TEMA 5: GESTIÓN DE MEMORIA

Gestor o administrador de memoria: Parte del sistema operativo que lleva registro de la memoria libre y ocupada, asigna la memoria a los procesos cuando la necesitan y la libera cuando han terminado su ejecución. También gestiona el intercambio entre memoria central y disco.
Es un recurso importante y caro hasta hace poco tiempo.

1. Conceptos básicos relacionados con el almacenamiento

1.1. Organización del almacenamiento. Evolución

El almacenamiento principal se considera un recurso caro y por ello los diseñadores de sistemas intentan optimizar su uso.

Se entiende por *organización del almacenamiento* la forma de considerar u organizar el almacenamiento principal. Se plantean las siguientes preguntas:

- ¿En el almacenamiento principal se debe colocar sólo un usuario o varios al mismo tiempo?

- Si hay varios programas de usuario al mismo tiempo en memoria principal, ¿se les asigna a cada uno la misma cantidad de espacio o se divide el almacenamiento en diferentes tamaños?

- ¿Se pueden ejecutar los trabajos en cualquier parte de la memoria o deben permanecer en unas particiones específicas?

Se han construido sistemas que aplican cada uno de estos esquemas; algunos de los cuales se verán en este tema.

En la siguiente figura podemos apreciar como ha evolucionado la organización del almacenamiento desde los sistemas de almacenamiento real dedicados a un solo usuario hasta los sistemas de almacenamiento virtual dedicado a múltiples usuarios.

<table>
<tr><th>Real</th><th colspan="3">Real</th><th colspan="3">Virtual</th></tr>
<tr><td>Sistemas dedicados de un solo usuario</td><td colspan="3">Almacenamiento real en sistemas de multiprogramación</td><td colspan="3">Almacenamiento virtual en sistemas de multiprogramación</td></tr>
<tr><td></td><td colspan="2">Multiprogramación con particiones fijas</td><td>Multiprogramación con particiones variables</td><td>Paginación pura</td><td>Segmentación Pura</td><td>Pagin. segm. combinadas</td></tr>
<tr><td></td><td>Absoluto</td><td>Reubicable</td><td></td><td></td><td></td><td></td></tr>
</table>

Evolución de la organización del almacenamiento

Asociado con la organización del almacenamiento se encuentra la asignación del almacenamiento. Esta puede ser contigua y no contigua.

Una *asignación de almacenamiento es contigua* cuando cada proceso ocupa un único bloque contiguo de posiciones (localidades) de memoria. Este esquema se empleaba en los primeros sistemas de computación.

Una *asignación de almacenamiento es no contigua* cuando un proceso que se encuentra dividido en varios bloques o segmentos, se coloca en el almacenamiento principal en fragmentos que no necesitan ser adyacentes. Es más difícil para un sistema operativo controlar la asignación del almacenamiento no contiguo, pero la ventaja es que si la memoria principal tiene muchos huecos, es posible en ocasiones poder aprovecharlos para cargar y ejecutar programas que en otro caso tendrían que esperar.

1.2. Administración del almacenamiento

Sea cual sea el esquema de organización del almacenamiento que se adopte para un sistema específico, es necesario decidir que estrategias se deben utilizar para obtener un rendimiento óptimo. Las *estrategias o políticas de administración del almacenamiento* determinan el comportamiento de una organización determinada cuando se siguen diferentes políticas. Estas políticas se utilizan para obtener el mejor aprovechamiento posible del almacenamiento principal. Se plantean las siguientes preguntas:

- ¿Cuándo se toma un nuevo programa para colocarlo en memoria?
- En que lugar del almacenamiento principal se coloca el siguiente programa a ejecutar?
- Si hay que colocar un nuevo programa y ésta está llena, ¿Cuál de los otros programas se desaloja?

La administración del almacenamiento se divide en las siguientes categorías:

- Políticas de obtención
- Políticas de colocación
- Políticas de reemplazo

Políticas de obtención

Determinan cuándo debe obtenerse la siguiente parte del programa o los datos que se van a transferir del almacenamiento secundario al principal. Se emplean básicamente la obtención por demanda y la anticipatoria.

Políticas de colocación

Tienen que ver con la determinación de la parte del almacenamiento principal donde se colocará un programa entrante (nuevo o intercambiando). Algunos de los que se emplean son las denominadas del *primer ajuste*, *mejor ajuste* y del *peor ajuste*.

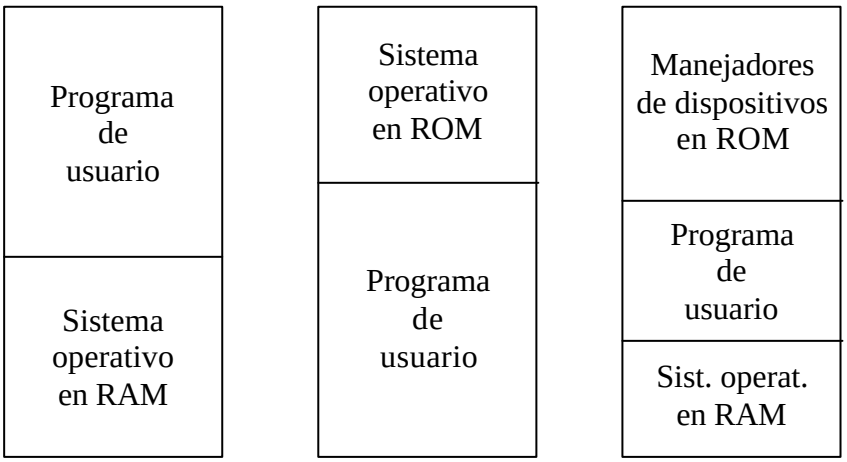
Políticas de reemplazo

Están relacionadas con la determinación de qué parte del programa o de los datos se debe desalojar para dejar espacio a los programas entrantes. Algunos de los que se emplean son FIFO, LRU, NFU, etc.

