Administración de Memoria

Objetivos

- Analizar formas de administración de memoria real y virtual
- Comparar algoritmos de asignación de memoria
- Estimar cuanta memoria se requiere

Problemas de la administración de memoria

- Crecen los requerimientos de usuario
- la memoria más rápida es más cara
  - por ello la mayoría de las computadoras tiene una jerarquía de memoria.

¿C

Modelos de administración

- Memoria real
- Memoria virtual

Administración de memoria Real

- Opciones de diseño:
  - Monoprogramación
    - Un programa en ejecución
  - Multiprogramación
    - Muchos programas en ejecución

Monoprogramación

- Carácterísticas
  - Un sólo un programa en la memoria a la vez
  - Comparte la memoria con el sistema operativo (SO)

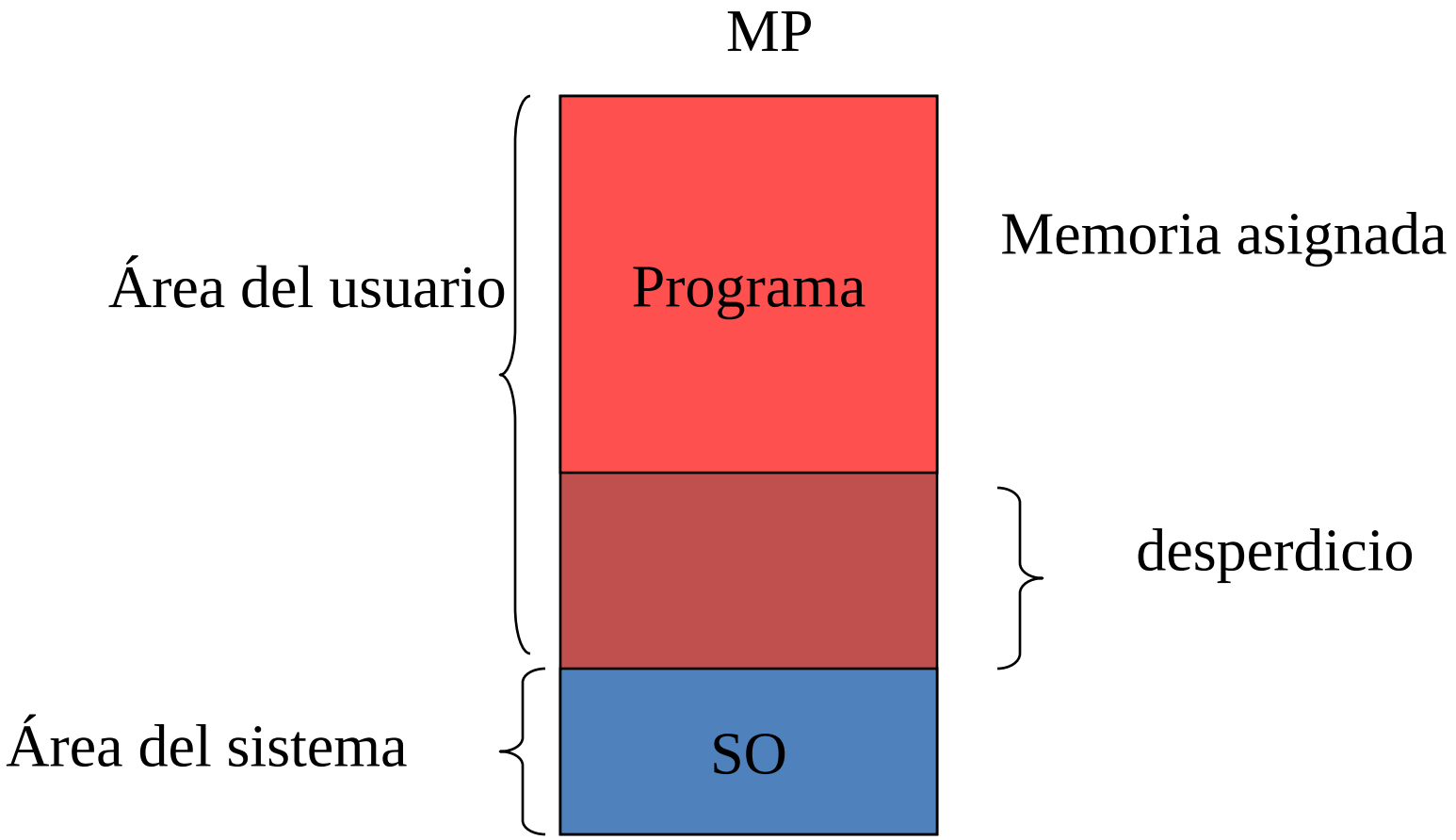
Ejemplo, MS-DOS

- Asigna en direcciones contiguas (crecientes) de memoria:
  - sistema operativo (SO)
  - programa de usuario
  - manejadores de dispositivos (en ROM)
- el SO sube (hace la carga: load) el programa en la memoria y lo ejecuta.
- Cuando el programa termina, el SO solicita un nuevo comando
  - carga el nuevo programa en la memoria, sobreescribiendo el anterior.

Metodología de análisis de un diseño de sistema de memoria

1. Algoritmos de:
   - Asignación:
     - ¿A que programa asignar?
     - ¿Cuanta memoria?
     - ¿Cuando?
   - Liberación
2. Hardware adicional
3. seguridad

Asignación de memoria contigua



Análisis

+ Asignación simple

liberación de memoria simple

+ no requiere hardware adicional para administrar

Problemas:

