

Capítulo 4 ADMINISTRACIÓN DE MEMORIA.

4.1 Introducción

Si bien la memoria que poseen las máquinas hoy en día es muy superior a las de las computadoras de finales del siglo pasado, los programas que ejecutan estas máquinas son también cada vez más grandes.

Por lo que la forma en la que los SO administran la memoria, sigue siendo de vital importancia.

Por otro lado, cada vez hay más dispositivos móviles con características similares y hasta superiores a las PC's del decenio de 1990, y éstos son también gobernados por un SO.

En los móviles los recursos son más limitados, y están sujetos en muchos casos a los problemas de las computadoras de escritorio de antaño.

En resumen, la gestión de memoria sigue estando vigente, y con la tendencia de cómputo ubicuo, su vigencia se vislumbra amplia.

4.2 Jerarquía de memoria

La mayoría de las computadoras tienen una jerarquía de memoria:

Memoria caché: muy rápida, pero pequeña, volátil y costosa

Memoria principal: mediana velocidad, relativamente grande, volátil y mediano precio

Memoria secundaria: lenta, muy grande, no volátil y económica

Corresponde al SO administrar esta jerarquía, coordinar y balancear de la mejor manera posible sus ventajas y desventajas.

La parte del SO que administra esta jerarquía se llama **Administrador de Memoria** (*Memory Manager*). Y su trabajo es:

- Mantenerse al tanto de qué partes de la memoria están en uso y cuales no lo están
- Asignar memoria a los procesos cuando la necesitan y recuperarla cuando terminan
- Controlar el intercambio entre la memoria principal y el disco (*swap* y memoria virtual)

4.3 Administración básica de memoria

Los sistemas de administración de memoria se pueden dividir en dos clases:

- Los que trasladan procesos entre la memoria y el disco durante la ejecución (intercambio y paginación)
- Los que no lo hacen

4.3.1 Monoprogramación sin intercambio ni paginación

El esquema de administración más sencillo posible es ejecutar sólo un proceso a la vez, compartiendo la memoria entre ese proceso y el SO.

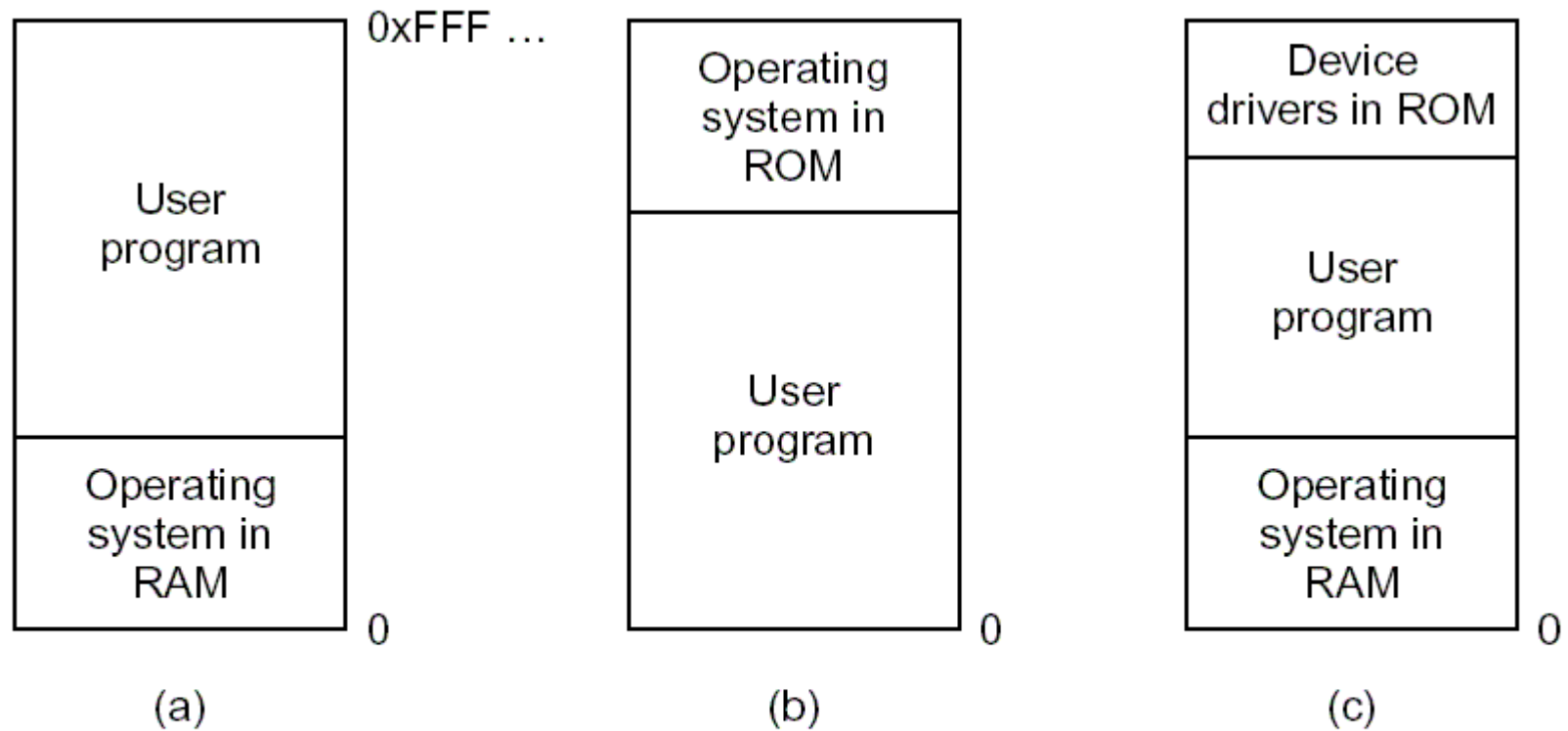


Figure 4-1. Three simple ways of organizing memory with an operating system and one user process. Other possibilities also exist.

4.3.2 Multiprogramación con particiones fijas

Este esquema permite ejecutar más de un proceso a la vez, y consiste en dividir la memoria en n particiones posiblemente desiguales, definidas por ejemplo, al iniciar el sistema (Fig. 4.2).

Cuando un proceso llega, se le coloca en la partición más pequeña que pueda contenerlo.

Ventajas: fácil de entender e implementar

Desventajas: desperdicio de memoria, discriminatorio si no se tiene cuidado con procesos pequeños, y los trabajos en las particiones se ejecutan hasta terminarse (OS/360).

