

Administración Memoria

Thursday, 2 April 2020 5:33 PM

¿Qué es / qué hace el MMU?

- Casi todos los sistemas operativos modernos requieren de una **unidad de manejo de memoria (MMU)**
 - Hardware, hoy es parte integral del CPU
 - Intermediario hacia el bus de acceso a memoria
- Trabaja *muy* de cerca con el sistema operativo
- Encargado de funciones de control de permisos, seguridad, y traducción de direcciones
- "Vigilando" todos los accesos a memoria que ejecuta el código
- Existen sistemas operativos multitarea que pueden funcionar sin MMU
 - Pagan como precio la confiabilidad del sistema



Gunnar Wolf

Administración de memoria: Funciones y operaciones

Introducción
Espacio de direccionamiento
El MMU
Espacio en memoria
Resolución de direcciones

¿Traducción de direcciones?

- Es común que un sistema requiera más memoria de la que está **directamente disponible**
 - Más memoria de la que existe físicamente
 - Más memoria de la que el hardware puede direccionar
- Un proceso no tiene por qué conocer los detalles de la asignación de memoria — Le damos una vista virtual simplificada
 - ¿Cuántos bloques de memoria me asignaron?
 - ¿Cuál es la ubicación de cada uno de ellos?
 - ¿Qué pasa si intento escribir (o leer) de donde tengo prohibido?
- A lo largo de esta unidad iremos viendo las estrategias para responder a esto.



¿Qué es la *resolución de direcciones*?

- Cuando escribimos un programa, sus funciones y variables son referidas por *nombre*
- El compilador va substituyendo los nombres por la *dirección en memoria* a donde debe referirse
 - Excepto en bibliotecas de ligado dinámico... Que abordaremos más adelante
- Pero en un sistema multiproceso, el compilador no necesariamente sabe dónde estará el espacio de memoria asignado al proceso
- Las direcciones indicadas en el texto del programa deben ser *traducidas* (o *resueltas*) a su ubicación definitiva
- Esto puede ocurrir en tres momentos...



Gunnar Wolf Administración de memoria: Funciones y operaciones

Introducción
Espacio de direccionamiento
El MMU
Espacio en memoria
Resolución de direcciones

En tiempo de compilación

- El texto de programa tiene la dirección *absoluta* de datos y funciones
- Muy común en arquitecturas no multiprocesadas
- En las PC tempranas, el formato ejecutable .COM es un *volumen de memoria* con las direcciones absolutas
 - Formato limitado a *un segmento* de memoria (64K — 16 bits, ¿Recuerdan?)
- Hoy en día vemos esto en sistemas embebidos, microcontroladores, o de función específica
 - p.ej. Arduino
- P.ej. la variable contador queda traducida en la imagen en disco como su dirección: 510200



En tiempo de carga

- El sistema operativo tiene una rutina llamada *cargador* (*loader*)
 - Le asiste *frecuentemente* el *ligador* (*linker*), para incluir en la sección de texto todas las bibliotecas externas que requiera
- Analiza el texto del programa que va cargando, y actualiza a las referencias a memoria para apuntar al lugar correcto
 - Agregando el *desplazamiento* (*offset*) necesario (la dirección *base*)
- Depende de que el compilador indique la ubicación de cada una de las variables y funciones



En tiempo de carga

- El sistema operativo tiene una rutina llamada **cargador (loader)**
 - Le asiste frecuentemente el **ligador (linker)**, para incluir en la sección de texto todas las bibliotecas externas que requiera
- Analiza el texto del programa que va cargando, y actualiza a las referencias a memoria para apuntar al lugar correcto
 - Agregando el **desplazamiento (offset)** necesario (la dirección **base**)
- Depende de que el compilador indique la ubicación de cada una de las variables y funciones
- P.ej. contador queda traducida como un desplazamiento: inicio + 5986
 - Al cargarse, inicio es **resuelto** a 504214



Gunnar Wolf

Administración de memoria: Funciones y operaciones

Introducción
Espacio de direccionamiento
El MMU
Espacio en memoria
Resolución de direcciones