- alto desperdicio

- mono programación

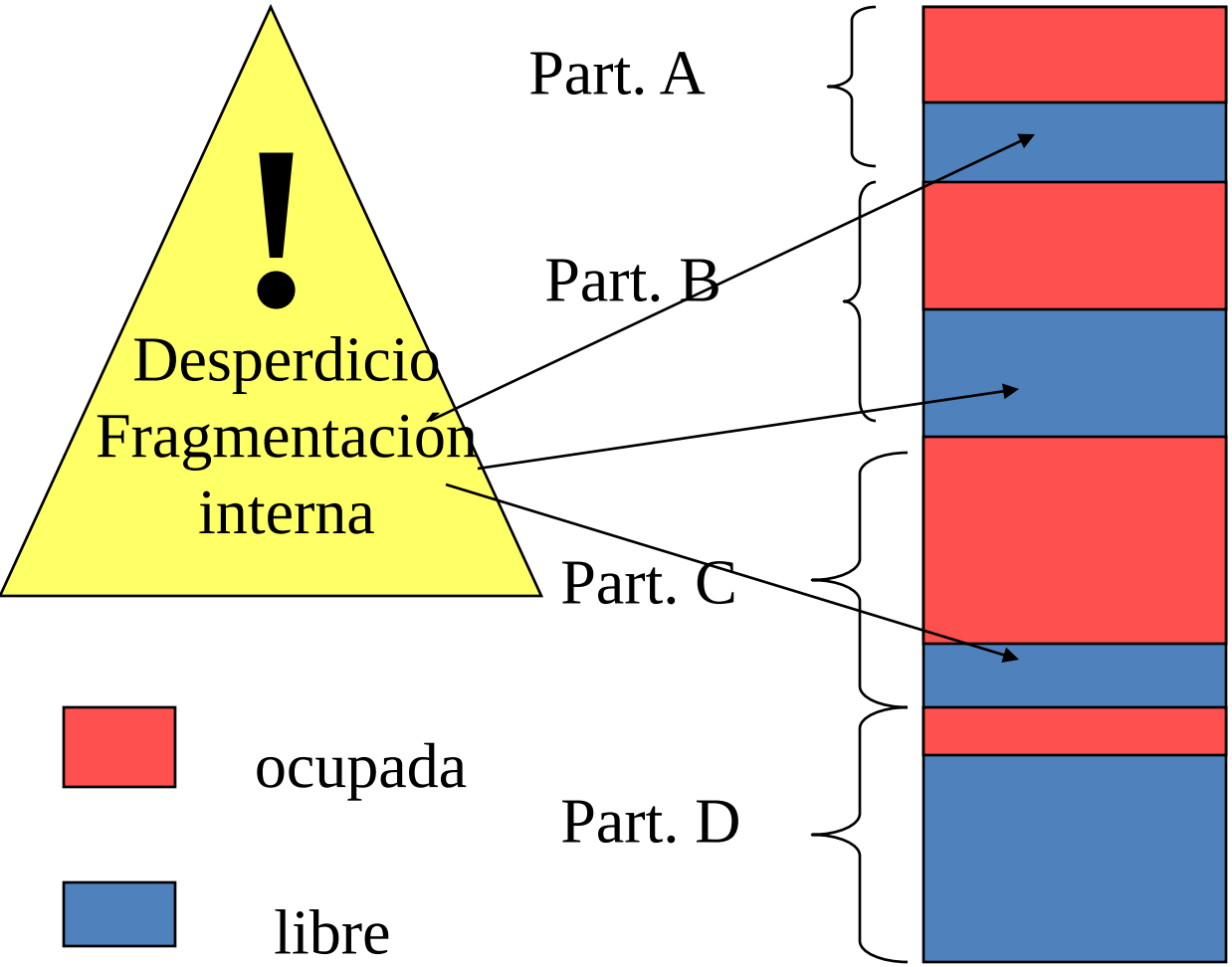
Asignación de memoria particionada

- La monoprogramación usa la CPU ineficientemente
- Solución:
  - Multiprogramación
- La idea es dividir la memoria en múltiples partes:
  - Memoria particionada estática
  - Memoria particionada dinámica

Multiprogramación con particiones fijas

- Las ventajas
  - multiprogramación
    - aumentan la utilización de la CPU
- Concepto:
  - divide la memoria en N particiones fijas y de tamaños no necesariamente iguales

¿Cómo funciona?



Algoritmo de asignación:

Se busca en la lista de particiones libres la partición >= al tamaño solicitado

Algoritmo de liberación:

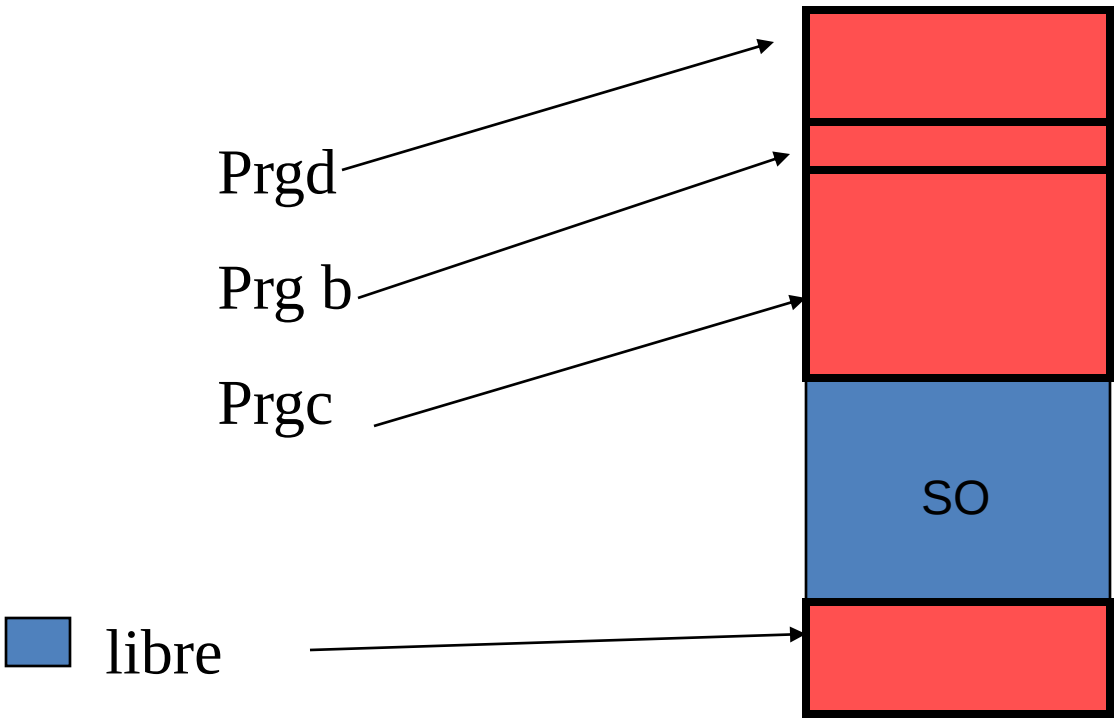
pasar la partición de la lista de part. Ocupadas a lista de part. Libres.