2. Gestión de memoria en monoprogramación / multiprogramación

2.1. Gestión de memoria en monoprogramación

- Sólo hay un proceso en memoria
- Cada proceso debe contener los controladores de los dispositivos de entrada / salida que use.
- Tres formas de organizar la memoria (con un s.o. y un proceso de usuario):



Ejemplos de organización de memoria en monoprogramación

Funcionamiento:

- El usuario escribe un comando sobre el terminal.
- El sistema operativo carga el programa desde disco a memoria y lo ejecuta.
- Al finalizar dicho programa, el sistema operativo muestra el prompt y espera a otra demanda.

Inconvenientes de la monoprogramación:

- Mala utilización de la memoria (desperdicio).
- Tiempo de ocio del procesador central en entrada / salida.
- Limitación en el tamaño total del programa.

2.2. Gestión de memoria en multiprogramación

Multiprogramación:

Ejecución entrelazada o concurrente de dos o más procesos. Se opera sobre más de un proceso a la vez, y los recursos del sistema se distribuyen entre esos procesos.

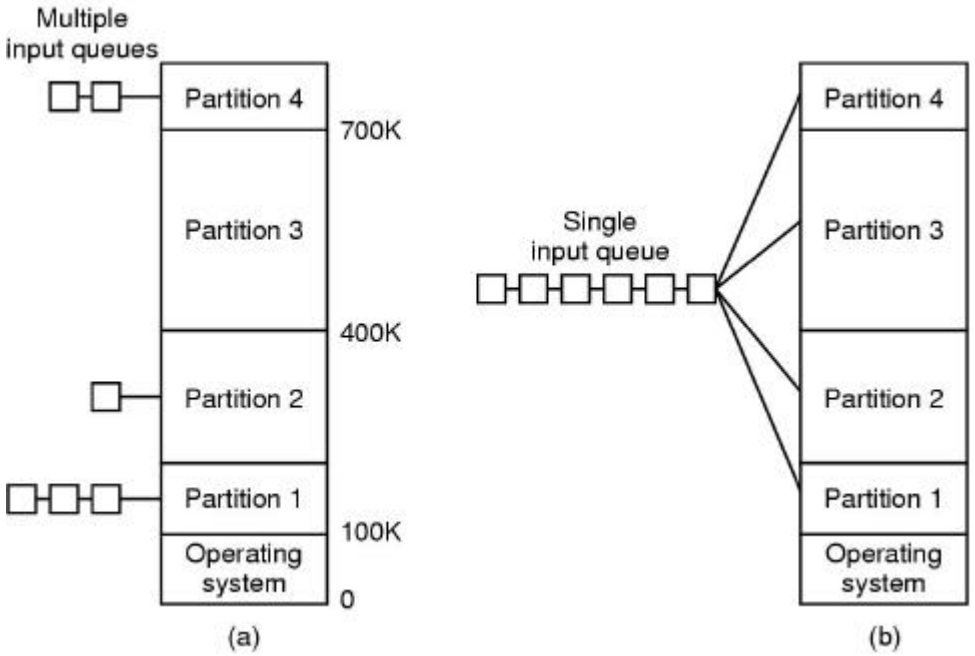
Motivaciones que nos inducen a usar la multiprogramación:

- Desdoblamiento de un programa en dos o más procesos. Hace más fácil programar una aplicación.
- Aprovechamiento de la CPU. A consecuencia de solapar los tiempos de entrada / salida con los de CPU.

3. Multiprogramación con almacenamiento real

3.1. Multiprogramación con particiones fijas

La memoria principal se divide en “n” particiones fijas. Los procesos entran en memoria por una cola simple o por colas separadas según tamaño (en la cola de la partición más pequeña mayor que el tamaño del proceso).



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

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

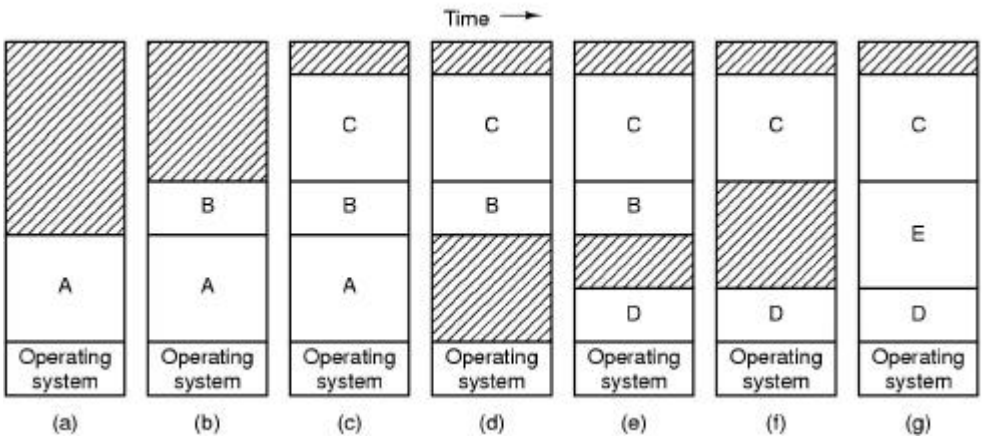
3.2. Multiprogramación con particiones variables

Partición Variable:

Tanto el número como el tamaño de los procesos varían dinámicamente.

Problema → La asignación y liberación de memoria es más complicada que con las particiones fijas.

Ventaja → Se aprovecha más la memoria ya que a cada proceso se le asigna la memoria principal que necesita.



Evolución de varios procesos en el tiempo en un sistema con particiones variables de memoria. Las regiones ralladas corresponden a memoria libre (no usada).