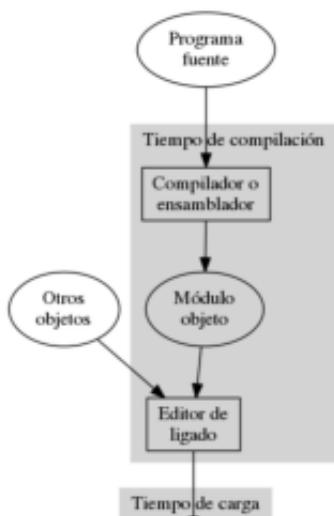
En tiempo de ejecución

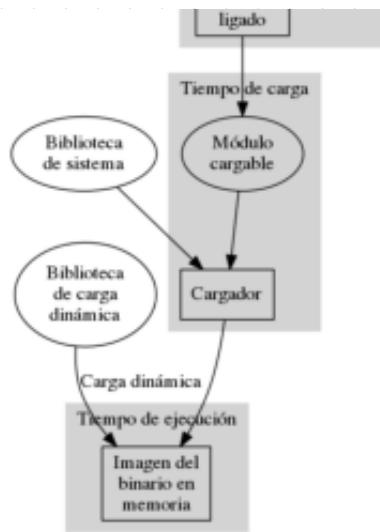
- El programa **nunca hace referencia a ubicaciones absolutas de memoria**
 - El código generado por el compilador siempre indica **base** y **desplazamiento (offset)**
- Permite que el proceso sea reubicado en la memoria *incluso estando ya en ejecución*
- Requiere de apoyo en hardware (MMU)
- P.ej. la variable contador queda traducida como un desplazamiento: inicio + 5986
 - inicio se mantiene como etiqueta durante la ejecución y es resuelta cada vez que se requiera



dll

↳ **Biblioteca de carga dinámica**





Asignación Memoria

Thursday, 2 April 2020 6:23 PM

Compartiendo la memoria desde...

- Como en tantos otros temas, comenzemos viendo cómo compartían la memoria los *primeros* sistemas multiprocesados
- Las primeras implementaciones siempre son las más sencillas
 - ...y las más ingenuas
- ¿Cómo eran estos *primeros* sistemas?
 - Poca memoria
 - Sin MMU
 - No interactivos



Gunnar Wolf

Administración de memoria: Asignación de memoria

Memoria contigua
Segmentación
Paginación

Particiones fijas

- Primer acercamiento: *Partir la memoria en varios bloques*
 - Originalmente del mismo tamaño (¡más sencillo!)
 - Por ejemplo: En 512KB de memoria física caben el sistema operativo mas otros 7 programas de 64KB (16 bits) cada uno
- El sistema operativo típicamente usa la *región más baja*, a partir de 0x0000
 - La *memoria mapeada* a los diversos dispositivos queda dentro del segmento del SO

Particiones fijas

512K	Libre	0x80000
448K	Proceso 5	0x70000
384K	Proceso 3	0x60000
320K	Libre	0x50000
256K	Libre	0x40000
192K	Proceso 2	0x30000
128K	Proceso 1	0x20000

Particiones fijas



Figura: Particiones fijas, con 3 particiones libres

Gunnar Wolf Administración de memoria: Asignación de memoria

Memoria contigua
Segmentación
Paginación

Particiones fijas: Ventajas y desventajas

- **¿Ventajas?** Principalmente, simplicidad
 - Resolución de direcciones en tiempo de carga
 - Registro base (no requiere siquiera de un registro límite)
 - Puede limitarse simplemente con un espacio de direccionamiento acorde en el compilador
- **¿Desventajas?** Rigidez
 - Grado de multiprocesamiento limitado
 - Si hay menos de 7 procesos, se desperdician recursos
 - Si hay más de 7, tienen que esperar a que se les abra espacio
 - Desperdicio de espacio (fragmentación interna)
 - Al asignarse la memoria en bloques fijos, un proceso pequeño podría desperdiciar mucho espacio



Particiones flexibles

- Cada proceso declara sus requisitos de memoria al iniciar su ejecución
 - Debe indicar su uso máximo previsto de memoria
 - Hay mecanismos para ajustar el tamaño de un proceso preexistente — ¡Pero pueden fallar! (p.ej. si falta memoria para satisfacer una solicitud)
- El OS tiene acceso directo a toda la memoria como un continuo
- Cada región en memoria está limitada (ahora sí) por un

continuo

- Cada región en memoria está limitada (ahora sí) por un registro base y un registro límite