Análisis

- +multiprogramación
- +relativamente simple de administrar
- Seguridad
  - simple
- Problemas:
  - Fragmentación interna (memoria asignada pero no utilizada)

Multiprogramación con particiones variables (MVT)

- Asigna dinámicamente la memoria



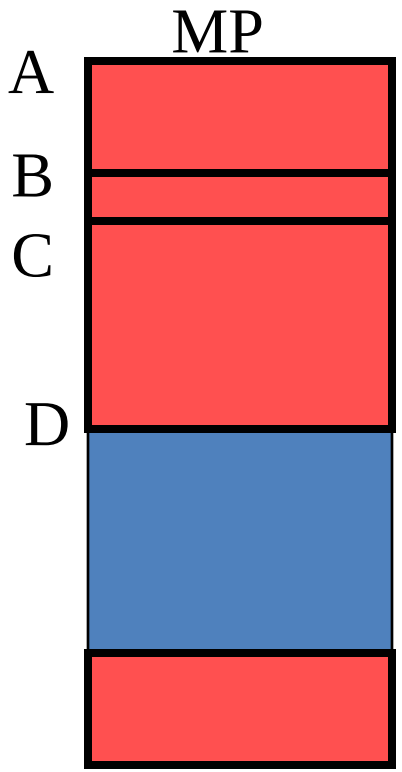
Asignación

Tabla de particiones ocupadas

#par	Dirección de inicio	tamaño
1 2 3	A B C	50 KB 150 KB 150 KB

Tabla de particiones libres

#par	Dirección de inicio	Tamaño
1 2	B D	30 KB 200 KB



Enlace dinámico (dynamic binding)

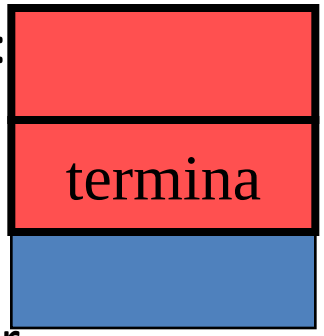
- Las particiones ya no son fijas, si no que van cambiando dinámicamente, tanto en cantidad como en ubicación y tamaño.
- cuando un proceso es pasado a disco (planificador a largo plazo), no hay ninguna garantía de que vuelva a quedar en la misma posición de memoria al traerlo de vuelta, de manera que es imprescindible el apoyo del hardware para hacer enlace en tiempo de ejecución.

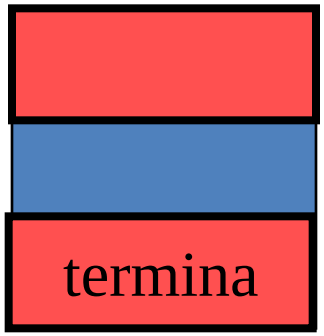
Algoritmos de asignación

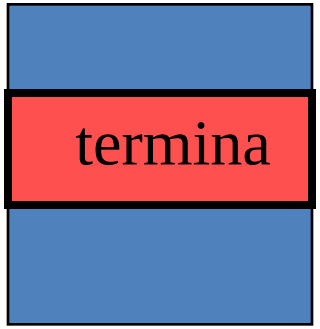
- Por la forma que se ordena la tabla de Part. Libres.
  - Primer ajuste
    - la tabla se ordena por orden creciente de direcciones de inicio
  - Mejor ajuste
    - la tabla se ordena por orden creciente de tamaño de partición
  - Peor ajuste
    - la tabla se ordena por orden decreciente de tamaño de partición
  - Próximo ajuste >> tarea para la casa

Algoritmos de liberación

- Simple de implementar:
- Analiza la tabla de particiones libres para ver si se pueden integrar particiones libres



