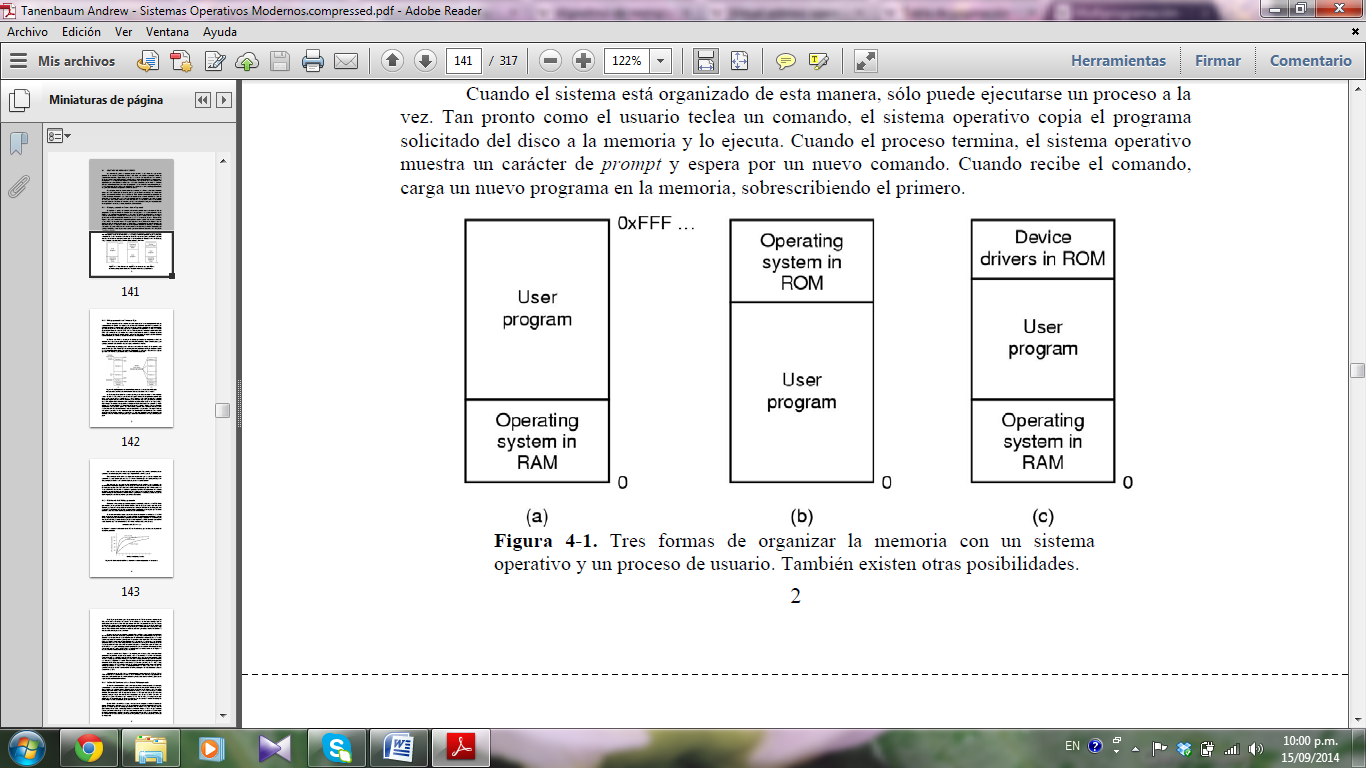
**Resumen Administración de Memoria**

**ASIGNACIÓN CONTIGUA DE LA MEMORIA**

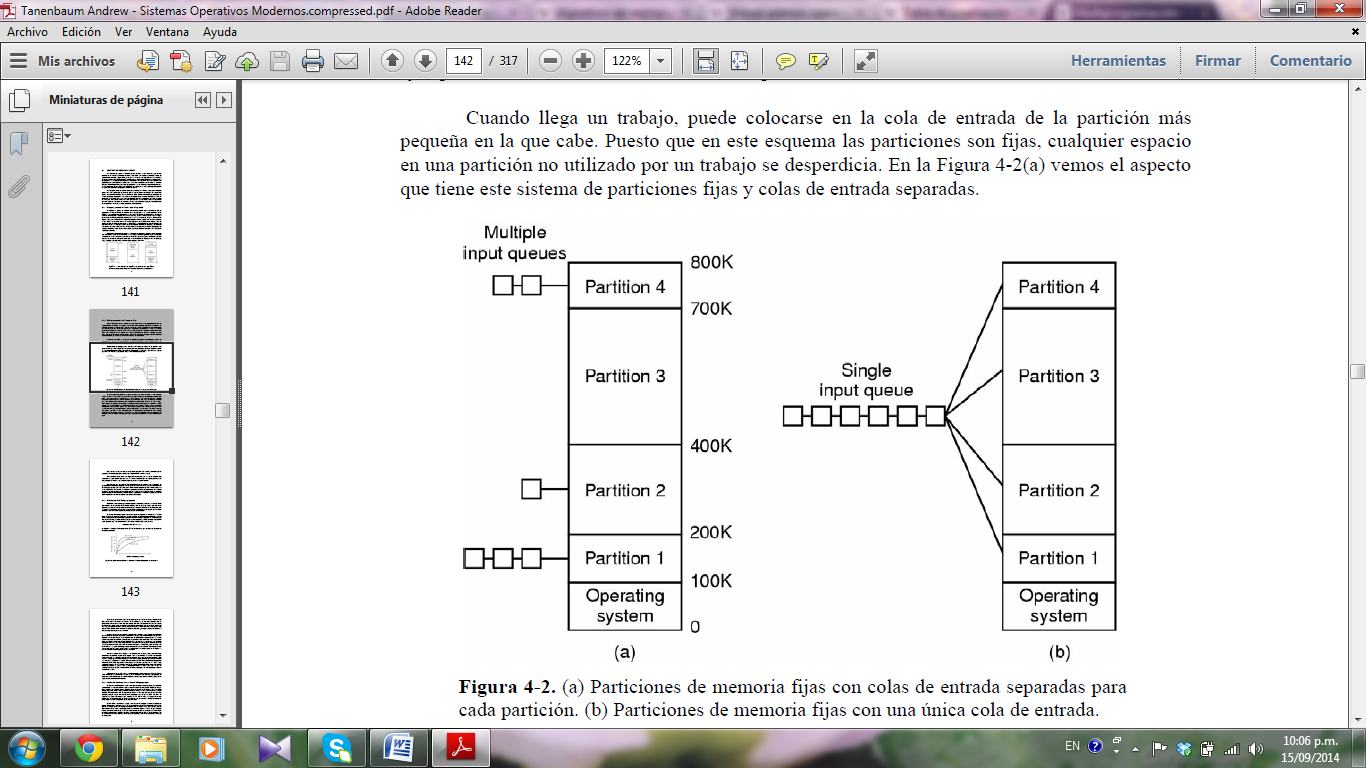
**Monoprogramación**

Consiste en ejecutar sólo un programa a la vez, repartiendo la memoria entre ese programa y el sistema operativo.

Tres formas de organizar la memoria con un sistema operativo y un proceso de usuario. También existen otras posibilidades.

**Multiprogramación con Particionamiento Estático**

El tener múltiples procesos ejecutándose a la vez significa que cuando un proceso se bloquea esperando a que termine una operación de E/S, otro proceso puede seguir haciendo uso de la CPU. Así la multiprogramación aumenta la utilización de la CPU.