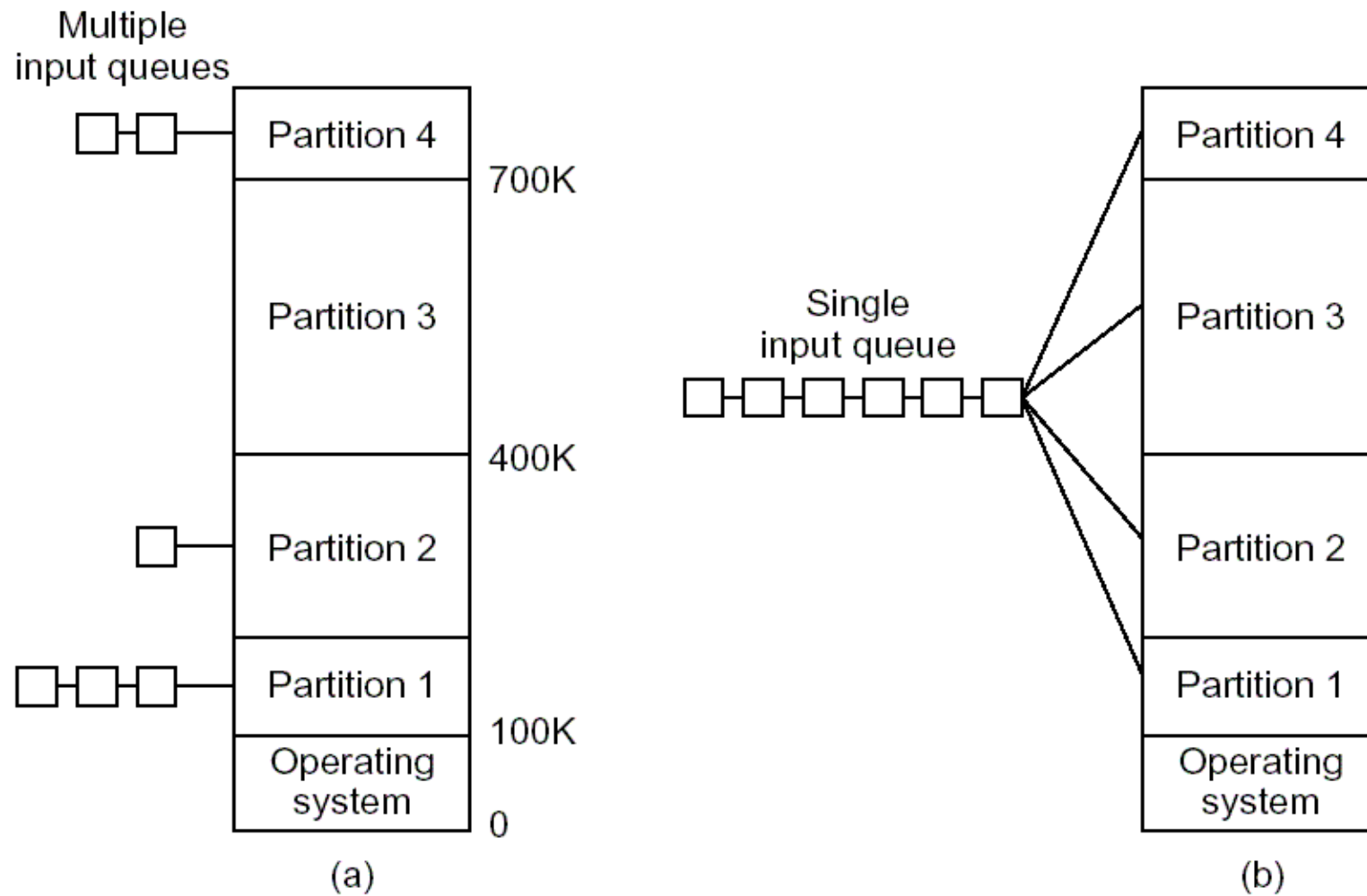


Figure 4-2. (a) Fixed memory partitions with separate input queues for each partition. (b) Fixed memory partitions with a single input queue.

Concepto de relocalización y protección.

4.4 Intercambio

Cualquier esquema que se piense, como los anteriores, son útiles y hasta deseables si es posible mantener en la memoria suficientes trabajos para conservar ocupada la CPU.

La realidad es otra, y no importa el tamaño de la memoria principal, en general no es suficiente para contener todos los procesos, por lo que algunos deben mantenerse en disco y traerse a la memoria dinámicamente.

En este sentido, existen dos enfoques:

Intercambio: traer a la memoria cada proceso en su totalidad, ejecutarlo durante algún tiempo y regresarlo a disco.

Memoria virtual: permite a los procesos ejecutarse aunque sólo estén parcialmente en la memoria principal.

El funcionamiento del intercambio se ilustra en la Fig. 4.3.

La principal diferencia respecto al esquema de particiones fijas, es que la ubicación y el tamaño de las particiones varían dinámicamente conforme los procesos se intercambian.

¿Qué ocurre si el intercambio crea múltiples huecos en la memoria? ¿Cómo corregir esta situación? (Compactación)