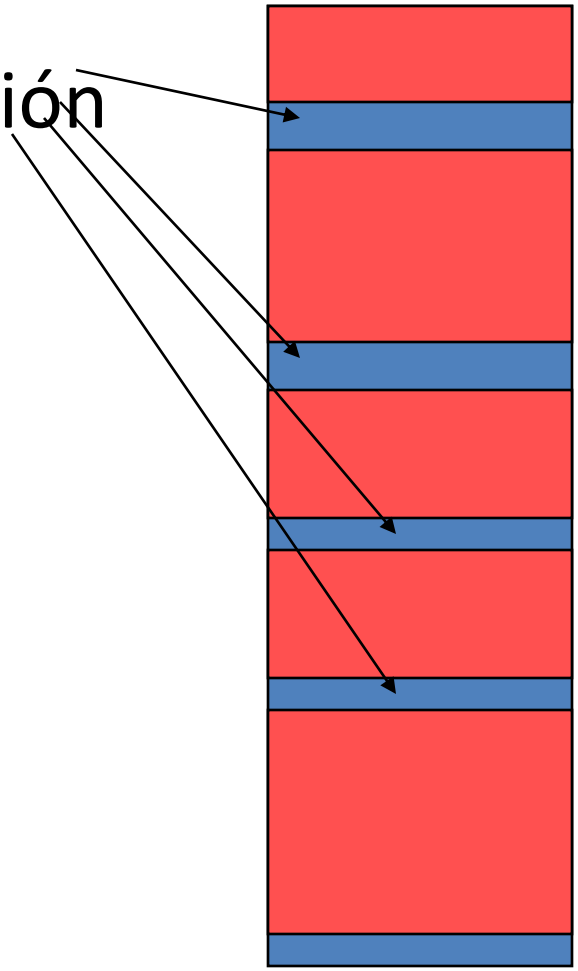




Problema de MVT

- Fragmentación

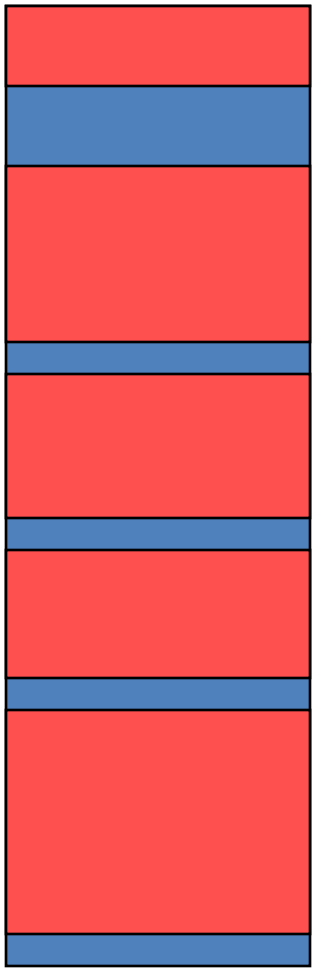
Es posible que haya suficiente espacio libre como para agregar un proceso a la memoria, pero no es contiguo



Solución:
reubicar

Reubicación

- La idea es compactar para obtener una única partición libre
- Es necesario mover los procesos de manera que queden contiguos

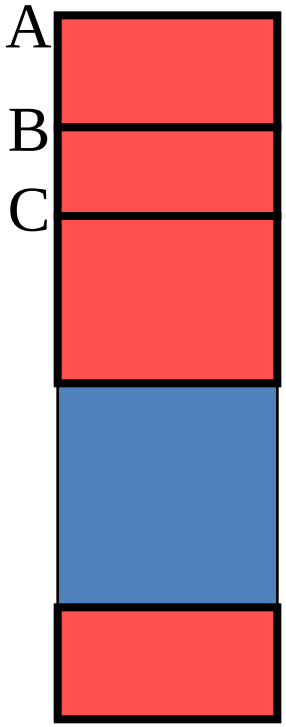


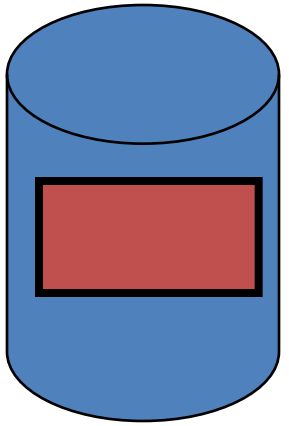
Análisis

- Si no hay una partición suficientemente grande, entonces compactar
- No hay fragmentación
- Problema:
  - El tiempo de compactación puede degradar el tiempo de respuesta de los usuarios
  - Si el tiempo de acceso a la memoria es $t_{acc,}$
  - Para mover cada registro de memoria se requerirán 2tacc (uno para leer y el otro tiempo para escribir)
  - Una maquina de 256 MB, para mover el 50% se requerirá $128*1024*1024*2*60x10^{-9}$ segundos = 15 seg.

Reubicación y swapping

- Si al compilar no se conocen todavía las direcciones absolutas, hay que generar código reubicable
- El cargador o loader, es responsable de determinar las direcciones absolutas al momento de poner el programa en memoria
- Si compactando no hay suficiente espacio en memoria, que hacer?
- Respuesta: bajar un proceso a disco y cargar el nuevo programa (swapping) e.d. pasar procesos a disco para hacer espacio en la memoria





Problema: tiempo de ejecución.

- Si el sistema operativo usa swapping un proceso puede haber cambiado de posición durante su ejecución.
- En este caso se usa hardware especial.

Problema de E/S

- Otro punto que hay que tener en cuenta al usar swappping, es la operación I/O que pudiera estar pendiente.
- Cuando se hace, por ejemplo,scanf(), se especifica una dirección de memoria donde se va a poner lo que se lea desde el dispositivo.

Problemas con la Entrada/Salida

- Supongamos que proceso A trata de leer del disco hacia la dirección d, pero el dispositivo está ocupado: su solicitud, por lo tanto, se pone en la cola.
  - Entretanto, el proceso A es intercambiado a disco y la operación se completa cuando A no está en memoria.
- En esas circunstancias, lo leído se escribe en la dirección d, que ahora corresponde a otro proceso.
- Para evitar el desastre (crash): no pasar a disco procesos con I/O pendiente, o bien hacer siempre I/O desde y hacia buffers del sistema operativo.

Windows 3.x

- Windows 3.x usa swapping, pero un proceso pasado a disco sólo vuelve a la memoria si el usuario activa su ventana
- Windows NT es multihilo
- Unix comienza a usar swapping cuando se supera un cierto límite de utilización de memoria

Seguridad

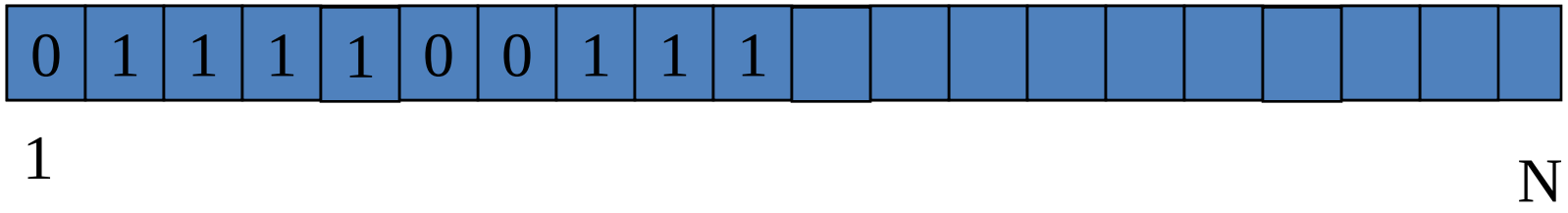
- Sistemas multiproceso
  - un programa ejecutando en una partición no pueda leer ni escribir la memoria que corresponde a otra partición.
- usar un registro base y límite
- Cuando se carga un programa en una partición, se hace apuntar base al comienzo de la partición y límite se fija como la longitud de la partición.