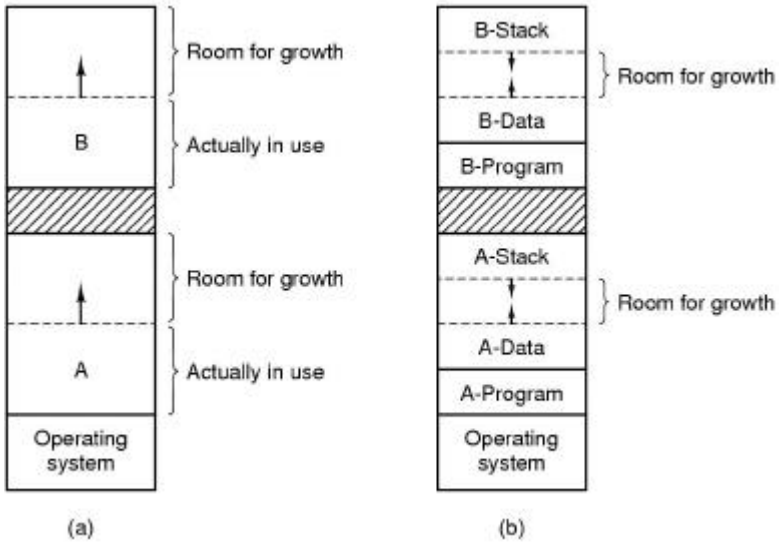
Surgen dos problemas:

1. Fragmentación de la memoria. Como consecuencia de la entrada/salida de procesos en la memoria.

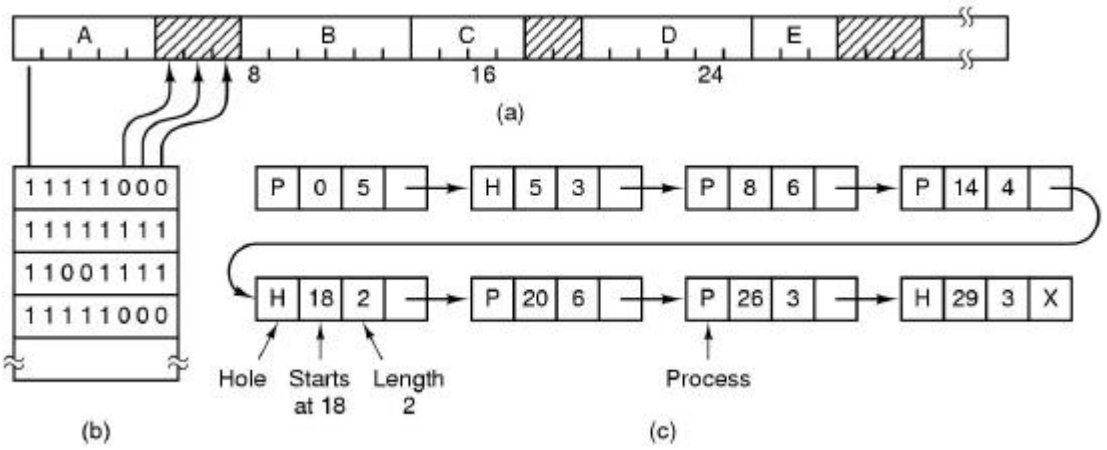
   Solución → Compactación de memoria. Se combinan todos los agujeros en uno más grande. Requiere de mucho tiempo de CPU o de un hardware especial.

2. Crecimiento dinámico de los segmentos de datos. Debido a que no existe un agujero adyacente el proceso (para asignarle espacio extra), y que no se puede mover a un agujero más grande.

   Solución → Reservar un área de memoria extra



(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 también.



Representación de una parte de la memoria mediante: Mapa de bits y listas enlazadas.

(a) Representación de una parte de la memoria con cinco procesos y tres huecos. Las rayitas muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres.

(b) Mapa de bits correspondiente.

(c) La misma información en forma de lista enlazada.

4. Multiprogramación con almacenamiento virtual (memoria virtual)

Cuando los programas son muy grandes para cargarlos en memoria, éstos se dividen en trozos llamados “overlays” (capas, recubrimientos), que son intercambiados entre memoria central y disco por el sistema operativo. La división del programa en “overlays” es misión del programador.

Memoria Virtual:

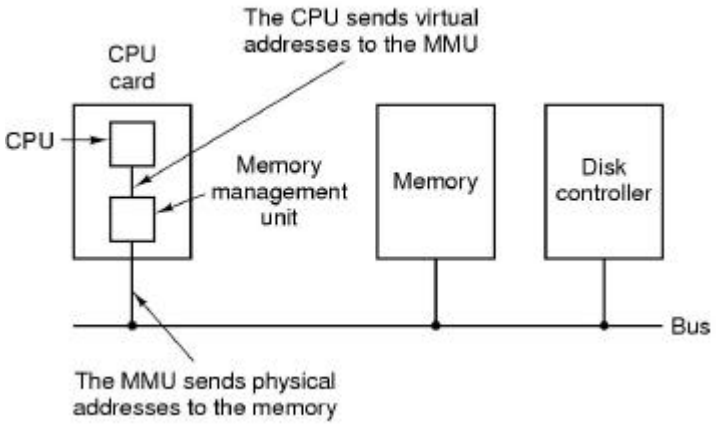
El tamaño combinado del programa, datos y pila puede exceder la cantidad de memoria física disponible. El sistema operativo guarda aquellas partes del programa concurrentemente en uso en memoria central y el resto en disco. Cuando un programa espera que se le cargue en memoria central de disco otra parte del mismo, la C.P.U. se puede asignar a otro proceso.

Utiliza una asignación no contigua de la memoria.