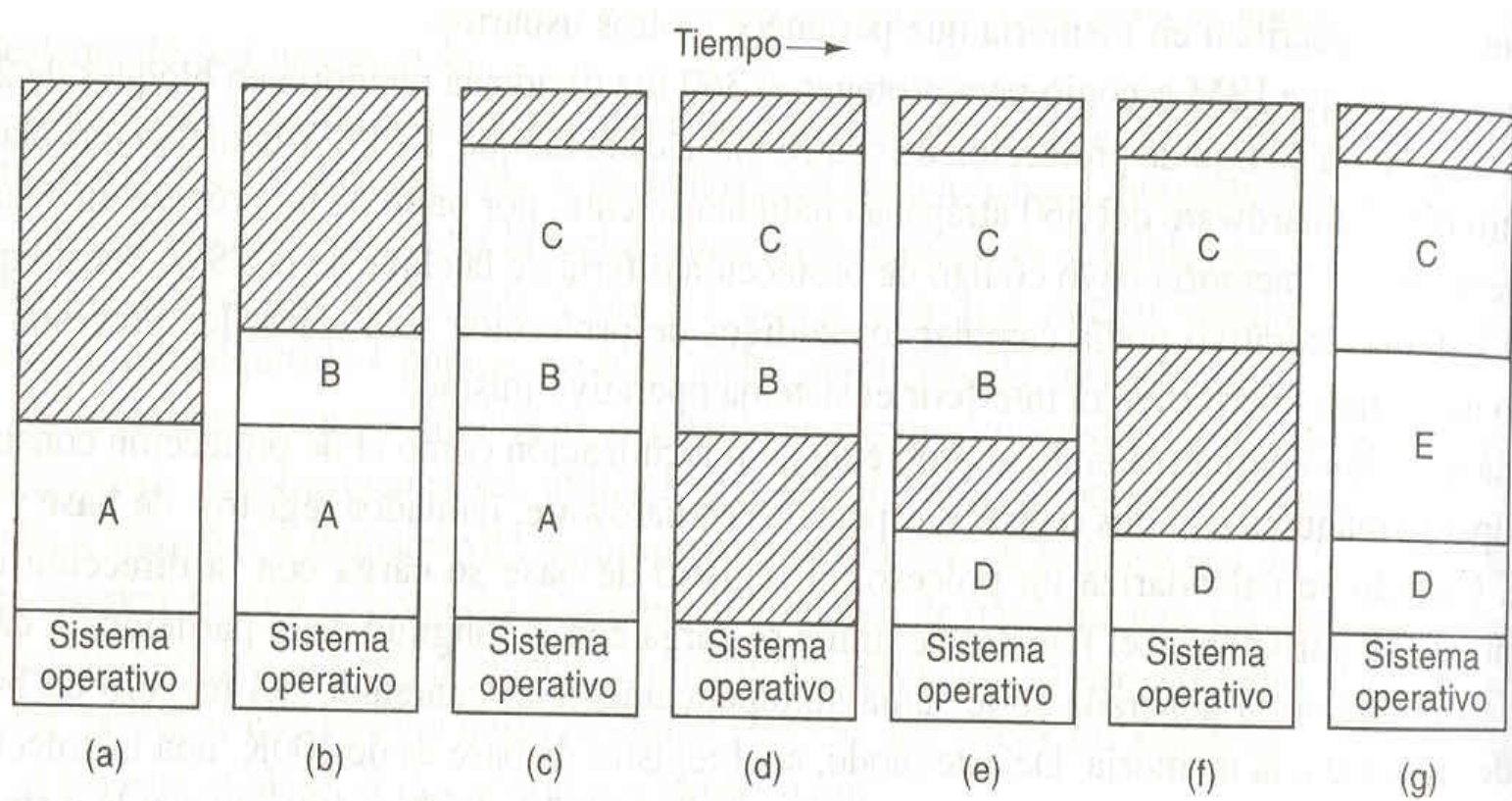


Figura 4-3. La asignación de memoria cambia conforme los procesos entran en la memoria y salen de ella. Las regiones sombreadas son memoria no utilizada.

Cuando los procesos se crean con un tamaño fijo que nunca cambia, la asignación de memoria es sencilla, pero si los segmentos de datos pueden crecer (memoria dinámica), los problemas surgen.

Si se espera que la mayoría de los procesos crezcan durante su ejecución, quizá lo más conveniente sea asignar memoria adicional cada vez que se traiga a la memoria (Fig 4.4).

En este esquema, al intercambiar un proceso a disco ¿Se intercambia sólo la memoria que se está utilizando realmente, o todo lo que se le asignó al proceso?

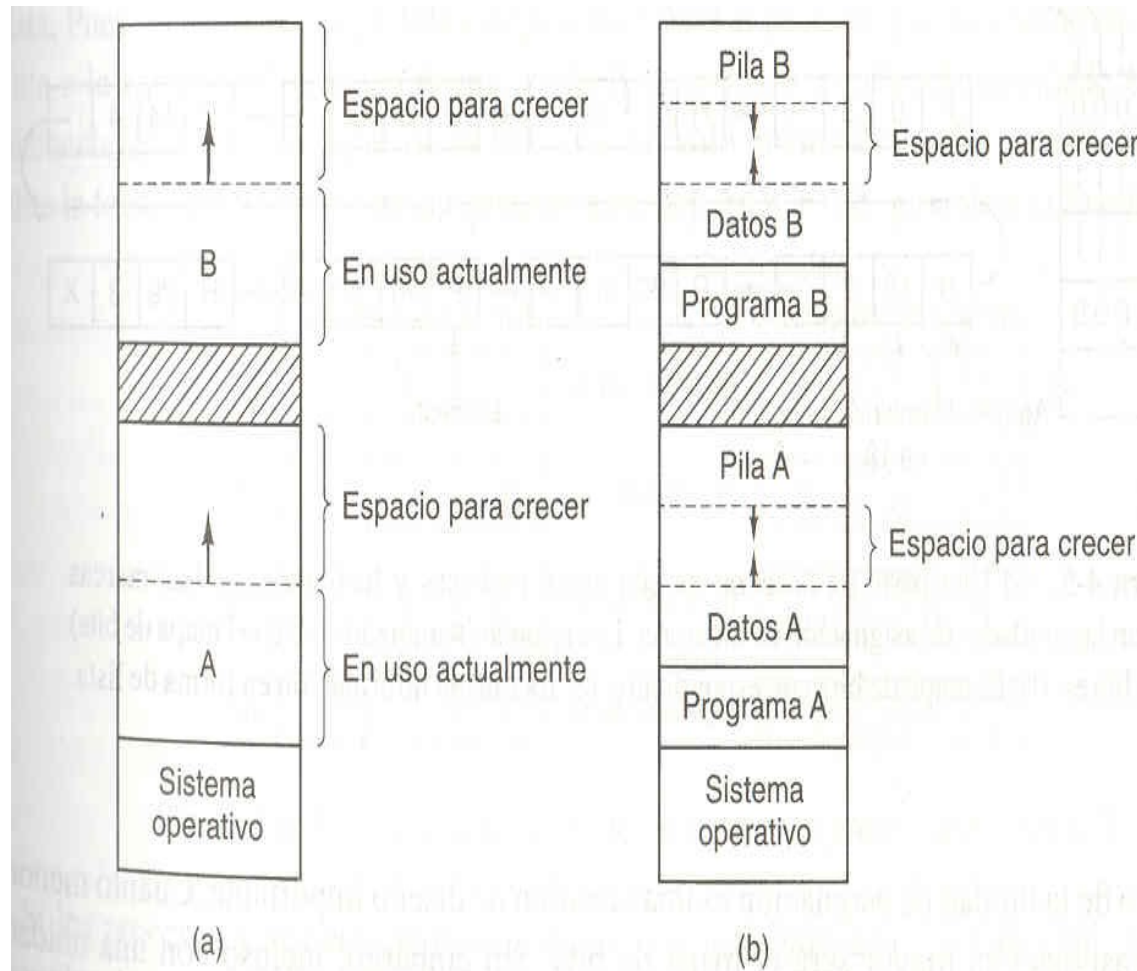


Figura 4-4. (a) Asignación de espacio para un segmento de datos que crece. (b) Asignación de espacio para una pila que crece y un segmento de datos que crece.

Como puede observarse, existen muchos aspectos a considerar.

Consulte y complemente conceptos revisando la administración de memoria con mapa de bits y la administración de memoria con listas enlazadas.

¿MINIX hace intercambio?

4.5 Memoria virtual

¿Qué sucede cuando un programa es demasiado grande para permanecer completo en la memoria disponible?

¿Qué soluciones propondría en este sentido?

La gran cantidad de memoria de las máquinas actuales ¿hacen que podamos minimizar estos cuestionamientos?

Uno de los primeros esquemas para subsanar esto se llamó: superposición.

La superposición consiste en dividir (programador) el programa en fragmentos (superposiciones), primero se ejecutaba la 0, la 1 y así sucesivamente.