Algoritmo de control

- Cada vez que la CPU quiere acceder a la dirección d, el hardware se encarga de que en realidad se acceda a la dirección física
  - base+**d**
- por supuesto, además se verifica que
  - **d** < límite
- **d** es la dirección lógica o virtual, y base+d la dirección física

Administración de la memoria con mapas de bits

- Dividir la memoria en pequeñas unidades y registrar en un mapa de bits (con tantos bits como unidades haya)



- las unidades ocupadas se marcan con un 1 y las desocupadas con un 0

problema

- Las unidades pueden ser de unas pocas palabras cada una, hasta de un par de KB
- A mayor tamaño de las unidades, menor espacio ocupa el mapa de bits, pero puede haber mayor fragmentación interna
- Desventaja:
  - para encontrar un espacio de N unidades hay que recorrer el mapa hasta encontrar N ceros seguidos

Administración de la memoria con listas enlazadas

- Lista enlazada de segmentos:
  - estado (ocupado o en uso), dirección (de inicio), tamaño.
- Cuando un proceso termina o se pasa a disco
  - si quedan dos particiones juntas, se juntan en un solo segmento.

Memoria Virtual

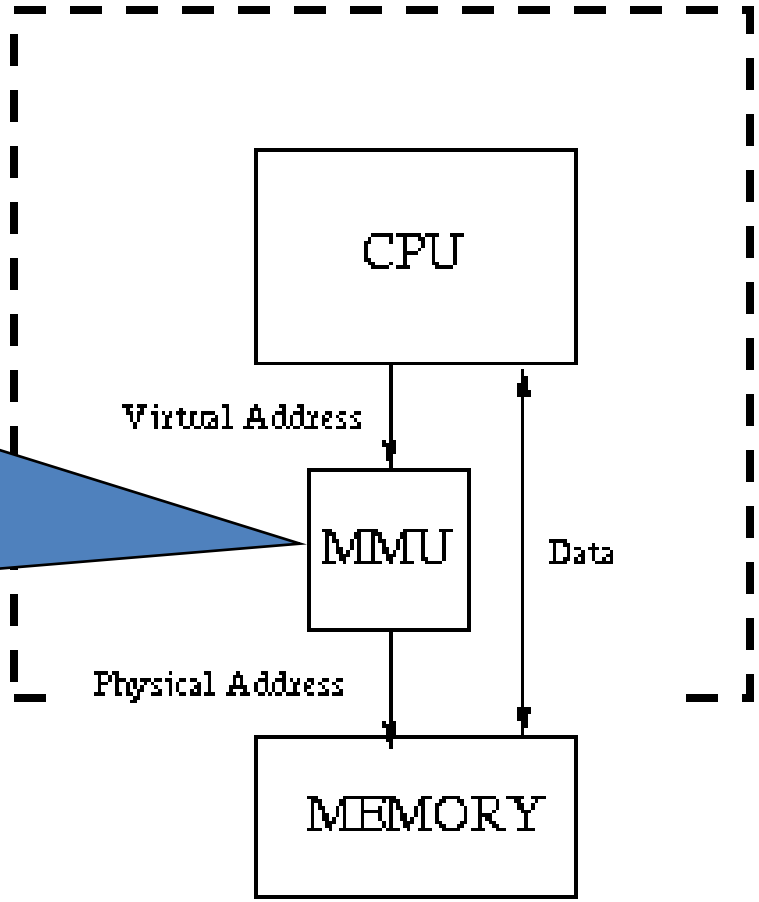
¿Por qué se denomina Memoria Virtual?

- Extender la memoria de manera virtual, es hacer que el proceso tenga la ilusión de que la memoria es mucho más grande que la memoria física.

Hardware de memoria virtual

Problema: el espacio de direcciones de memoria virtual es mayor que el espacio de direcciones de memoria física

Solución: Nuevo Hardware El Memory Management Unit (MMU)



El Memory Management Unit (MMU)

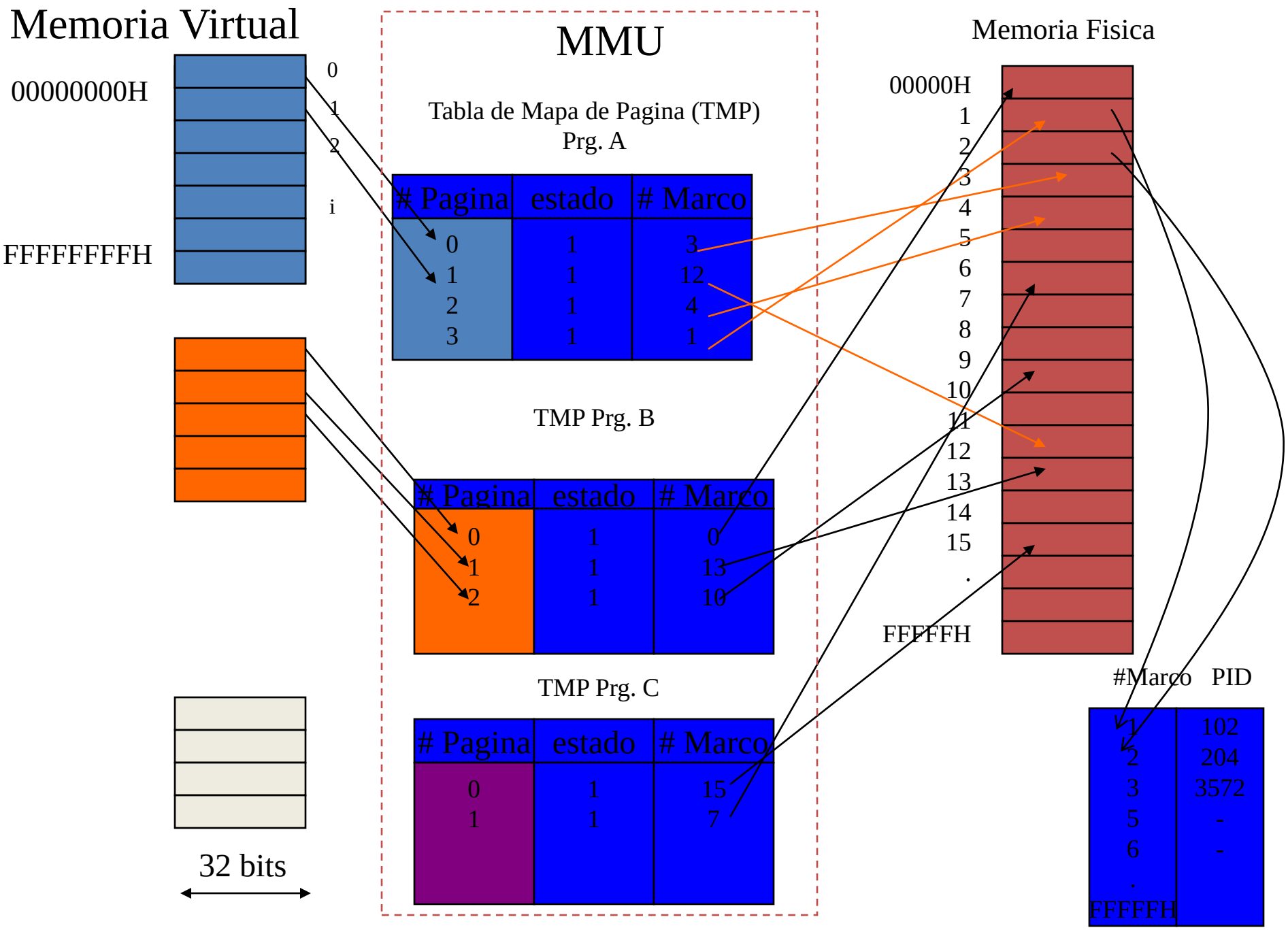
- El programa de usuario sólo ve direcciones lógicas
- La unidad de administración de memoria (MMU) traduce a direcciones físicas.