Memoria contigua
Segmentación
Paginación

Particiones flexibles

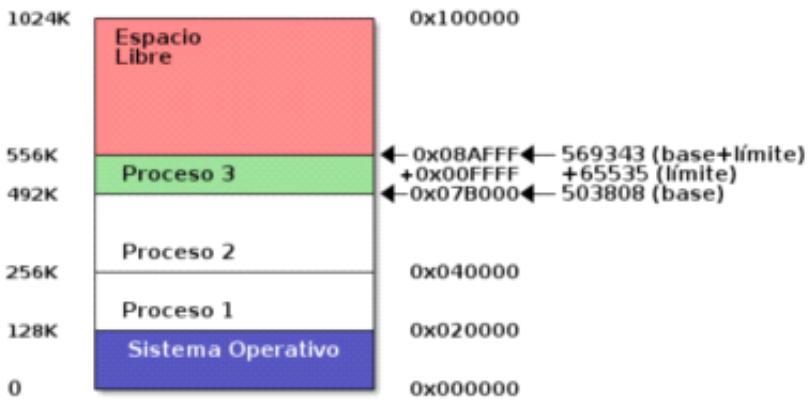


Figura: Espacio de direcciones válidas para el proceso 3 definido por un registro base y un registro límite



Particiones flexibles: Ventajas y desventajas

• ¿Ventajas? Sigue siendo: Simplicidad

- Cuando inicia la ejecución del sistema, este esquema parece ideal
- Sobrecarga mínima, con un MMU muy básico
- Cada proceso puede direccionar el total de memoria disponible

• ¿Desventajas? Vienen con el tiempo...

- Conforme van iniciando y terminando los procesos, se van creando agujeros en la asignación de memoria
- Según análisis estadístico (Silberschatz, p.289), por cada N bloques asignados se pierden del orden de $0,5N$ por fragmentación



Gunar Wolf Administración de memoria: Asignación de memoria

Memoria contigua Segmentación Paginación

Fragmentación en la memoria



Figura: Termina el proceso 1 (de 128K); inician 4 (de 64K), 5 (de 156K) y 6 (32K). Se va fragmentando la memoria libre.

↳ Entre procesos hay agujeros

Fragmentación interna y externa

Fragmentación interna Espacio desperdiciado dentro de la memoria asignada a un proceso

- Porque tuvo que solicitar *toda la memoria que emplearía* desde un principio (y la desperdicia la mayor parte del tiempo)
- Por tener que *alineararse* a cierta frontera de memoria (p.ej. con particiones pre-establecidas)

Fragmentación externa Espacio de memoria desperdiciado entre los distintos fragmentos

- En el esquema anterior, hay 320K libres, pero no puede lanzarse ningún proceso > 192K, porque no es un bloque *contiguo*



Memoria contigua
Segmentación
Paginación

¿Cómo ubicar un nuevo proceso?

Hay tres estrategias principales para dar espacio en la memoria a un nuevo proceso:

Primer ajuste Asigna al nuevo proceso al *primer bloque* de tamaño suficiente

Mejor ajuste Asigna al nuevo proceso al *bloque más chico* en que quiepa

Peor ajuste Asigna al nuevo proceso al *bloque más grande* que haya disponible

¿Qué ventajas / desventajas? puede tener cada uno?



- 1º Ajuste → Más rápido
- 2º Ajuste → Desperdicio menor
 - ↳ Bloques pequeños
- 3º Ajuste → Revisión de bloques

Compactación

- Independientemente del esquema que elijamos, bajo particiones flexibles se irá fragmentando cada vez más la memoria
 - Si no se hace nada al respecto, no podrán lanzarse procesos nuevos
- La compactación consiste en:
 - Suspender temporalmente a un proceso
 - Moverlo a otra dirección de memoria
 - Ajustar su registro base
 - Continuar con el siguiente, hasta crear un sólo bloque disponible (de los muchos existentes)
- Tiene un costo alto, porque requiere:
 - Muchas transferencias de memoria
 - Suspensión sensible de los procesos implicados



Compactación

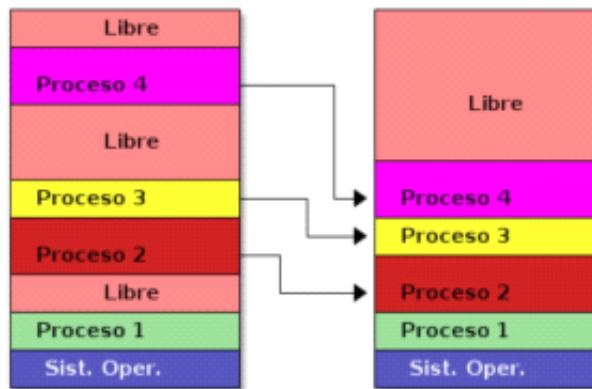


Figura: Compactación de la memoria de procesos en ejecución

¿Cuándo compactar?