El intercambio de las superposiciones corría a cargo del SO.

La memoria virtual (Fotheringham 1961) deja todo este trabajo a la computadora.

En la **memoria virtual**, el SO mantiene en la memoria principal las partes del programa que se están utilizando, y el resto de ellas, en el disco.

En un sistema de multiprogramación, se pueden mantener segmentos de muchos programas en la memoria a la vez.

4.5.1 Paginación

La paginación es el esquema de memoria virtual más utilizado.

Las direcciones generadas por los programas se denominan **direcciones virtuales** y constituyen el **espacio virtual de direcciones**.

A diferencia de un sistema sin memoria virtual, las direcciones virtuales no pasan directamente al bus de memoria sino que se envían a la **unidad de administración de memoria** (MMU).

La MMU es la entidad encargada de transformar direcciones virtuales a direcciones físicas (fig. 4.7).

En la fig 4.8 se muestra un ejemplo sencillo de transformación.

El espacio virtual de direcciones se divide en **páginas** y el espacio físico de direcciones en **marcos de páginas**, y siempre tienen el mismo tamaño.

Los tamaños de página son potencias de 2 (1024K, 64K, etc).

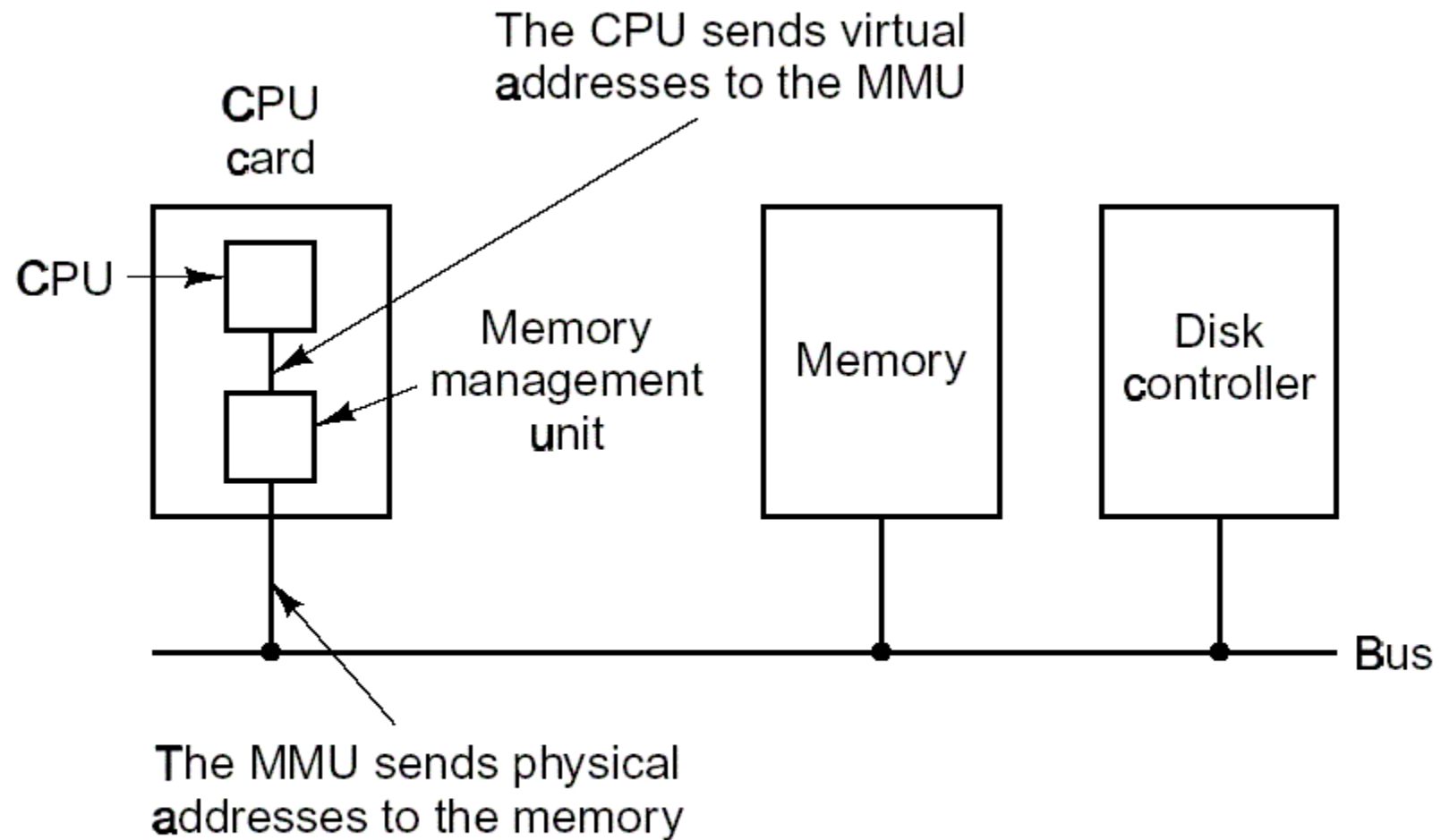


Figure 4-7. The position and function of the MMU.