Problema

- Los procesos tengan que usar un espacio contiguo de la memoria era un impedimento serio para poder optimizar el aprovechamiento de la memoria
- Idea: que las direcciones lógicas sean contiguas, pero que no necesariamente correspondan a direcciones físicas contiguas
- Solución: dividir la memoria física en bloques de tamaño fijo, llamados marcos o frames y dividir la memoria lógica (la que los procesos ven) en bloques del mismo tamaño denominados páginas

Tabla de Mapas de Página (TMP)

- Se usa una tabla de páginas para saber en que marco se encuentra cada página
- Se necesita el apoyo del hardware (MMU)
- Cada vez que la CPU intenta acceder a una dirección, la dirección se divide en número de página **p** y desplazamiento **u** offset **d**
- El número de página se transforma en el frame (marco de página) correspondiente, antes de acceder físicamente la memoria


Memoria Virtual
00000000H
FFFFFFFFH
0
1
2
i
32 bits
MMU
Tabla de Mapa de Pagina (TMP)
Prg. A
Pagina
estado
Marco
0 1 3
1 1 12
2 1 4
3 1 1
TMP Prg. B
Pagina
estado
Marco
0 1 0
1 1 13
2 1 10
TMP Prg. C
Pagina
estado
Marco
0 1 15
1 1 7
Memoria Fisica
00000H
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
.
FFFFFH
#Marco
PID
1 102
2 204
3 3572
5 -
6 -
.
FFFFFH

Tamaño de página

- El tamaño de las páginas es una potencia de 2, típicamente entre 0.5 y 8K.
- Si las direcciones son de m bits y el tamaño de página es 2n, entonces los primeros m-n bits de cada dirección forman p, y los restantes n forman d.

Tablas de página

- Cada proceso tiene su propia tabla, es decir, la tabla de páginas forma parte del contexto:
- Cuando la CPU se concede a otro proceso, hay que cambiar la tabla de páginas a la del nuevo proceso.
- La paginación en general encarece los cambios de contexto.
- La seguridad sale gratis, ya que cada proceso sólo puede tener acceso a las páginas que están en su tabla de páginas.

¿Fragmentación?

- Sólo interna, y de media página por proceso, en promedio.
- Esto sugeriría que conviene usar páginas chicas, pero el problema es que aumenta el sobrecosto de administrar las páginas (entre otras cosas, aumenta el tamaño de la tabla de páginas).
- En general, el tamaño de página ha ido aumentando con el tamaño de la memoria física de una máquina típica.

Tablas de Mapas de Páginas

- Si las direcciones son de m bits y el tamaño de página es 2n, la tabla de páginas puede llegar a contener 2m-n entradas.
- En una CPU moderna, m=32 (y ahora, incluso, 64), y n=12 o 13, lo que significa que se requerirían cientos de miles de entradas: ya no es posible tener la tabla en registros.

Solución

- Manejar la tabla de páginas de cada proceso completamente en memoria y usa sólo un registro que apunte a la ubicación de la tabla.
- Para cambiar de un proceso a otro, sólo se cambia ese registro.
- Ventaja: agregamos sólo un registro a la conmutación de contexto.

Desventaja

- El costo de cada acceso
- La memoria se duplica, porque primero hay que accesar la tabla (indexada por el número de página).
- sin paginación costaba 10ns cada acceso, y ahora cuesta 20!

Solución típica (intermedia)

- Usar un pequeño y rápido caché especial de memoria asociativa, llamado translation look-aside buffer (TLB)
- Memoria Asociativa
  - La memoria asociativa guarda pares (clave, valor) y cuando se le presenta la clave, busca simultáneamente en todos sus registros, y retorna, en pocos nanosegundos, el valor correspondiente.

TLB

- Obviamente, la memoria asociativa es cara; por eso, los TLBs rara vez contienen más de 64 registros
- El TLB forma parte de la MMU, y contiene los pares (página, marco) de las páginas más recientemente accedidas
- Cuando la CPU genera una dirección lógica a la página p, la MMU busca una entrada (p,f) en el TLB
- Si está, se usa marco f, sin acudir a la memoria

TLB en procesadores Intel

- por pequeño que sea el TLB, la probabilidad de que la página esté en el TLB (tasa de aciertos) es alta, debido a que los programas suelen hacer muchas referencias a unas pocas páginas
  - por ejemplo, copiar un arreglo.
- Si la tasa de aciertos es del 90% y un acceso al TLB cuesta 10 ns, entonces, en promedio, cada acceso a memoria costará 87 ns o sea, sólo un 24% más que un acceso sin paginación.
- Una Intel x86 tiene 32 entradas en el TLB
  - Intel asegura una tasa de aciertos del 98%!.