Direcciones virtuales (no reales) → Las generadas por el programa. El conjunto de direcciones virtuales forman el espacio de direcciones virtual

Funcionamiento:

- En sistemas sin memoria virtual, la dirección del programa se coloca directamente sobre el bus.
- Cuando se usa memoria virtual, la dirección virtual se envía a la unidad de gestión de memoria (MMU, "Memory Managment Unit"). La MMU lo forma aquella parte del hardware que transforma la dirección virtual en la dirección de memoria física que se coloca directamente en el bus:



Posición y función de la MMU

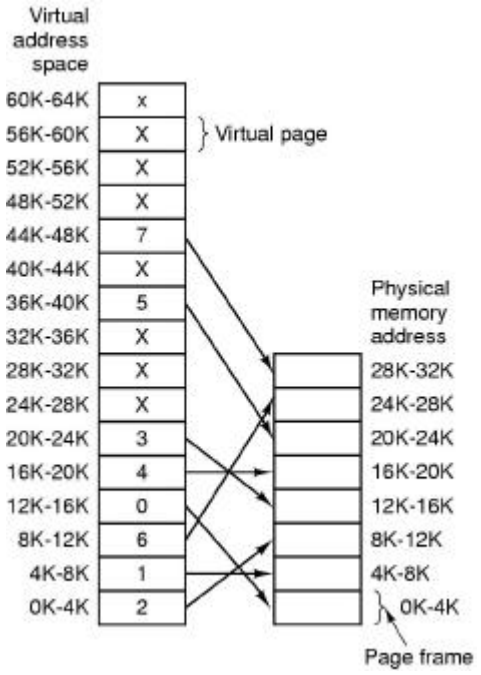
4.1. Paginación

El espacio virtual de direcciones se divide en unidades llamadas páginas, todas del mismo tamaño. La memoria principal se divide en marcos de páginas o cuadros de páginas (page frames) del mismo tamaño que las páginas virtuales y son compartidas por los distintos procesos del sistema (en cada marco de página se carga una página de un proceso).

No todo el espacio virtual de direcciones está cargado en memoria central. Una copia completa se encuentra en disco y las páginas se traen a memoria central cuando se necesitan.

Ejemplo:

- Página de 4KB
- Espacio de direcciones virtual de 64KB (direcciones de 16 bits)
- Espacio de direcciones físicas de 32KB



La relación entre direcciones virtuales y direcciones de la memoria física está dada por la tabla de páginas.

Transformación de direcciones:

Dirección Virtual			**Dirección Física**	
0000000000000000	0	Pág. 0, desplaz. 0	8192	Pág. 2, desplaz. 0
0100000000000000	8192	Pág. 2, desplaz. 0	24576	Pág. 6, desplaz. 0
1010000000010100	20500	Pág. 5, desplaz. 20	12308	Pág. 3, desplaz. 20
1101101011000000	28000	Pág. 6 (no cargada)	No tiene dirección física asociada	

La transformación del número de página virtual al número de página física de memoria se realiza mediante la **tabla de páginas**.

Hay páginas que no tienen correspondencia en memoria central. El hardware asocia un bit de presencia/ausencia en cada entrada.

- Páginas activas: páginas de un proceso residente en memoria central.
- Páginas inactivas: páginas de un proceso no residente en memoria principal (en memoria secundaria).

Si se intenta usar una dirección virtual que no se encuentra en memoria física, la MMU envía una interrupción al sistema operativo (falta de página, page fault). En ese caso, el sistema operativo saca, si es necesario, una página física de la memoria central, la copia en el disco (por si ha sufrido alguna modificación) y carga la página necesaria en memoria central cambiando la tabla de páginas y restaurando la instrucción interrumpida.

Transformación de la dirección virtual en dirección física:

Los bits de mayor peso de la dirección se interpretan como el número de la página y los de menor peso como el número de palabra dentro de la página.

Si el tamaño de la página es igual a 2^n, con "n" bits se direcciona cualquier palabra dentro de una página (el resto se utiliza para direccionar la página).

Ejemplo:
Direcciones virtuales de 20 bits de longitud, tamaño de la página igual a 512 palabras (direcciones, bytes) y direcciones físicas de 14 bits.

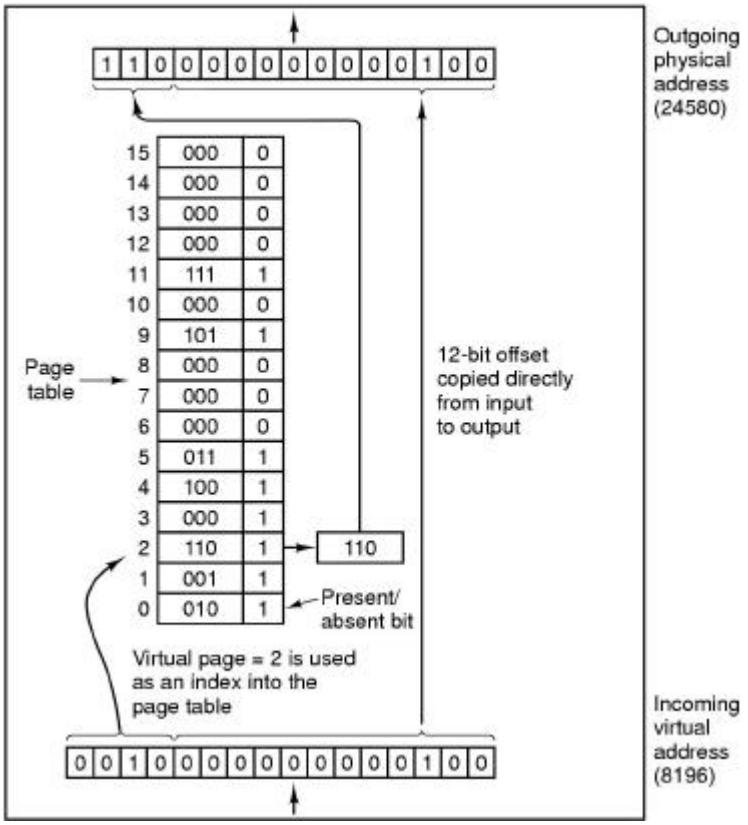
- Tamaño de memoria virtual = 2^{20} palabras
- Tamaño de la página = 512 palabras = 2^9
  Con los 9 bits inferiores se direcciona la palabra, el resto es para el número de la página.
- Número de páginas virtuales : 2^{11} páginas.
- Número de páginas físicas: $2^{14-9} = 2^5 = 32$ páginas

Ejemplo:

Transformación interna de una dirección virtual en una dirección física.