No hay una sola respuesta

- Basado en umbrales, verificando periódicamente el estado del sistema
- Basado en eventos, cada vez que no pueda satisfacerse una solicitud por haber demasiada fragmentación

Señales que indican necesidad de compactar

- Relación entre el número de bloques libres y ocupados
- Relación entre la memoria total disponible y el tamaño del bloque más grande
- ...



Intercambio (swap)

- El SO puede comprometer más memoria de la que tiene disponible
- Cuando inicia un sistema que *no cabe en memoria*, puede elegir suspender a un proceso y grabarlo a almacenamiento secundario
 - Por ejemplo, un proceso que esté bloqueado esperando un bloqueo externo

secundario

- Por ejemplo, un proceso que esté bloqueado esperando un bloqueo externo
- ¿Qué pasa con las operaciones E/S que tiene pendientes el proceso?
 - Puede exigirse que sólo se pueda hacer E/S empleando buffers en el espacio del SO



Segmentación

¿Cómo es la visión del programador?

- El trabajo del compilador es traducir lo que ve/entiende el programador a algo que pueda entender la computadora
- Para el programador, la memoria no es un *espacio contiguo*, sino que hay separaciones muy claras
 - El programador no tiene por qué ver relación entre las secciones de texto y datos
 - No tiene por qué preocuparse de la cercanía entre el *espacio de libres* y la pila
- No tiene por qué importarle la estructura representada en la pila
- Las bibliotecas externas enlazadas son meras *cajas negras*



Gunnar Wolf

Administración de memoria: Asignación de memoria

Memoria contigua
Segmentación
Paginación

Traduciendo la visión del programador

- ... ¿No podría traducirse esta separación a algo generado por el compilador?
 - Y que, de paso, pueda aprovechar el sistema...
- El espacio de un proceso se traduce en *varios segmentos* en memoria
 - En vez de sólo un registro *base* y un registro *desplazamiento*, requerimos de uno por segmento
 - Una *tabla de segmentos* por proceso.
 - Típicamente, un juego de *registros especiales* en el CPU
 - La resolución de direcciones es análoga a la descrita anteriormente, con apoyo del MMU
 - El direccionamiento se hace *explícitamente* indicando segmento y desplazamiento



- El direccionamiento se hace *explicitamente* indicando segmento y desplazamiento



Podemos pedir incluso más de lo que nos permite

Paginación

Tuesday, 14 April 2020 5:38 PM

Evitando la fragmentación externa

- La paginación nace para evitar *definitivamente* la fragmentación externa
 - Y, por tanto, la necesidad de compactación
- Requiere hardware más especializado y dar seguimiento a mucha más información
- Simplifica fuertemente las cosas desde el punto de vista del proceso
 - A costo de complicarlas para el SO



Gunnar Wolf Administración de memoria: Asignación de memoria

Memoria contigua Segmentación Paginación

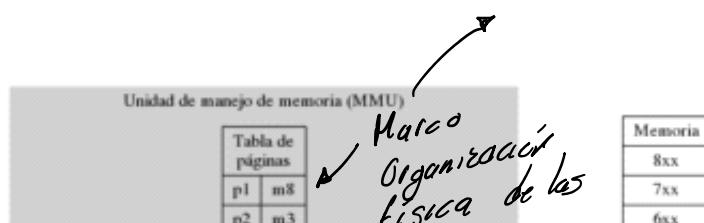
Adios a los registros base+desplazamiento

- Al proceso se le muestra únicamente una *representación lógica* de la memoria
- Para cada proceso, el espacio de direccionamiento comienza en 0, y llega hasta el máximo posible de la arquitectura
- La memoria está dividida en una serie de *páginas*, todas ellas del mismo tamaño
 - Históricamente desde 512 bytes (2^9)
 - Típicamente 4 u 8K (2^{12} o 2^{13})
 - Máximos hoy, 16MB (2^{24})
- La memoria asignada ya no requiere ser contigua
 - A pesar de usar un *modelo plano* (no segmentado)