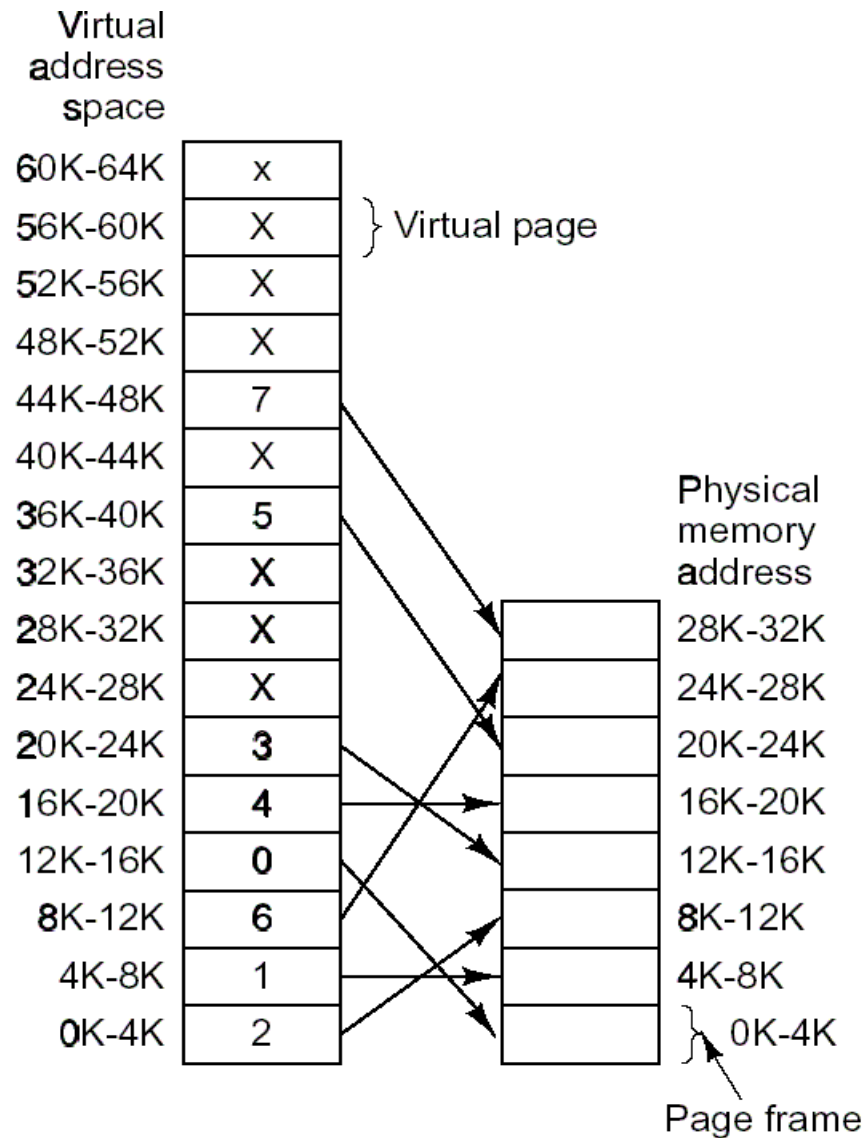


Figure 4-8. The relation between virtual addresses and physical memory addresses is given by the page table.

Las transferencias entre la memoria y disco siempre se efectúan en unidades de una página.

La transformación de direcciones mapeadas a través de la MMU resulta sencillo, pero ¿qué pasa si un programa trata de usar una página sin correspondencia?

Cuando esto sucede la MMU hace que la CPU pase el control por una trampa del SO llamada **falla de página**.

El SO escoge un marco de página poco utilizado, lo escribe en disco, carga la página a la que se hizo referencia, y la coloca en dicho marco, modifica el mapa y reinicia la instrucción atrapada.

¿Cómo funcionará internamente la MMU?

Observe y analice la fig 4.9.

El número de página se utiliza como índice de la **tabla de páginas**, produciendo el número de marco de página que le corresponde a esa página virtual.

Si el bit presente/ausente es 0, se genera la trampa, si es 1, se hace la traducción correspondiente.

El registro de salida es el que contiene la dirección que se coloca en el bus de memoria como dirección física.

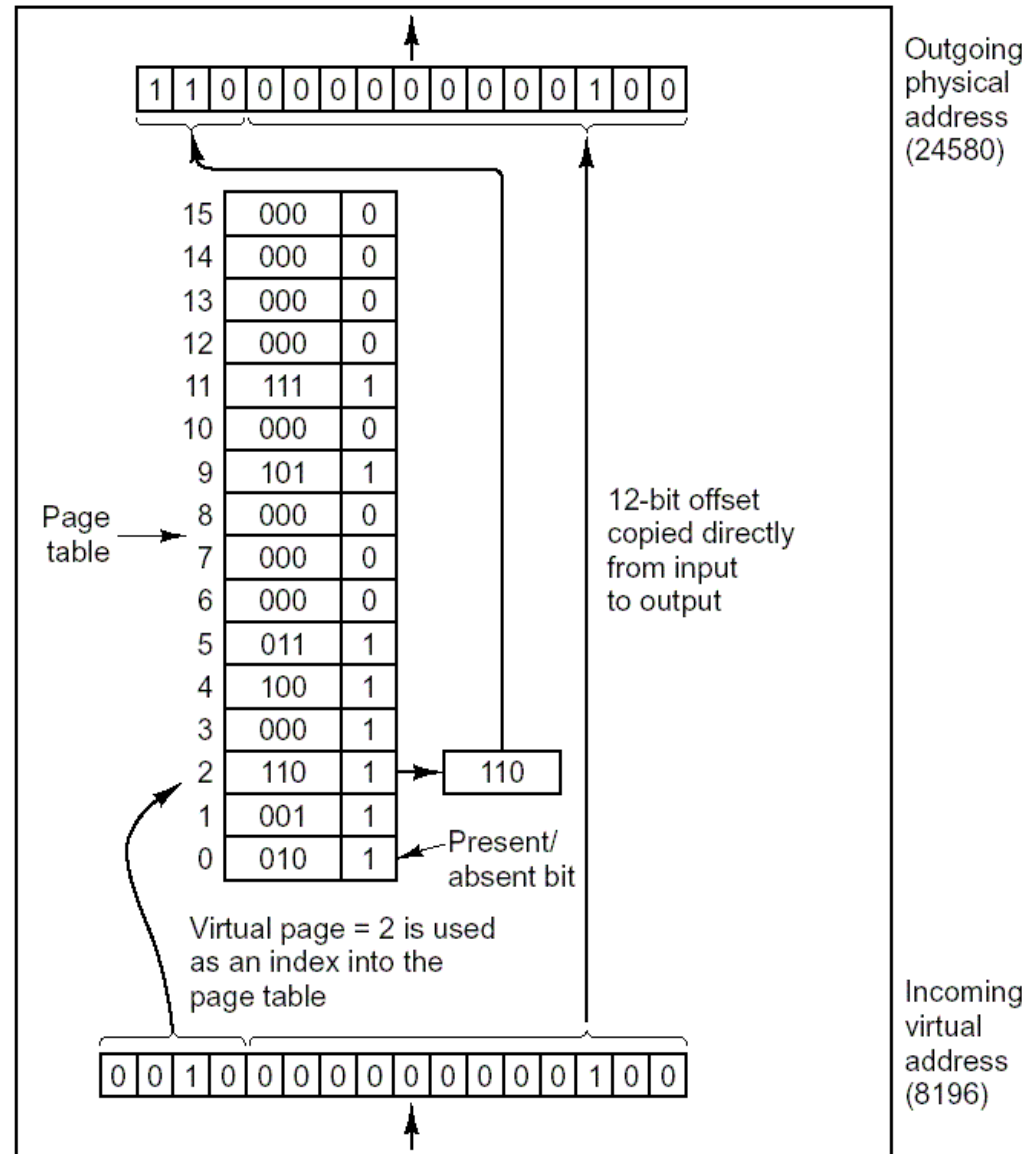


Figure 4-9. The internal operation of the MMU with 16 4K pages.

Considere la generalización del concepto para la capacidad de direccionamiento de 32 y 64 bits.

No olvide además que se mantiene una tabla de páginas por proceso ¿puede visualizar las implicaciones de esto?

La esencia de la transformación radica en las tablas de páginas, revise en la bibliografía aspectos de su implementación y rendimiento, así como enfoques alternativos de traducción.

Revise el concepto de tablas de páginas multinivel en el libro de Tanenbaum.

4.6 Algoritmos de sustitución de páginas

Cuando ocurre un fallo de página ¿qué página quitar de la memoria para dar cabida a la nueva?