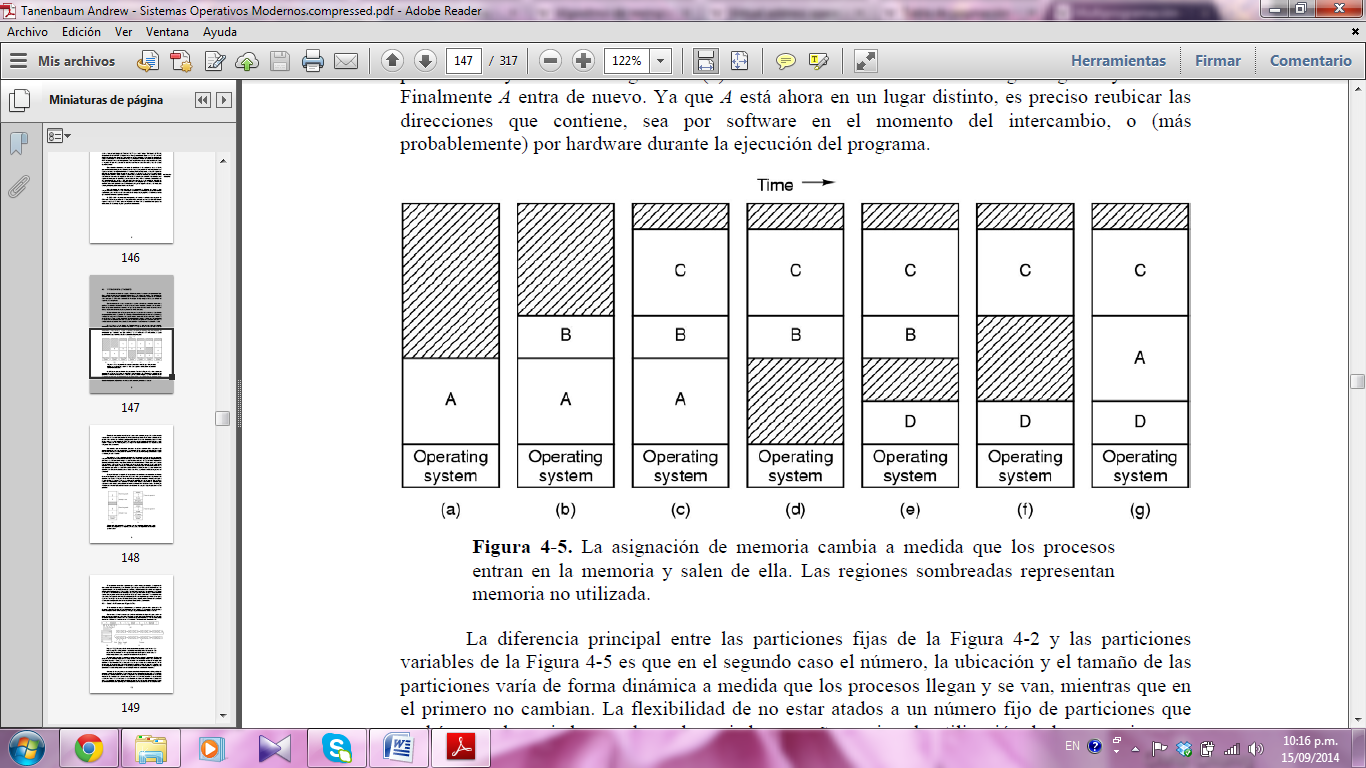
La forma más fácil de conseguir la multiprogramación es simplemente dividir la memoria en *n* particiones (posiblemente de diferentes tamaños). Esta división puede, por ejemplo, realizarse manualmente cuando se pone en marcha el sistema. Cuando llega un trabajo, puede colocarse en la cola de entrada de la partición más pequeña en la que cabe. Puesto que en este esquema las particiones son fijas, cualquier espacio en una partición no utilizado por un trabajo se desperdicia.

(a) Particiones de memoria fijas con colas de entrada separadas para cada partición.

(b) Particiones de memoria fijas con una única cola de entrada.

**Intercambio (Swapping)**

Consiste en cargar en la memoria un proceso entero, ejecutarlo durante un rato y volver a guardarlo en el disco.



La asignación de memoria cambia a medida que los procesos entran en la memoria y salen de ella. Las regiones sombreadas representan memoria no utilizada.

**Reubicación Dinámica**

Consiste en equipar la máquina con dos registros especiales de hardware, llamados el registro de base y el registro de límite. Cuando se planifica un proceso, se carga el registro de base con la dirección donde comienza su partición, y el registro de límite se carga con la longitud de la partición. Cada vez que se genera una dirección de memoria, se le suma de forma automática el contenido del registro de base antes de enviarla a la memoria. Por ejemplo, si el registro base contiene el valor 100K, una instrucción CALL 100 se convierte efectivamente en una instrucción CALL 100K + 100, sin que la instrucción en sí se modifique. También se comparan las direcciones con el registro de límite para asegurar que no intentan direccionar memoria fuera de la partición actual. El hardware protege los registros de base y de límite para evitar que los programas de usuario los modifiquen.

**Particionamiento Estático**

**Compactación**

Es cuando el intercambio crea múltiples huecos en la memoria, se combinan todos esos huecos en uno solo más grande, moviendo todos los procesos hacia abajo hasta donde sea posible.

**Compartición de Datos**

**Programas**

**ASIGNACIÓN NO CONTIGUA DE LA MEMORIA**

**Concepto y Gestión de Memoria Virtual**

**Página**

Pequeña parte de un proceso.