PM, Tue 14 Apr

Esquematización del proceso de paginación



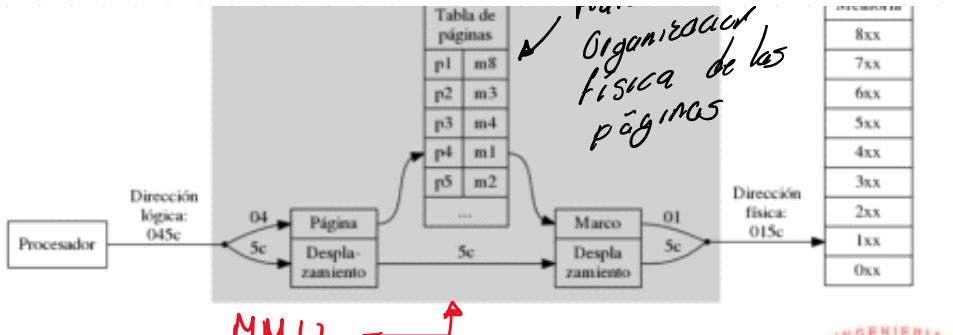
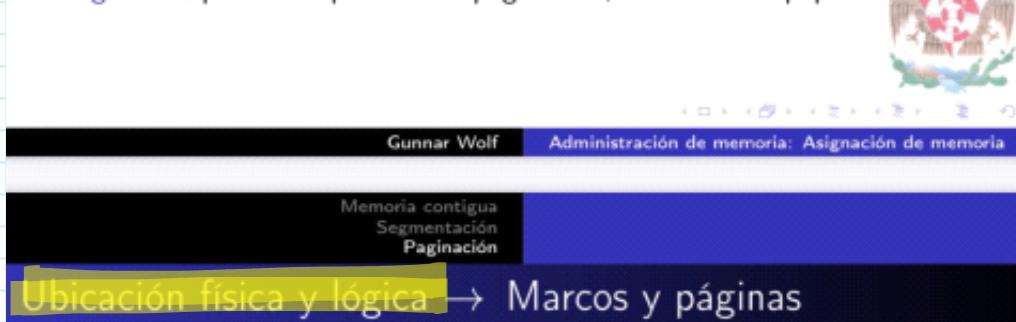


Figura: Esquema del proceso de paginación, ilustrando el papel del MMU



- Cada página corresponde a un *marco (frame)* en la memoria física
 - Relacionados a través de *tablas de páginas*
 - Los marcos *siempre* miden lo mismo que las páginas
 - El tamaño siempre será una potencia de 2
- La ubicación real de un *marco de memoria* es su *ubicación física*
- La dirección que conoce el proceso es su *ubicación lógica*
- La porción *más significativa* de una dirección de memoria indica la *página*; la *menos significativa*, el *desplazamiento*

Ejemplo de página y desplazamiento

Dirección especificada — 0001 1101 0111 0010

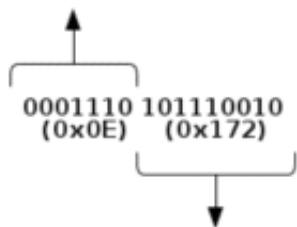


Figura: Página y desplazamiento, en un esquema de direccionamiento de 16 bits y páginas de 512 bytes

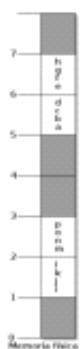
A screenshot of a presentation slide. The title bar at the top reads "Gunnar Wolf" and "Administración de memoria: Asignación de memoria". Below the title, there is a navigation bar with icons for back, forward, search, and other presentation controls. The main content area has a dark blue header with white text that says "Memoria contigua", "Segmentación", and "Paginación". Below this, a large white area contains the text "La tabla de páginas (page table)". In the bottom right corner of the slide, there is a small watermark or logo featuring a coat of arms.

- Guarda la relación entre cada *página* y el *marco* correspondiente
- El MMU no puede ya operar basado en unos cuantos registros
 - En un esquema limitado como el recién presentado, tenemos hasta 128 páginas (2^7)
 - Cada entrada requiere 14 bits (7 para la página, 7 para el marco)
 - → 1792 bytes (en un sistema bastante primitivo)
- El manejo de una tabla de páginas resulta en una suerte de resolución en tiempo de ejecución
 - Pero realizada por el MMU, transparente al programa (e incluso al procesador)
 - Con una *dirección base* distinta para cada una de las páginas

Ejemplo (minúsculo) de tabla de páginas

255	0
254	1
253	2
252	3
251	4
250	5
249	6
248	7
247	8
246	9
245	a
244	b
243	c
242	d
241	e
240	f
239	g
238	h
237	i
236	j
235	k
234	l
233	m
232	n
231	o
230	p
229	q
228	r
227	s
226	t
225	u
224	v
223	w
222	x
221	y
220	z

3	2
2	3
1	4
0	5



- Direcciónamiento: 5 bits
- Página: 3 bits
- Desplazamiento: 2 bits
- Para referirse a la letra F.