Describe qué consideraciones deben hacerse antes de sobrescribir la página seleccionada para su eliminación.

4.6.1 Algoritmo óptimo

El algoritmo de sustitución de páginas óptimo, dice que debe eliminarse la página que más vaya a tardar en ser referida, es decir, aplazar el problema el mayor tiempo posible.

4.6.2 Algoritmo no usadas recientemente (NRU)

Considere 2 bits de estado por cada página: R (Referida) y M (Modificada).

R se enciende cada vez que la página asociada es referida ya sea para lectura o escritura.

M se enciende la página es modificada.

Esta modificación de los bits puede ser por HW o por SW (SO).

Inicialmente todos los bits están en 0 cuando se inicia un proceso y, periódicamente se apaga el bit R.

Esto crea 4 clases de páginas:

Clase	Bit R	Bit M
0	0	0
1	0	1
2	1	0
3	1	1

El algoritmo NRU elimina una página al azar de la clase no vacía que tiene el número más bajo.

4.6.3 Algoritmo FIFO

Aquí el SO mantiene una cola de todas las páginas en memoria, siendo la de la cabeza la más antigua.

Ante un fallo de página, se elimina la que está a la cabeza y se agrega la nueva al final.

¿Qué sucede si la más antigua es, si no la más utilizada, sí es de las más utilizadas?

4.6.4 Algoritmo de segunda oportunidad

Este algoritmo es básicamente una modificación del anterior y consiste en inspeccionar el bit R de la página más antigua.

Si es 0, además de vieja no ha sido utilizada recientemente y se debe reemplazar.

Si es 1, se apaga el bit y se coloca al final de la cola, como si fuera una página recientemente traída a memoria.

La búsqueda continúa.

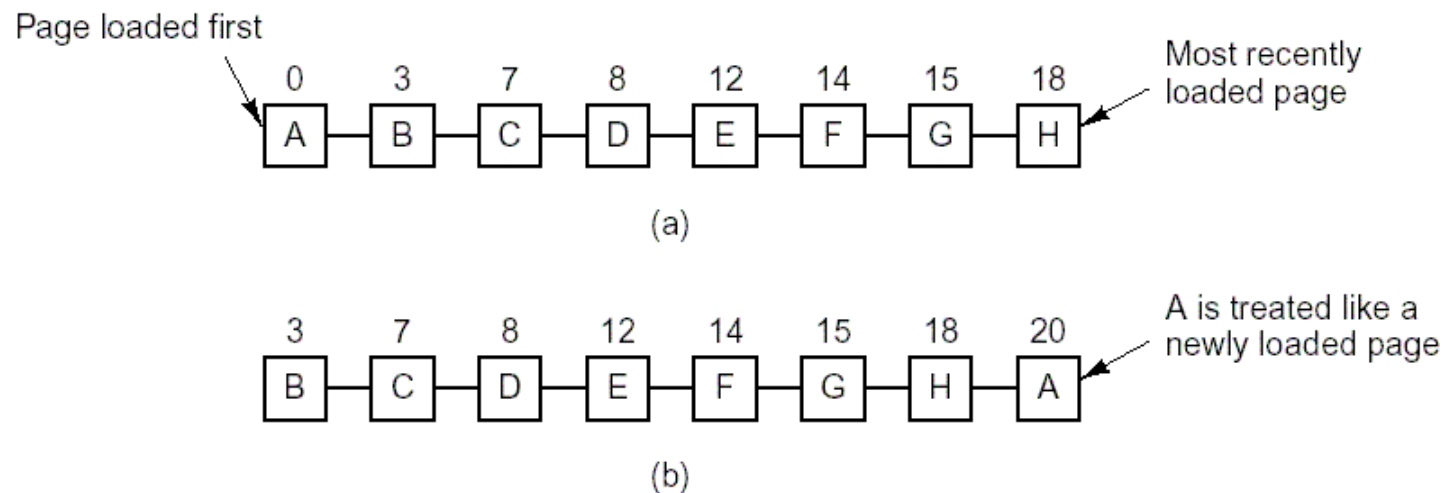


Figure 4-13. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and *A* has its *R* bit set.

4.6.5 Algoritmo por reloj

Este es básicamente una variación del anterior en lo que respecta a la implementación.

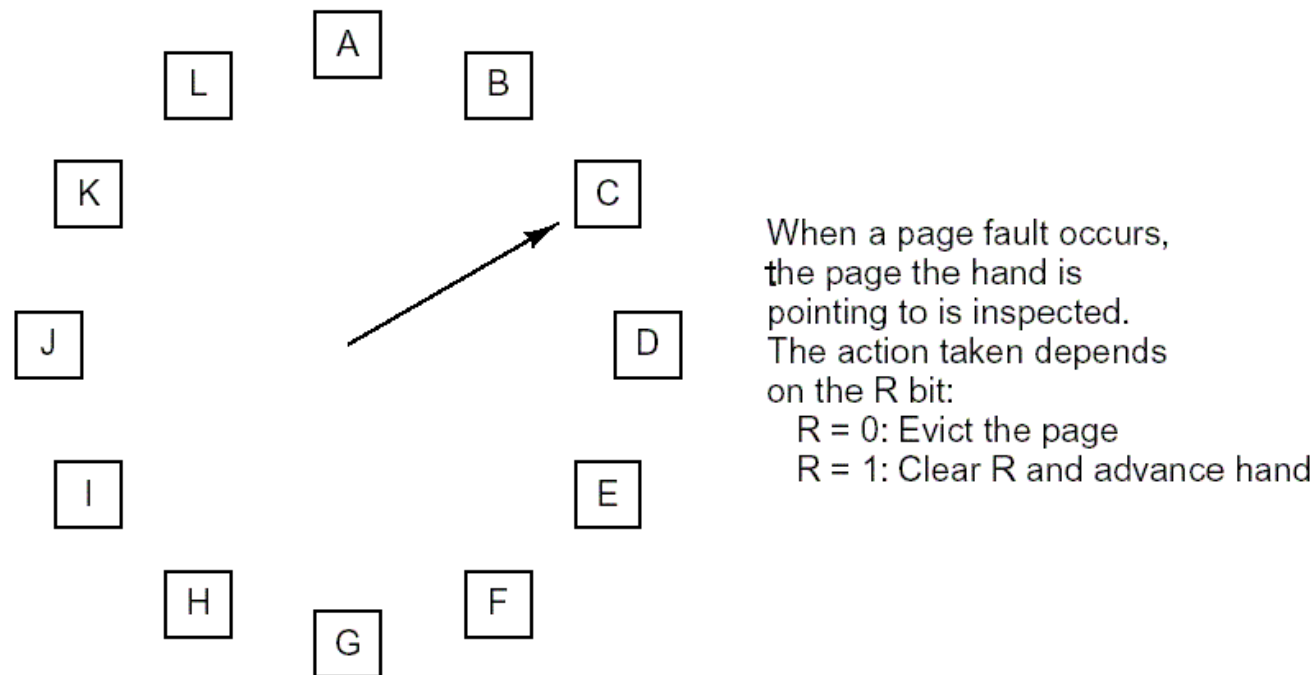


Figure 4-14. The clock page replacement algorithm.