- Tamaño de la memoria física = 32K palabras
- Tamaño de la memoria virtual (espacio de direcciones virtuales) = 64K palabras
- Tamaño de la página = 4K palabras

Número de páginas virtuales = $2^{16 - 12} = 2^4 = 16$ páginas virtuales



Operación interna de la MMU con 16 páginas de 4K

Dirección virtual = 8196 = 8192 + 4

Dirección física = 24580 = 24576 + 4

Offset (desplazamiento) de 12 bits (4096 direcciones)

Espacio de direcciones unidimensional

Otros esquemas de transformación de direcciones en sistemas paginados

1. Transformación directa de direcciones:

Se tiene una tabla de páginas para cada proceso en memoria central y disponemos de un registro hardware que contiene la dirección base de la tabla de páginas del proceso en ejecución (su valor forma parte del entorno volátil).

Para realizar la transformación directa de direcciones, la tabla de páginas contiene una entrada para cada página virtual (hay que acceder dos veces a memoria central)

2. Transformación asociativa de direcciones:

La tabla de páginas físicas (bloques) se guarda en una memoria asociativa consistente en un conjunto de registros de direcciones de página (PAR, "Page Address Register"), una por cada página física. Cada página de la dirección virtual se busca de forma simultánea en todos los PARs (propiedad de los PAR).

En cada PAR habría que añadir un campo para distinguir páginas de los distintos procesos.

Inconveniente:
Se necesitan tantos PARs como páginas físicas hay en memoria (si ésta es grande, el costo aumenta ya que se necesitan muchos PARs).

3. Solución intermedia:

Traducción de direcciones por combinación de los dos casos anteriores. Se dispone de una pequeña memoria asociativa para referenciar a unas pocas páginas (las más recientes) asociadas a un proceso. Se tendrá una memoria asociativa para todos los procesos con lo cual la información que contiene será parte del entorno volátil. Otra posibilidad sería tener una memoria asociativa por proceso con lo cual se limitaría el número de procesos. La búsqueda se solapará entre la memoria asociativa y la tabla de páginas que hay en memoria para optimizar el tiempo de localización.

El rendimiento de sistemas que usan un mapa parcial puede llegar a ser del 90% (o mayor) del rendimiento correspondiente al mapa asociativo completo con sólo usar 8 o 16 registros PAR.

4.2. Segmentación

- El espacio de direcciones se divide en segmentos, cada uno de los cuales corresponderá a una rutina (procedimiento, función), un programa o un conjunto de datos (una entidad lógica). Todo aquello que se corresponda con sub-espacio de direcciones independientes.

- Cada programa contiene una cierta cantidad de segmentos. Los primeros segmentos se reservan para procedimientos, datos y pila, pertenecientes al programa en ejecución. Los segmentos restantes contienen un fichero por segmento, así que los procesos pueden direccionar todos sus ficheros directamente sin tener que abrirlos ni usar primitivas especiales de entrada/salida. Cada fichero puede crecer de forma completamente independiente de los otros, con cada byte direccionado por un par (segmento, offset).

- Por otro lado, colocando objetos diferentes en diferentes segmentos, se facilita la compartición de estos objetos entre procesos múltiples.

Ejemplo:

Se utiliza un esquema segmentado / paginado
Soporte de hardware para una máxima de 16 procesos cada uno con 1024 páginas de 4K (espacio virtual de 4 memorias para cada proceso).
El hardware del MMU contiene una tabla con 16 secciones (una por proceso). Cada sección tiene 64 descriptores de segmentos (el espacio de direcciones se descompone en 64 segmentos, cada uno de 16 páginas).

Ejemplo:

Un compilador de Pascal podría crear segmentos separados para:

- Variables globales
- Pila de llamadas a procedimientos (para almacenar parámetros y direcciones de retorno)
- Porción de código de cada procedimiento y función.
- Variables locales de cada procedimiento o función.
- Heap

4.3. Diferencias entre paginación y segmentación

- En paginación, el espacio de direcciones es lineal (unidimensional) y se hace una división física del mismo. En segmentación, el espacio de direcciones es bidimensional, se hace una división lógica del mismo (tiene sentido compartir segmentos).

- Las páginas son de tamaño fijo, determinado por la arquitectura del sistema. Los segmentos pueden ser de cualquier tamaño.

- La división de las direcciones virtuales en número de página y desplazamiento es tarea del hardware, mientras que la división en segmento y desplazamiento es tarea del programador.

4.4. Algoritmos de reemplazamiento de página

Cuando ocurre una falta de página, el sistema operativo elige una página para sacarla de la memoria y hacer sitio para traer la página necesaria. Si la página reemplazada ha sido modificada, se debe reescribir en la copia del disco. La página que se trae a memoria central se sobrescribe sobre la página sacada.

Por tanto, hay que sustituir una página de memoria que no se utilice mucho. Veamos los algoritmos más interesantes:

4.4.1. Reemplazamiento de la página óptima

Cada página se etiqueta con el número de instrucciones que serán ejecutadas antes de referenciar esa página. La página a reemplazar será aquella con el valor de la etiqueta más alta.

Es imposible de implementar ya que el sistema operativo no tiene forma de saber cuando cada página será referenciada próximamente cuando suceda una interrupción de falta de página.

Lo que se hace es simular el comportamiento de estos algoritmos sobre la información de referencia recogida en una primera ejecución de los programas (los resultados sirven de referencia a los demás algoritmos).

4.4.2. Reemplazamiento de la página no usada recientemente (NRU, Not Recently Used)

A cada página le asociamos 2 bits:

- R: bit de referencia → lo coloca el hardware a 1 en cualquier lectura o escritura en la página.
- M: bit de modificado → lo coloca el hardware a 1 cuando se escribe en una página.