Problema

- En cada cambio de contexto, hay que limpiar el TLB, lo que puede ser barato
  - hay que considerar un costo indirecto
  - si los cambios de contexto son muy frecuentes, la tasa de aciertos se puede reducir

Otras ventajas de la paginación

- Se facilita almacenamiento de procesos en el disco
  - permite que procesos compartan páginas
- Por ejemplo, varios procesos ejecutando el mismo código (que tiene que ser reentrante): las primeras páginas lógicas apuntan a las mismas páginas físicas, que contienen el código
- El resto, apunta a datos locales, propios de cada ejecución

Segmentación

Más cerca del programador

- Cuando se usa paginación, el sistema divide la memoria para poder administrarla, no para facilitarle la vida al programador
- Lo que ocurre es que la vista lógica que el programador tiene de la memoria no tiene nada que ver con la vista física que el sistema tiene de ella

Compartir segmentos

- Paginación:
  - El sistema ve un sólo gran arreglo dividido en páginas
- Cuando uno programa, uno piensa en términos de un conjunto de subrutinas y estructuras de datos
- A las cuales se refiere por su nombre: la función coseno, el stack, la tabla de símbolos, sin importarnos su ubicación en memoria, y si acaso una está antes o después que la otra.

Implementación

- Similar a paginación: en vez de tabla de páginas, hay una tabla de segmentos; para cada programa
- Es necesario saber su tamaño y dónde comienza (dirección base).
- Una dirección virtual (s,d), se traduce a base(s)+d.
- Si d es mayor que el tamaño del segmento, entonces ERROR.

Ventajas:

- Al usuario se le simplifica el manejo de estructuras de datos de tamaño dinámico.
- Se facilita el que los procesos compartan memoria.
- Los segmentos pueden estar protegidos según la semántica de su contenido.

Ejemplo

- un segmento que contiene código, puede especificarse como sólo para ejecución (y nadie puede copiarlo ni sobreescribirlo)
- un arreglo puede especificarse como read/write pero no ejecutable.
- Esto facilita enormemente la detección de errores en el código.
- Librerías compartidas de enlace dinámico (DLLs).

Problema de la segmentación

- Pero la memoria sigue siendo, físicamente, un sólo arreglo de bytes, que debe contener los segmentos de todos los procesos.
- A medida que se van creando y eliminando procesos, se va a ir produciendo, inevitablemente fragmentación externa.

Segmentación y Paginación

Eliminando la fragmentación externa

- Como cada proceso tiene varios segmentos, puede que el problema de la partición externa se reduzca, pues ya no necesita espacio contiguo para todo el proceso, sólo para cada segmento
- Para eliminar completamente el problema de la fragmentación externa, se puede usar una combinación de segmentación y paginación, en la que los segmentos se paginan

Segmentación paginada en Pentium

- En una Pentium
  - 8K segmentos privados
  - 8K segmentos globales
  - compartidos con todos los otros procesos.
- Existe una tabla de descripción local (LDT) y una tabla de descripción global (GDT) con la información de cada grupo de segmentos.
- Cada entrada en esas tablas tiene 8 bytes con información detallada del segmento, incluyendo su tamaño y dirección.