4.6.6 Algoritmo menos recientemente usadas (LRU)

Un patrón que presenta frecuentemente, es la observación de que las páginas que se han usado mucho en las últimas instrucciones, probablemente se usarán mucho en las siguientes.

Por el contrario, las páginas que hace mucho no se usan, probablemente permanecerán así.

Este patrón sugiere un algoritmo casi óptimo aunque costoso: cuando ocurra un fallo de página, se eliminará la página que haya estado más tiempo sin usarse (LRU).

Una posible implementación es mantener una lista enlazada de páginas activas en memoria, con la más recientemente utilizada al frente. El problema es actualizar la lista en cada referencia a memoria y modificarla en correspondencia.

Otra manera es equipar el HW con un contador de 64 bits (C), el cual se incrementaría automáticamente en cada instrucción. Cada entrada en la tabla de páginas debe contener una entrada para almacenar el contador. Así, después de cada referencia a memoria, el valor de C se almacenaría en la entrada correspondiente a la página a la que se acaba de hacer referencia.

En el enfoque anterior, la LRU es la que contenga el contador más bajo, lo cual implica una búsqueda.

Otra implementación es, para una máquina con n marcos de página, se tiene una matriz de $n \times n$ bits inicialmente en 0. Cada vez que se haga referencia al marco de página k , se ponen en 1 los bits de la fila k y en 0 los bits de la columna k .

En el esquema anterior, la fila cuyo valor binario sea el más bajo será la página LRU.

Para la fig. 4.15, se tienen las siguientes referencias a páginas:

0 1 2 3 2 1 0 3 2 3

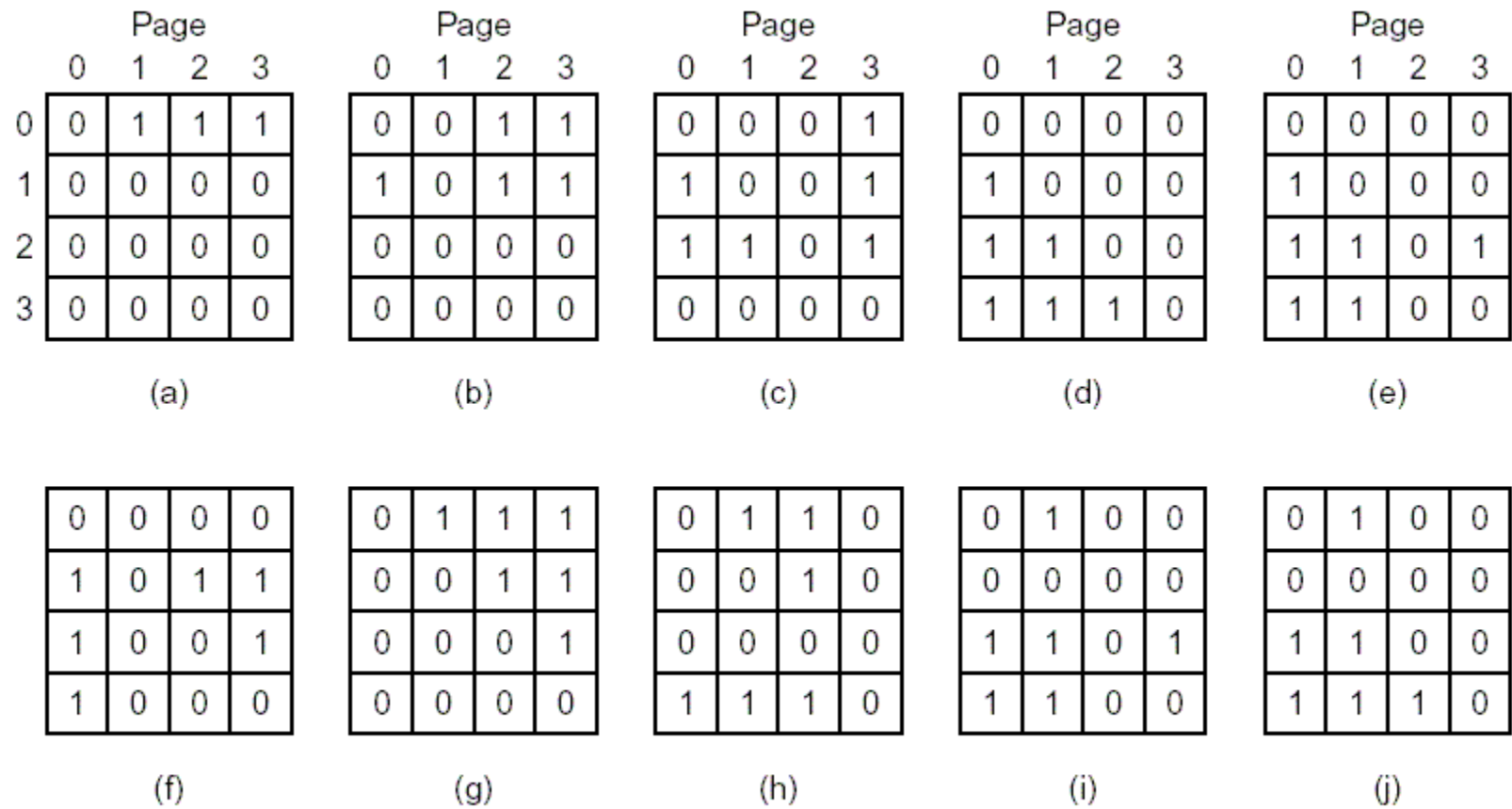


Figure 4-15. LRU using a matrix.

Existen otras variaciones como el algoritmo no utilizada frecuentemente (NFU) y algoritmos de maduración de páginas entre otros.

Se recomienda revisar estos enfoques en la bibliografía.

4.7 Segmentación

Aunque la memoria virtual es un enfoque sumamente novedoso, para muchos problemas el tener dos o más espacios de direcciones virtuales independientes sería mucho mejor.

Considere la situación representada en la fig. 4.19.