Cuando un proceso empieza, el sistema operativo coloca a 0 esos 2 bits (para todas las páginas). Periódicamente (cada tick de reloj, que suele producirse a intervalos de 20 ms aproximadamente), el bit R se limpia (se pone a 0) para distinguir páginas no referenciadas de las que si lo han sido.

En una falta de página, el sistema operativo examina los bits R y M de todas las páginas y saca una página al azar de la clase más baja no vacía:

Clase 0: no referenciada (R=0), no modificada (M=0)
Clase 1: no referenciada (R=0), modificada (M=1)
Clase 2: referenciada (R=1), no modificada (M=0)
Clase 3: referenciada (R=1), modificada (M=1)

El algoritmo tiene un comportamiento adecuado.

4.4.3. Reemplazamiento primera en entrar, primera en salir (FIFO)

- El sistema operativo mantiene una lista de las páginas concurrentes en memoria, siendo la página de cabeza de la lista la más antigua y la última de la lista, la última en llegar. Ante una falta de página, se reemplaza la página de cabeza y la nueva página se añade al final de la lista.

  Inconveniente → La página más vieja puede ser muy usada por las otras.
  Ejemplo: Páginas compartidas (código reentrante).

- Dos variantes FIFO usando bits R y M:
  1. Extraer la página de la clase más baja empezando a buscar la más vieja.
  2. "second chance" → Se inspecciona la página más vieja como víctima potencial. Si su bit R esté a 0, se reemplaza y si está a 1 se coloca a 0 y se pone la página al final de la lista buscando en la siguiente página de la lista (si todas las páginas han sido referenciadas tendremos el FIFO puro).

Cuantas más páginas tiene la memoria, menos faltas de páginas tiene un programa → Anomalía de Belady: FIFO causa más faltas de página con 4 páginas que con 3 (es una curiosidad, no es un resultado importante).

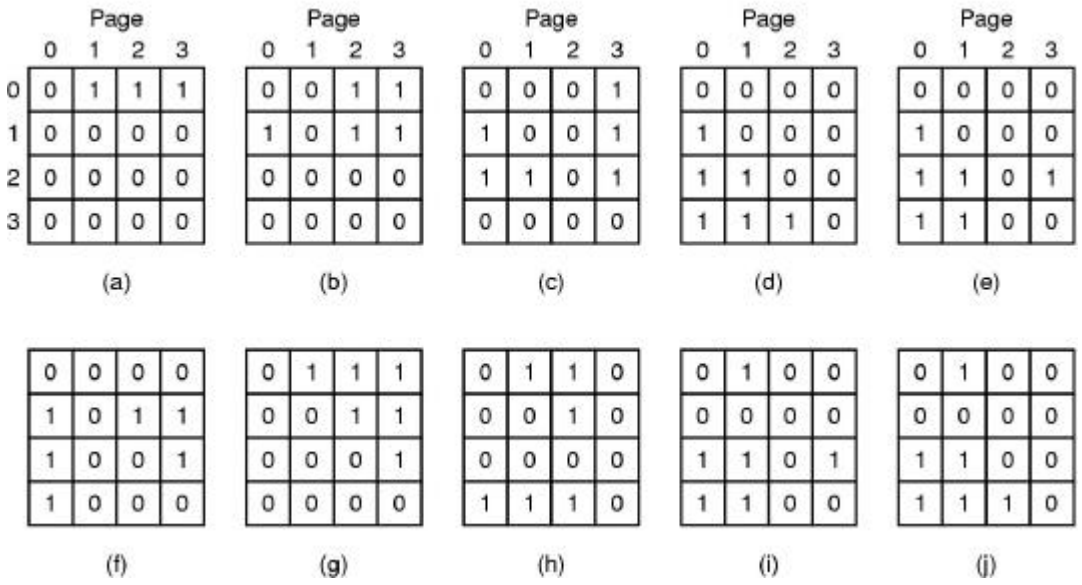
4.4.4. Reemplazamiento de la página menos usada recientemente (LRU, Least Recently Used)

Las páginas que se han usado mucho en la última instrucción probablemente serán usadas mucho en la próxima instrucción, y las que no han sido usadas durante muchas instrucciones no lo serán durante mucho tiempo.

Algoritmo difícil de implementar → se necesita una lista encadenada, ordenada según tiempo de la última referencia, de todas las páginas en memoria, con las más usadas el principio y las menos usadas al final. Esa lista hay que procesarla en cualquier referencia a memoria (se consume mucho tiempo, con lo cual se necesita de un hardware especial).

Dos ejemplos de implementaciones particulares:

1. Hardware viene equipado con un contador de 64 bits, C, que se auto incrementa en cada instrucción. Cada entrada a la tabla de páginas debe tener un gran campo que contenga el contador. Después de cada referencia a memoria, el valor actual de C se almacena en la entrada a la tabla de páginas para la página referenciada. Cuando sucede una falta de página el sistema operativo saca la página que tenga el valor más pequeño en su contador de la tabla de páginas.
2. Para N páginas físicas, el hardware LRU debe mantener una matriz de NxN bits, todas a 0 al principio. Al referenciar una página k, el hardware coloca todos los bits de la fila k a y todos los bits de la columna k a 0. La página menos usada recientemente es la que tiene la fila con el valor binario más bajo.