Figura: Ejemplo de paginación
(Silberschatz, p.292)

Gunnar Wolf

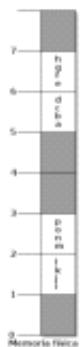
Administración de memoria: Asignación de memoria

Memoria contigua
Segmentación
Paginación

Ejemplo (minúsculo) de tabla de páginas

255	0
254	1
253	2
252	3
251	4
250	5
249	6
248	7
247	8
246	9
245	a
244	b
243	c
242	d
241	e
240	f
239	g
238	h
237	i
236	j
235	k
234	l
233	m
232	n
231	o
230	p
229	q
228	r
227	s
226	t
225	u
224	v
223	w
222	x
221	y
220	z

3	2
2	3
1	4
0	5



- Direcciónamiento: 5 bits
- Página: 3 bits
- Desplazamiento: 2 bits
- Para referirse a la letra F.



Figura: Ejemplo de paginación

Efecto ante la fragmentación

- La fragmentación externa desaparece
 - O más bien, nos deja de preocupar
- ¿Y la fragmentación interna?
- Al dividirse la memoria en bloques de 2^n bytes, cada proceso desperdiciará en promedio $\frac{2^n}{2}$ bytes
 - Peor caso: $2^n - 1$ bytes
- Podemos tener cientos o miles de procesos en el sistema
- Respuesta lógica: Hagamos a n tan pequeño como sea posible
- Muchas páginas, muy pequeñas... ¿O no?



Impacto del tamaño de las páginas

¿En qué se expresaría la *sobrecarga administrativa* de tener páginas demasiado pequeñas?

- Mayor carga en un cambio de contexto
 - Crecimiento del PCB
- Mayor número transferencias DMA requerido (p.ej. para leer/escribir del disco)
- Disminución de velocidad de *todas* las transferencias a memoria



¿Cuánto ocuparía en una computadora real actual?

59 of 83

Computadora sencilla actual: Páginas de 4096 bytes (2^{12}), espacio de direccionamiento de 32 bits...

- 1,048,576 páginas (2^{20})
- Cada entrada de 40 bits (20 para la página, 20 para el marco)
- $\frac{40}{8} \times 1048576 \rightarrow$ ¡¿5MB para la tabla de páginas?!



¿Cuánto ocuparía en una computadora real actual?

Computadora sencilla actual: Páginas de 4096 bytes (2^{12}), espacio de direccionamiento de 32 bits...

- 1,048,576 páginas (2^{20})
- Cada entrada de 40 bits (20 para la página, 20 para el marco)
- $\frac{40}{8} \times 1048576 \rightarrow$ ¡¿5MB para la tabla de páginas?!
- $\frac{5242880}{4096} = 1280 \rightarrow$ ¡¿La tabla de páginas mide 1280 páginas?!



Impacto en las transferencias DMA

- Es mucho más eficiente hacer una transferencia larga que varias cortas
 - Recordemos: Una transferencia DMA se *inicia* indicando:
 - Dirección física base de memoria
 - Cantidad de datos a transferir
 - Puerto del dispositivo
 - Dirección de la transferencia
- Tenemos que hacer (como máximo) transferencias del tamaño de la página en memoria
 - Es poco probable que las páginas lógicamente consecutivas estén físicamente contiguas
- Fragmentar la memoria física en páginas muy pequeñas reduce la velocidad de transferencia de disco

Gunnar Wolf • distop/gwolf.org • http://distop.gwolf.org
Sistemas Operativos (11554) • Facultad de Ingeniería UNAH



Gunnar Wolf Administración de memoria: Asignación de memoria

Memoria contigua
Segmentación
Paginación

Impacto en el PCB

Volvamos a hacer cuentas:

¿Cuánto pesa cada esquema de asignación de memoria?