**Marco de Página**

Espacio definido de la memoria, y posee el mismo tamaño de una página.

**Paginación**

Dividen los programas en pequeñas partes o páginas. Del mismo modo, la memoria es dividida en trozos del mismo tamaño que las páginas llamados marcos de página.

**Tabla de Paginación**

Es una parte integral del [Sistema de Memoria Virtual](http://es.wikipedia.org/wiki/VMS), cuando se utiliza [paginación](http://es.wikipedia.org/wiki/Paginaci%C3%B3n). Son usadas para realizar las traducciones de direcciones de memoria virtual a memoria física y en general el sistema operativo mantiene una por cada proceso corriendo en el sistema.

**Falta de Página**

Es cuando la página no tiene correspondencia en la memoria física y la MMU provoca una excepción que hace que la CPU ceda el control al sistema operativo.

**Algoritmos de Reemplazo de Páginas**

* **Algoritmo Óptimo**

El algoritmo óptimo reemplaza la página que tardará más tiempo en volver a ser referenciada. Desafortunadamente, no hay forma de determinar qué página será la que tarde más tiempo en referenciarse, por lo que este algoritmo no puede utilizarse en la práctica. Sin embargo, es útil como base de comparación con otros algoritmos en un banco de pruebas.

* **Algoritmo NRU (Not Rencently Used)**

El algoritmo NRU divide las páginas en cuatro clases según el estado de los bits *R* y *M*. Se escoge una página al azar de la clase de número más bajo. Este algoritmo es fácil de implementar, pero es muy burdo. Los hay mejores.

* **Algoritmo FIFO (First In, First Out)**

El algoritmo FIFO recuerda el orden en el que se cargaron las páginas en la memoria, utilizando una lista enlazada. Resulta trivial eliminar la página más antigua, pero es posible que esa página todavía esté en uso, por lo que FIFO es una mala elección.

* **Algoritmo de la Segunda Oportunidad**

El algoritmo de la segunda oportunidad es una modificación del FIFO que comprueba si una página se está usando o no, antes de sustituirla. Si se está usando, se le perdona la vida. Esta modificación mejora el rendimiento de forma considerable.

* **Algoritmo del Reloj**

El algoritmo del reloj es simplemente una implementación diferente del algoritmo de la segunda oportunidad. Tiene las mismas propiedades de rendimiento, pero la ejecución del algoritmo es más rápida.

* **Algoritmo LRU (Least Rencently Used)**

LRU es un algoritmo excelente, pero no puede implementarse sin un hardware especial. Si no se cuenta con ese hardware, LRU no puede usarse.

* **Algoritmo NFU (Not Frequently Used)**

NFU es un intento burdo de aproximarse al LRU. No es muy bueno.

* **Algoritmo de Envejecimiento (Aging)**

El algoritmo de envejecimiento es una aproximación mucho mejor al LRU y puede implementarse eficientemente. Es una buena elección.

* **Algoritmo WSClock**

El algoritmo WSClock es una variante que no solo tiene un buen rendimiento, sino que también puede implementarse eficientemente.