LRU que usa una matriz

4.4.5. Simulando LRU en software (Algoritmo NFU, No Usada Frecuentemente – Not Frequently Used)

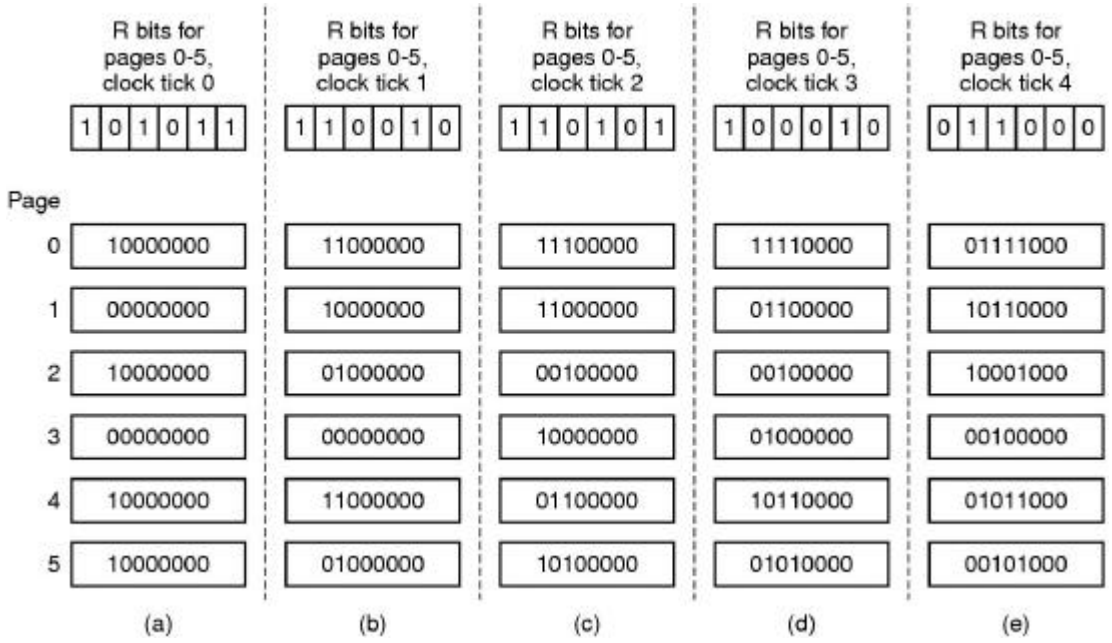
Algoritmo de reemplazamiento de la página no usada frecuentemente.
Requiere de un contador de software asociado con cada página e inicializado a 0. Este contador contiene las referencias a las páginas. En cada interrupción de reloj, el sistema operativo añade el bit R (0 a 1) el contador. Cuado ocurre una falta de página, la página con el contador más bajo es reemplazada.

Inconveniente → En un compilador de varias pasadas, las páginas muy usadas en la pasada 1 pueden no ser reemplazadas en detrimento de otras páginas útiles para pasadas subsecuentes. **(El algoritmo nunca olvida)**

Modificación →

Algoritmo de "aging" (envejecimiento)

- Los contadores se desplazan a la derecha un bit.
- El bit R se añade al bit más izquierdo.



El algoritmo de envejecimiento simula LRU en software. Se muestran seis páginas para cinco pulsos de reloj. Los cinco pulsos de reloj se representan por los incisos del (a) al (e).

Cuando sucede una falta de página, la página cuyo contador es el más bajo se reemplaza (página que menos ha sido referenciada en los últimos ticks de reloj).

Diferencias del algoritmo de “aging” con el LRU:

- No sabemos cuál fue la última página referenciada (después del tick 3 entre las páginas 3 y 5).
- Los contadores tienen un número finito de bits (8). Sólo tenemos referencias de los últimos 8 ticks de reloj (si cada tick de reloj se produce aproximadamente cada 20 msg., tendremos referencia de los últimos 160 msg., lo cual **puede ser suficiente**).

4.5. Características de diseño para sistemas paginados

Estas características se utilizan para obtener un buen rendimiento de un sistema paginado.

Políticas de carga (criterio de selección):

1. Por demanda → Cargar las páginas cuando sean necesarias. Fáciles de implementar.
2. Anticipatorias → Las cargan por adelantado.

Paginación por demanda:

Un proceso (al iniciar su ejecución) empieza sin ninguna página en memoria, generando una interrupción de "falta de página" que hace que el sistema operativo traiga la página que contiene la primera instrucción. Las restantes páginas se irán cargando a medida que se necesiten y no por adelantado.

Paginación por adelantado:

Hay que prever el comportamiento futuro del programa basándose en su comportamiento pasado y de la construcción del mismo. Para prever el comportamiento, se basa en el *Principio de Localidad (Postulado de Denning)*

Principio de Localidad (Postulado de Denning) →

1. Las referencias de un programa tienden a acumularse en pequeñas zonas del espacio de direcciones, que tienden a cambiar sólo de forma intermitente.
2. Durante cualquier fase de la ejecución de un proceso, sólo se referencia una pequeña fracción de sus páginas. Dos tipos:

    - Localidad Temporal: Las direcciones referenciadas recientemente tienen una alta probabilidad de ser referenciadas en un futuro próximo.
      Ejemplo: Bucles, procedimientos, pilas, etc.

- Localidad Espacial: Las direcciones referenciadas tienden a acumularse de forma que al referenciar una posición, es muy probable referenciar otra próxima.
  Ejemplo: Procesado de vectores, ejecución secuencial de código.

4.5.1. El modelo del conjunto de trabajo (working set)

Conjunto de páginas que un proceso está actualmente usando; o sea a las que se han hecho referencia “recientemente”.

Definición formal:
w (t , h) = {página i / página i aparece en las “h” últimas referencias}

w (t, h) varía lentamente con el tiempo (se deduce del principio de localidad).

Si el conjunto de trabajo está entero en memoria, el proceso se ejecutará sin causar muchas faltas de páginas hasta que entre en otra fase de ejecución. En caso contrario, el programa causará muchas faltas y su ejecución será lenta. Un programa que causa faltas de página cada pocas instrucciones, se dice que se está azotando (thrashing).