- ① **Partición fija** Un registro *base* para indicar el *inicio* del espacio del proceso
- ② **Partición flexible** Un registro *base* y un registro *límite* para indicar su tamaño total
- ③ **Segmentación** Varios registros *de segmento* (p.ej. 6 en x86)
- ④ **Paginación** Modelo presentado (128 páginas de 512 bytes, espacio de direccionamiento de 16 bits): 1792 bytes

Gunnar Wolf • distop/gwolf.org • http://distop.gwolf.org
Sistemas Operativos (11554) • Facultad de Ingeniería UNAH



Almacenamiento de la tabla de páginas en memoria

- Almacenar toda esta información en registros es sencillamente imposible
 - Costo económico
 - Tiempo perdido en el cambio de contexto
- Una estrategia: Almacenar *la propia tabla de páginas* en memoria privilegiada (accesible únicamente para el SO, no para el proceso usuario)
 - Emplear sólo dos registros especiales:
 - PTBR Registro Base de Tabla de Páginas (*Page Table Base Register*)
 - PTLR Registro Límite de Tabla de Páginas (*Page Table Limit Register*)
 - ¿Por qué el PTLR? La mayor parte de los procesos, además, nunca requerirá direccionarlo completo; para no tener un mapa completo mayormente vacío, PTLR indica la extensión máxima empleada



Desventaja de tener la tabla de páginas en memoria:

- Velocidad: *Cada acceso a memoria se ve penalizado con (al menos) un acceso adicional*
 - Para resolver dónde está la página que buscamos
- *Duplica el tiempo efectivo de acceso a memoria → Inaceptable.*



TLB: El buffer de traducción adelantada

- ¿Respuesta? Emplear un caché
 - Pero, por su frecuencia y naturaleza de uso, un caché especializado
- El TLB (*Translation Lookaside Buffer*) es una tabla asociativa (*hash*)
 - Las consultas (*llaves*) son las páginas
 - Los valores son los marcos correspondientes
 - Las búsquedas se realizan en *tiempo constante*



Gunnar Wolf Administración de memoria: Asignación de memoria

Memoria contigua Segmentación Paginación

Proceso de paginación con TLB

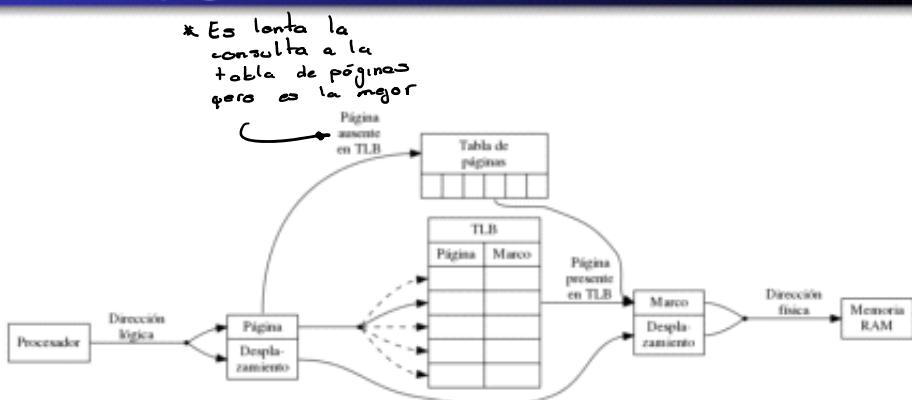


Figura: Esquema de paginación empleando un *buffer de traducción adelantada* (TLB)

Funcionamiento del TLB

- Típicamente, entre 64 y 1024 entradas
 - Busca aprovechar la *localidad de referencia*
- Si *conoce* ya a una página (*TLB hit*), el MMU traduce de inmediato para obtener el marco
 - Silberschatz: Penalización de hasta 10 %
- Si *no* conoce la página (*TLB miss*), lanza una falla de página (*page fault*) y consulta en la memoria principal cuál es el marco correspondiente
 - Penalización >120 %; la nueva página queda registrada en el TLB



marco correspondiente

- Penalización >120 %; la nueva página queda registrada en el TLB



Reemplazo de las entradas del TLB

- El TLB es *mucho más pequeño* que el espacio de páginas del proceso
 - Al recibir una entrada nueva, puede ser necesario reemplazar la entrada de una página ya conocida
 - Reemplazar la *menos recientemente utilizada* (*LRU*)
 - Ventaja: Localidad de referencia; probablemente no la necesitemos pronto
 - Desventaja: Contabilizar los accesos dentro del TLB (*muy frecuentes*) agrega latencia y costo
 - Reemplazar una página al azar
 - Ventaja: Más simple y barato (pero... ¿Algo aleatorio es barato en el cómputo?)
 - Desventaja: Obvia. Puede reemplazarse una página en uso frecuente y causar demoras