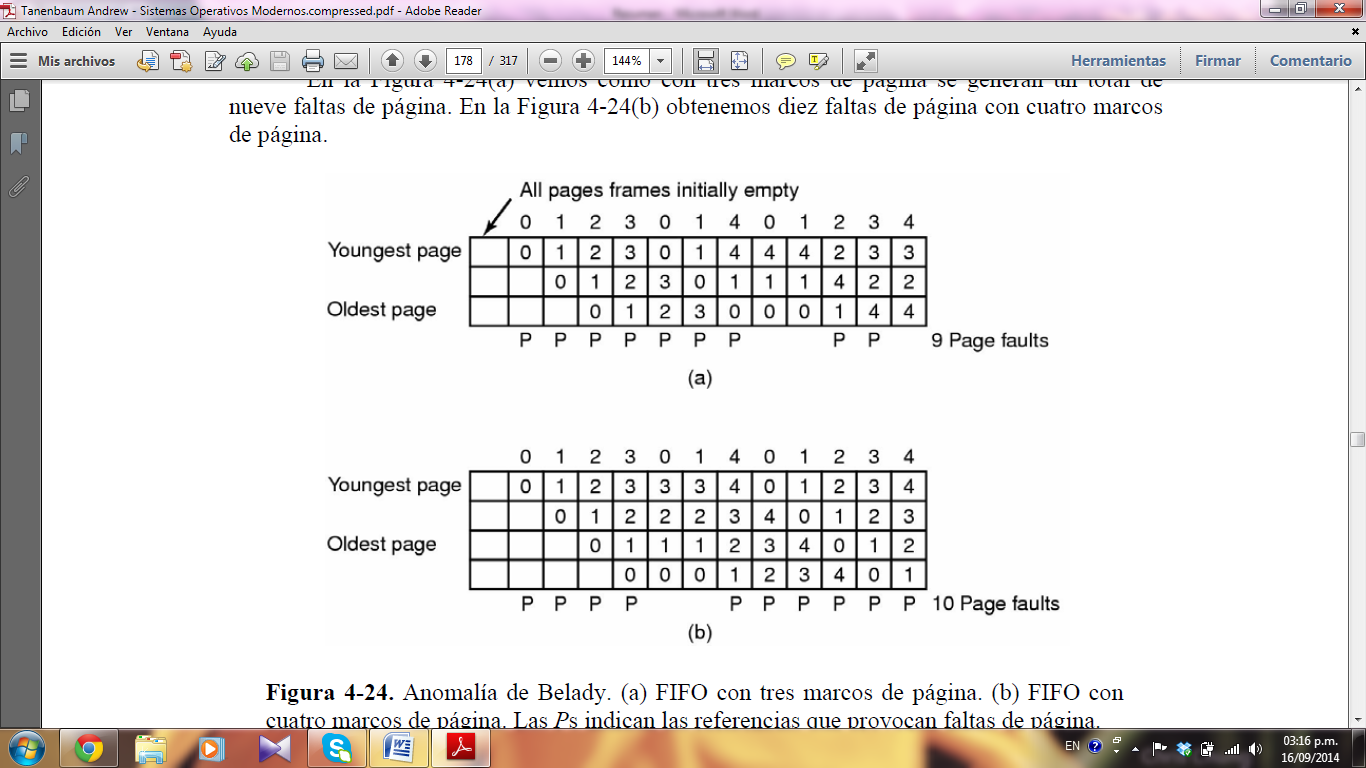
**Modelación de Algoritmos de Paginación**

* **Anomalía de Belady**

Intuitivamente, podríamos pensar que cuanto más marcos de página tenga la memoria, menos faltas de página experimentará un programa. Aunque parezca bastante sorprendente, no siempre sucede así. Belady y otros (1969) descubrieron un contraejemplo en el que FIFO provocaba más faltas de página con cuatro marcos de página que con tres. Esta situación se conoce como la **anomalía de Belady**, y se ilustra en la siguiente Figura para un programa con cinco páginas virtuales, numeradas del 0 al 4. Las páginas se referencian en el orden

0 1 2 3 0 1 4 0 1 2 3 4

En la Figura (a) vemos como con tres marcos de página se generan un total de nueve faltas de página. En la Figura (b) obtenemos diez faltas de página con cuatro marcos de página.



(a) FIFO con tres marcos de página.

(b) FIFO con cuatro marcos de página. Las *P*s indican las referencias que provocan faltas de página.