Denning demostró que el tamaño de w(h) varía con “h”:

Modelo del conjunto de trabajo

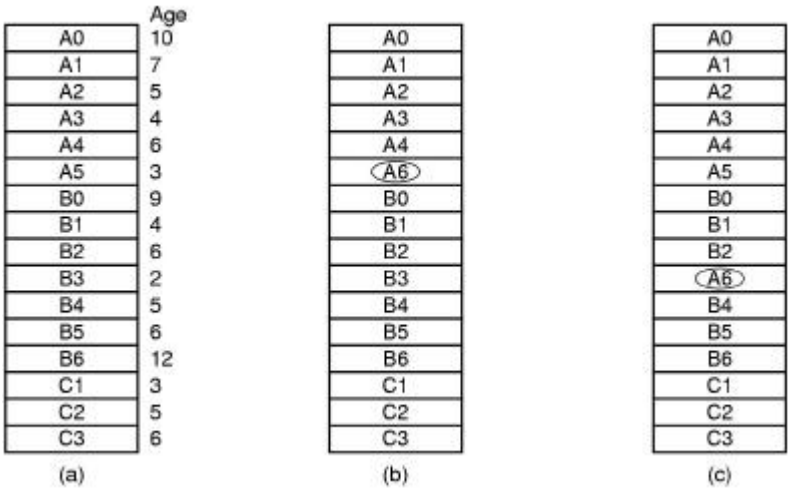
Se carga el conjunto de trabajo entero de cada proceso en memoria antes de hacerlo ejecutable (prepaginado). Se reduce el ritmo de falta de páginas. No se debe de sacar nunca una página que pertenezca al conjunto de trabajo de algún proceso.

El tamaño total del conjunto de trabajo de todos los procesos no debe exceder la memoria disponible (caso contrario se produce el azotado), por tanto, el sistema operativo, debe reducir el grado de multiprogramación (número de procesos en memoria central).

El sistema operativo debe saber qué páginas forman parte del conjunto de trabajo. Si se emplea el algoritmo de "aging", si una página no ha sido referenciada n tick de reloj consecutivos, se saca del conjunto de trabajo.

4.5.2. Políticas de asignación local y global.

¿Cuánta memoria se asigna a los diferentes procesos ejecutables?



¿Quiere entrar la página A_6 ?

Sustitución de páginas local frente a sustitución global.

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

Reemplazamiento de página bcal: A cada proceso se le asigna una cantidad fija de memoria. En la gráfica anterior, se sustituye A_5. Si el conjunto de trabajo crece ocurrirá el azotado (incluso con páginas físicas libres), y si disminuye, se gasta memoria.

Reemplazamiento de página global: Las páginas físicas se asignan dinámicamente a los procesos ejecutables. En la gráfica anterior, se sustituye B_3 . Suele ser mejor, especialmente cuando el conjunto de trabajo varía sobre la media de los procesos. El sistema debe continuamente decidir cuántas páginas física se asigna a cada proceso.

4.5.3. Tamaño de la página.

Fragmentación interna (fragmentación de página)

En promedio, cada proceso desperdicia media página.

N procesos gastan $\frac{n \cdot p}{2}$, siendo p el tamaño de la página. Si las páginas son pequeñas, se desperdicia poca memoria por fragmentación interna, pero se necesitan muchas páginas y por tanto una gran tabla de páginas. Además, transferir, desde disco una página pequeña toma casi el mismo tiempo que una grande.

4.6. Compartición de memoria.

En los sistemas multiprogramados, en especial los de tiempo compartido, es normal que muchos usuarios ejecuten los mismo programas (por ejemplo un compilador). Si se asignaran copias individuales de esos programas a cada usuario, se desperdiciaría mucho almacenamiento primario. La solución obvia consiste en compartir información.

La compartición reduce la cantidad de almacenamiento primario necesaria para la ejecución eficiente de un grupo de procesos y puede hacer posible que un sistema de servicio a más usuarios.

Compartición en un sistema de paginación

La compartición debe controlarse con cuidado con respecto a la protección de la información.

En la mayor parte de los sistemas actuales que realizan compartición, los programas se dividen en áreas separadas para código y para datos. Por tanto, se hace evidente la necesidad de identificar las páginas que se pueden o no compartir.

La forma de compartir páginas es haciendo que ciertas entradas en la tablas de páginas de diferentes procesos apunten a la misma página (o marco de página); con lo cual, varios procesos compartirán esa página.

Compartición en un sistema con segmentación

Una de las ventajas que tiene la segmentación sobre la paginación es que la segmentación es un concepto lógico y no físico.
Un segmento puede estar formado por una matriz, un procedimiento, etc.
Si se quiere compartir una estructura de datos que ocupa tres páginas y media, en paginación se tendrá cuatro entradas apuntando a las correspondientes páginas compartidas (por proceso que lo comparte) mientras que en un sistema de segmentación se tendrá una simple entrada (por proceso) una vez que el segmento se ha declarado como compartido. La situación empeora si la estructura de datos es dinámica ya que la información puede crecer/disminuir y ocupar/desocupar nuevas páginas. Toda esta gestión se tendrá que realizar en tiempo de ejecución en un sistema de paginación mientras que en uno de segmentación puede crecer sin afectarlo.

5. Swapping (Intercambio)

En sistemas en tiempo compartido se puede presentar la situación de que la suma de los tamaños de los procesos de usuario y sistema sea mayor que la memoria principal disponible.

Las técnicas de swapping consisten en intercambiar (momentáneamente) entre memoria principal y disco aquellos procesos (o la parte de los procesos que se tenga en memoria principal) que no se van a ejecutar antes.

NOTA:
Al intercambiarse el programa a disco, sólo se copia la memoria usada (y no toda la asignada).

La zona de disco donde se guardan momentáneamente los procesos se llama *área de swap.*

Consideraciones con el área de swap del disco:

- En algunos sistemas, a un proceso en memoria no se le asigna espacio en disco. Cuando se intercambia, se le debe de asignar espacio en el área de swap del disco. En cada intercambio, podrá ser colocado en cualquier lugar del disco.
- Otros sistemas, al crear un proceso le asignan espacio de swap en disco para cuando se intercambie. Al finalizar el proceso, se libera ese espacio.

El espacio de disco de un proceso se debe asignar como un número entero de bloques de disco.

Con particiones fijas y poca memoria, el swapping es muy poco eficiente.