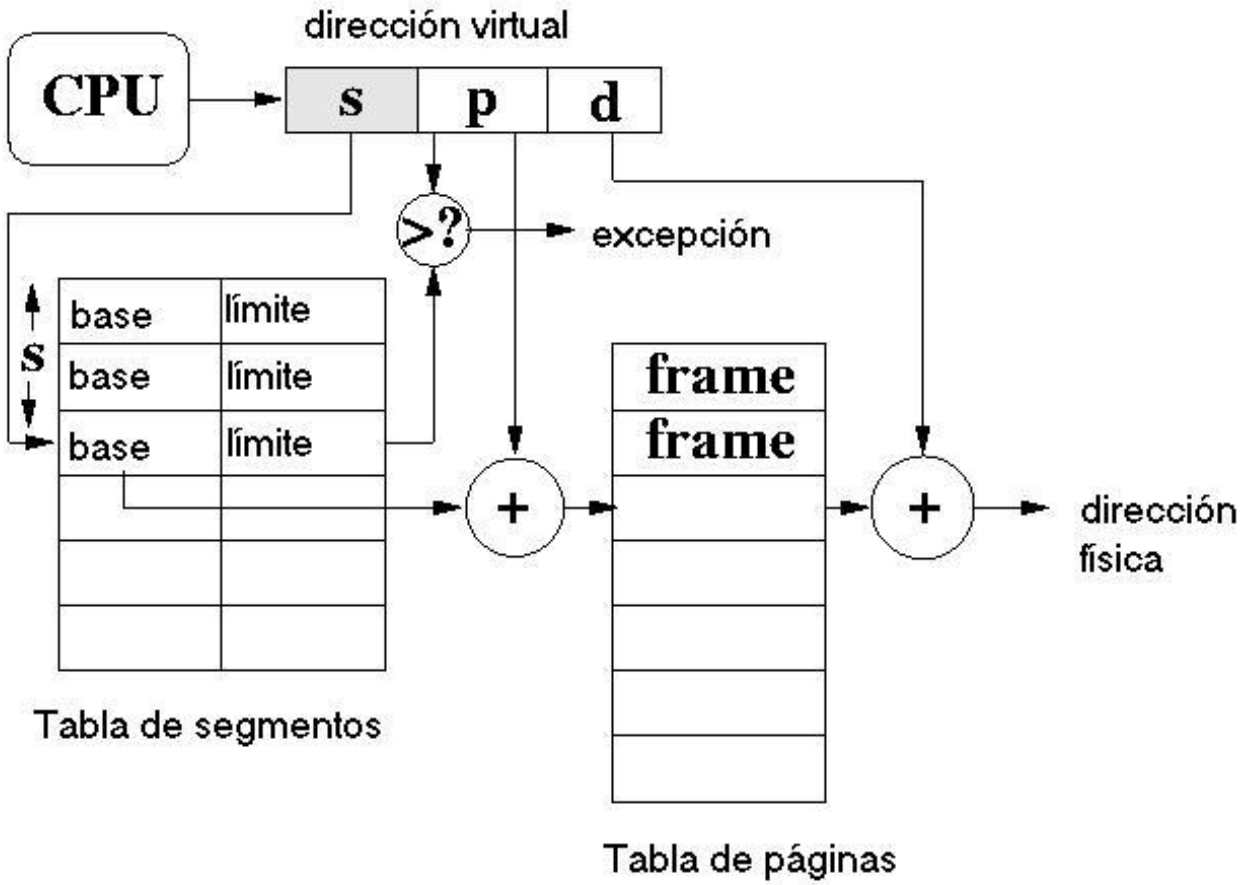
Cache en Intel

- Una dirección lógica es un par (selector, desplazamiento), donde el desplazamiento es de 32 bits y el selector de 16, dividido en:
  - 13 para el segmento, 1 para global/local, y 2 para manejar la protección.
- La CPU tiene un caché interno para los segmentos.

Paginación en Intel

- La memoria lógica puede ser mucho más grande que la física
- Problema:
  - tamaño de la tabla de páginas.
  - Con direcciones de 32 bits, la memoria virtual puede alcanzar los 4GB
  - Con páginas de 4 KB, serían necesarias 512K entradas.
  - Si cada entrada usa 32 bits, entonces la tabla podría ocupar 4 MB de la memoria real.

Segmentación y paginación



Implementación de la paginación

- Problema:
  - cada proceso tiene su tabla y esta tabla debe estar siempre en memoria física.
- Solución:
  - usar tablas de varios niveles.

Paginación a varios niveles

- Sólo se mantiene en memoria la tabla de primer nivel.
- Las otras (que también ocupan exactamente una página), se tratan como páginas ordinarias, es decir, pueden estar en disco.
- Al utilizar la propia memoria virtual para estas páginas, se mantienen en memoria física sólo las más usadas.
- Si las direcciones son de 64 bits, se usan más niveles.

Reemplazo de páginas

- Cuando un proceso hace referencia a una página que no está en memoria, se busca el contenido en disco y se carga en algún marco.
- Problema: no hay ninguno libre
- Solución: usar algún algoritmo de reemplazo de páginas para elegir un marco víctima.
- Una posibilidad es pasar a disco todo un proceso, así los marcos que el ocupaba quedan libres.

Intercambio (swapping)

- Para usar un marco ocupado, primero su contenido se debe pasar a disco.
- Si la página original no fue modificada, la copia en disco es idéntica, así que se puede descartar esta operación de escritura.
- Para saber si una página ha sido modificada se examina un bit asociado a la página (bit de modificado), que el hardware se encarga de poner en 1 cada vez que se escribe en ella.

Interrupciones por excepción de página

- Si una página no está en memoria, entonces se producen transferencias de páginas entre la memoria y el disco.
- Subir o reemplazar supone un gasto de tiempo.
- Un objetivo es reducir las interrupciones por excepción de página (faltas) al mínimo.
- Para comparar algoritmos de reemplazo vale contar cuántas interrupciones se produjeron dadas algunas secuencias de acceso (Trace) a la memoria.

Algoritmos de reemplazo

Algoritmo óptimo

El algoritmo que minimiza el número de faltas consiste en escoger la página para la cual el lapso de tiempo hasta la próxima referencia es el mayor.

El algoritmo PEPS

- Consiste en reemplazar la página más antigua (la que lleva más tiempo en memoria física)
- Es simple, pero no es bueno, ya que la página más antigua no necesariamente es la menos usada.

Anomalía de Belady

PEPS sufre de la Anomalía de Belady:

- las faltas pueden aumentar si aumentamos el número de marcos disponibles

La Menos Usada Recientemente (LRU)

- Se selecciona la página menos recientemente usada con la esperanza de que, si no ha sido accedida en el pasado reciente, no lo será en el futuro cercano.
- Los procesos presentan un patrón denominado **localidad temporal** de referencia.
- Problema: es que la implementación requiere un apoyo del hardware que es caro de implementar.

Implementación

- Por ejemplo, con un **contador** en la CPU y un campo adicional (de unos 64 bits), para registrar el valor acumulado, para cada página.
- Cada vez que se accede a una página, se incrementa el contador y se guarda en el campo de la página accedida.
  - Así la página menos recientemente usada es aquella con el menor valor en su campo.

Aproximaciones a LRU

- La mayoría de las máquinas proveen un bit de referencia por cada página cuya implementación en hardware es barata.
- El sistema operativo pone en 0 el bit de referencia de cada página al inicializar un proceso y el hardware lo pone en 1 cada vez que se accede a la página.

Algoritmo de maduración

- Leer los bits de referencia a intervalos regulares, registrarlos, y poner a cero.
- Si registramos en el bit de mayor orden de un byte, previo desplazamiento a la derecha, e interpretamos el byte como un entero sin signo, entonces hay que reemplazar la página cuyo byte sea el menor es la menos recientemente usada.

Segunda oportunidad

- Es un caso especial del anterior, se usa PEPS con un agregado: si la potencial víctima tiene el bit de referencia en 1, se pone en 0 y se le da una segunda oportunidad (se escoge otra).

El algoritmo de reloj

- Una implementación es con lista circular.
- por eso también se conoce como algoritmo del reloj.

No Usada Recientemente (NRU)

- Una optimización se basa en considerar que sale más caro reemplazar una página que ha sido modificada, pues hay que actualizar la copia en disco antes de reemplazarla.
- Según los bits de referencia y de modificación se clasifican en: 00, 01, 10, 11.
  - Se escoge según PEPS dentro de la clase (no vacía) más baja. La clase 01 significa no referenciada, pero modificada.

Práctica No. 3

- Compararla segmentación vs. Paginación.
- Ejercicios del libro de texto:
- 1,3,4,5,7,13,23,24,25,29,31,37.

- FIN.