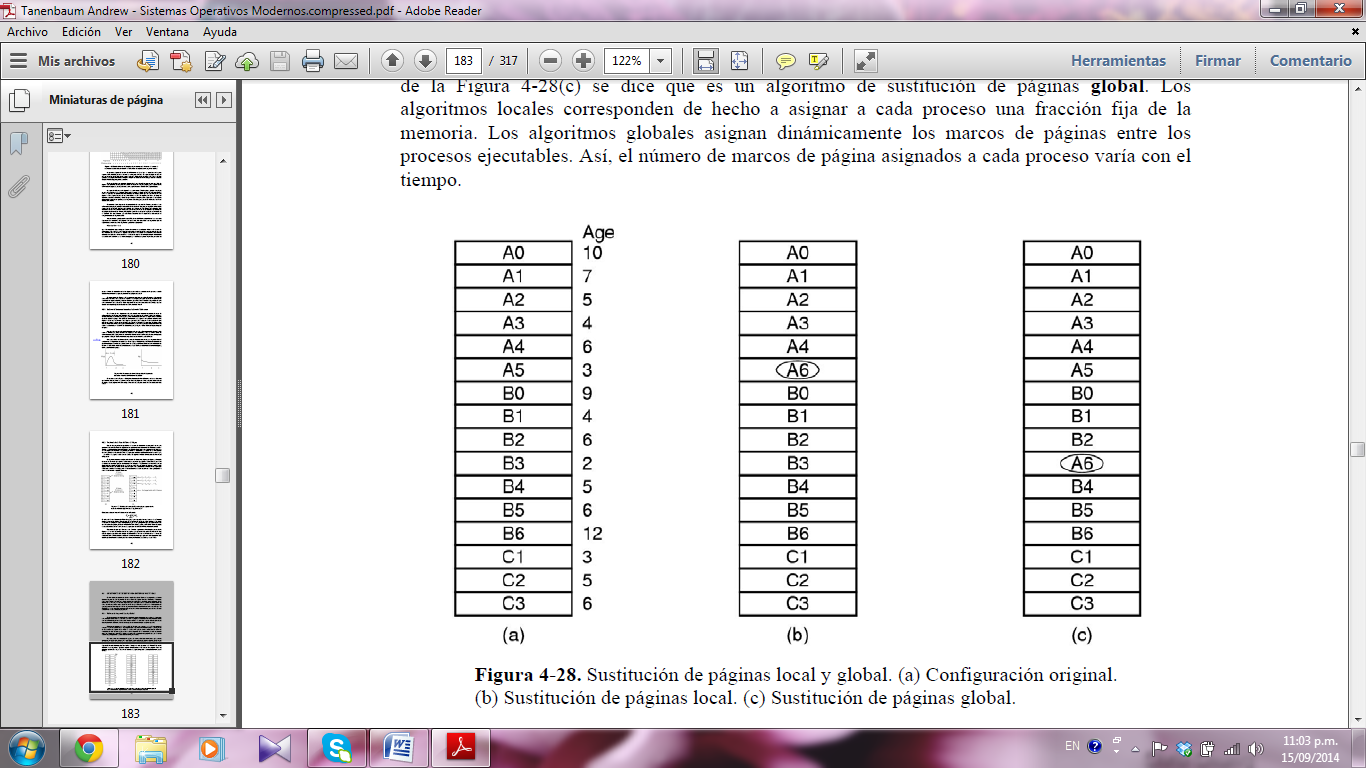
* **Algoritmos de Pila**

Los algoritmos de pila presentan la propiedad de que, dada una secuencia de referencias, una memoria con más marcos de página, tras un número determinado de referencias, mantiene el conjunto de páginas que contendría con un tamaño menor en la misma referencia. En otras palabras, la tasa de fallos nunca aumenta si se incrementa el tamaño de memoria. Los algoritmos de pila evitan la anomalía de Belady.

* **La Serie de Distancias Asociada a la Serie de Referencias**
* **Predicción de la Tasa de Faltas de Página**

**Aspectos de Diseño e Implantación para los Sistemas de Paginación**

* **Políticas de Asignación Local y Global**

Los algoritmos locales corresponden de hecho a asignar a cada proceso una fracción fija de la memoria. Los algoritmos globales asignan dinámicamente los marcos de páginas entre los procesos ejecutables. Así, el número de marcos de página asignados a cada proceso varía con el tiempo.

(a) Configuración original. (b) Sustitución de páginas local. (c) Sustitución de páginas global.

* **Control de Carga**
* **Tamaño de Página**

Suele ser un parámetro que puede elegir el sistema operativo. Determinar el tamaño de página óptimo requiere equilibrar varios factores en conflicto. Como resultado, no existe un tamaño óptimo para todos los casos. Para comenzar, hay dos factores que favorecen el uso de páginas pequeñas. Un segmento de código, datos o pila cualquiera no tiene porqué llenar completamente un número entero de páginas. En promedio, la mitad de la última página estará vacía, y ese espacio adicional se desperdicia. Este desperdicio se denomina **fragmentación interna**. Si hay *n* segmentos en la memoria y las páginas son de *p* bytes, se desperdiciarán *np*/2 bytes debido a la fragmentación interna. Este razonamiento nos lleva a escoger páginas pequeñas.