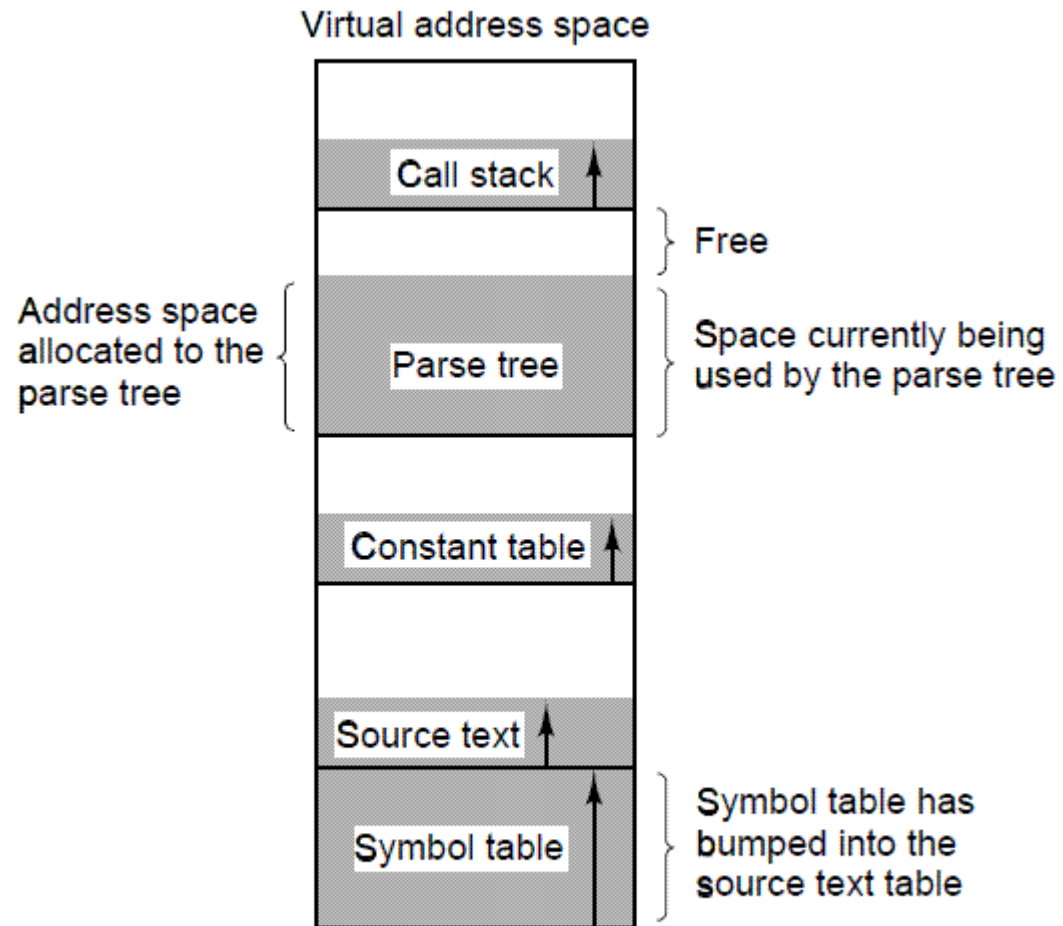


Figure 4-19. In a one-dimensional address space with growing tables, one table may bump into another.

Una solución a lo anterior y mucho más general es proveer a la máquina con muchos espacios de direcciones completamente independientes llamados **segmentos**.

Puesto que cada segmento constituye un espacio de direcciones separado, puede crecer o encogerse de forma independiente sin afectarse entre sí.

Para especificar una dirección en una memoria segmentada o bidimensional, el programa debe proporcionar una dirección de 2 partes: número de segmento y dirección dentro del segmento.

La fig 4.20 ilustra una memoria segmentada para el ejemplo anterior.

Es importante recalcar que un segmento es una entidad lógica de la cual el programador está consciente.

Un segmento podría contener un procedimiento o función, un arreglo, una pila o una colección de variables, pero por lo general no contiene una mezcla de diferentes cosas.

La segmentación también facilita las bibliotecas compartidas.

La compartición del segmento de texto para aplicaciones en estaciones de trabajo es un ejemplo clásico de ello.

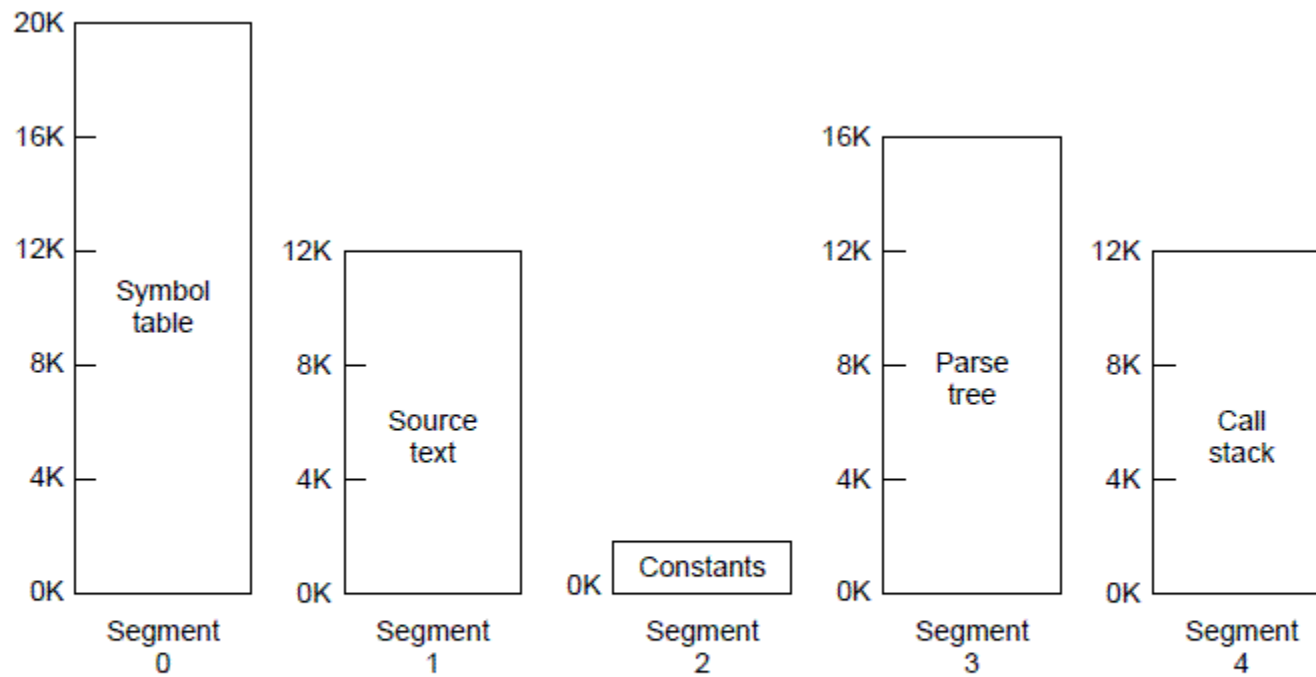


Figure 4-20. A segmented memory allows each table to grow or shrink independently of the other tables.

Existen también esquemas combinados, como la segmentación con paginación.

Consulte información complementaria respecto a la segmentación y la implementación de, en la bibliografía.

Revise también la organización de la memoria en MINIX.