**Segmentación**

La segmentación permite al programador contemplar la memoria como si constara de varios espacios de direcciones o segmentos. Con memoria virtual, el programador no necesita preocuparse de las limitaciones de memoria impuestas por la memoria principal. Los segmentos pueden ser de distintos tamaños, incluso de forma dinámica. Las referencias a memoria constan de una dirección de la forma (número de segmento, desplazamiento). Esta organización ofrece al programador varias ventajas sobre un espacio de direcciones no segmentado:

1. Simplifica el manejo de estructuras de datos crecientes. Si el programador no conoce *a priori* cuan larga puede llegar a ser una estructura de datos determinada, es necesario suponerlo amenos que se permitan tamaños de segmento dinámicos. Con memoria virtual segmentada, acada estructura de datos se le puede asignar a su propio segmento y el sistema operativo expandirá o reducirá el segmento cuando se necesite. Si un segmento necesita expandirse en memoria y no dispone de suficiente sitio, el sistema operativo puede mover el segmento a un área mayor de la memoria principal, si la hay disponible, o descargarlo. En este último caso, el segmento agrandado será devuelto a memoria en la siguiente ocasión.

2. Permite modificar y recompilar los programas independientemente, sin que sea necesario recompilar o volver a montar el conjunto de programas por completo. Esto se puede llevar nuevamente a cabo gracias al uso de varios segmentos.

3. Se presta a la compartición entre procesos. Un programador puede situar un programa de utilidades o una tabla de datos en un segmento que pueda ser referenciado por otros procesos.

4. Se presta a la protección. Puesto que un segmento puede ser construido para albergar un conjunto de procedimientos y datos bien definido, el programador o el administrador del sistema podrá asignar los permisos de acceso de la forma adecuada.



Formatos de Gestión de Memoria

**Segmentación Paginada**

La paginación, que es transparente al programador, elimina la fragmentación externa y, de este modo, aprovecha la memoria principal de forma eficiente. Además, puesto que los fragmentos que se cargan y descargan de memoria principal son de tamaño constante e igual para todos, es posible construir algoritmos de gestión de memoria sofisticados que aprovechen mejor el comportamiento de los programas, tal y como se verá. La segmentación, que es visible para el programador, tiene las ventajas antes citadas, incluida la capacidad de manejar estructuras de datos que puedan crecer, la modularidad y el soporte de la compartición y la protección. Para combinar las ventajas de ambas, algunos sistemas están equipados con hardware del procesador y software del sistema operativo que las permiten. En un sistema con paginación y segmentación combinadas, el espacio de direcciones de un usuario se divide en varios segmentos según el criterio del programador. Cada segmento se vuelve a dividir en varias páginas de tamaño fijo, que tienen la misma longitud que un marco de memoria principal. Si el segmento tiene menor longitud que la página, el segmento ocupará sólo una página. Desde el punto de vista del programador, una dirección lógica también está formada por un número de segmento y un desplazamiento en el segmento. Desde el punto de vista del sistema, el desplazamiento del segmento se ve como un número de página dentro del segmento y un desplazamiento dentro de la página.

La figura 7.11 propone una estructura para soportar la combinación de paginación y segmentación. Asociada con cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada segmento del proceso. Cuando un proceso determinado está ejecutándose, un registro contendrá la dirección de comienzo de la tabla de segmentos para ese proceso. Dada una dirección virtual, el procesador emplea la parte de número de segmento como índice en la tabla de segmentos del proceso para encontrar la tabla de páginas de dicho segmento. Entonces, la parte de número de página de la dirección virtual se usará como índice en la tabla de páginas para localizar el número de marco correspondiente. Este se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.

La figura 7.2a propone los formatos de la entrada de la tabla de segmentos y de la entrada de la tabla de páginas. Como antes, la entrada de la tabla de segmentos contiene la longitud del segmento. También contiene un campo base, que ahora se refiere a una tabla de páginas. Los bits de presencia y modificación no son necesarios, puesto que estos elementos se manejan a nivel de página. Pueden usarse otros bits de control para compartición y protección. La entrada de la tabla de páginas es, básicamente, la misma que se usa en un sistema de paginación pura. Cada número de página se convierte en el número de marco correspondiente si la página está presente en memoria. El bit de modificación indica si se necesita escribir la página a disco cuando se asigna el marco a otra página. Además, puede haber otros bits de control para ocuparse de la protección y de otros aspectos de la gestión de memoria.

****

**POLÍTICAS**

Lectura, Ubicación, Reemplazo, Gestión del Conjunto Residente, Vaciado y Control de Carga. Protección de Memoria: Anillos de protección en MULTICS y en Intel. Carga y Enlace, carga absoluta, carga relocalizable, carga dinámica en tiempo de ejecución.