Memoria Virtual

Tuesday, 21 April 2020 5:31 PM

Disociar por completo memoria física y lógica

- El primer gran paso hacia la memoria virtual lo cubrimos al hablar de paginación
 - Cada proceso tiene una *vista lógica* de su memoria
 - Cada proceso se *mapea* a la memoria física
 - Pero es exclusivo, distinto del de los demás procesos
- Ahora cada proceso tiene un espacio de direccionamiento exclusivo y muy grande
 - Pero omitimos cómo es que podemos ofrecer más memoria que la físicamente disponible
- Aquí entra en juego la *memoria virtual*
 - La memoria física es sólo una *proyección parcial* de la memoria lógica, potencialmente mucho mayor



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto

Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Retomando el intercambio

- Vimos el intercambio en primer término al *intercambio (swap)* al hablar de memoria particionada
 - Espacio de memoria completo de un proceso
- Mejora cuando hablamos de segmentación
 - Intercambio parcial; segmentos no utilizados.
 - El proceso puede continuar con porciones *congeladas* a almacenamiento secundario
- Con la memoria virtual, el intercambio se realiza *por página*
 - Mucho más rápido que por bloques tan grandes como un segmento
 - Completamente transparente al proceso



Gunnar Wolf • sistop@gwoll.org • http://sistop.gwoll.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Esquema general empleando memoria virtual

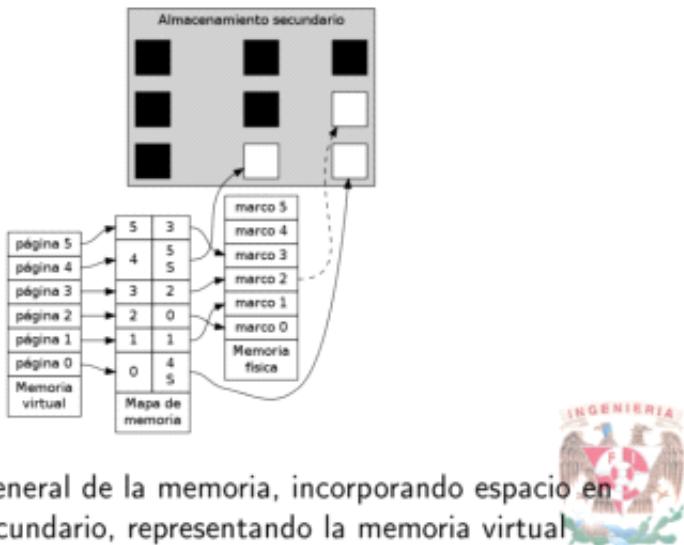


Figura: Esquema general de la memoria, incorporando espacio en almacenamiento secundario, representando la memoria virtual

Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Pequeño cambio de nomenclatura

- El *intercambio* (*swap*) deja de ser un *último recurso*
 - Pasa a ser un elemento más en la jerarquía de memoria
- El mecanismo para intercambiar páginas al disco ya no es un mecanismo aparte
 - Ya no hablamos del *intercambiador* (*swapper*)
 - Sino que del *paginador*

► Reemplazo de Páginas

Manteniendo el sobre-compromiso

- Cuando *sobre-comprometemos* memoria, los procesos en ejecución pueden terminar requiriendo que se carguen más páginas de las que caben en la memoria física
- Mantenemos el objetivo del sistema operativo: *Otorgar a los usuarios la ilusión de una computadora dedicada a sus procesos*
- No sería aceptable terminar la ejecución de un proceso ya aceptado
 - Mucho menos si ya fueron aprobados sus requisitos y nos quedamos sin recurso
 - → Tenemos que llevar a cabo un *reemplazo de páginas*



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Importancia del reemplazo de páginas

- Parte fundamental de la paginación sobre demanda
- La pieza que posibilita una verdadera separación entre memoria lógica y física
- Mecanismo que permite liberar alguno de los marcos actualmente ocupado



Mecanismo para liberar un marco ocupado

- Cuando todos los marcos están ocupados (o se cruza el umbral determinado), un algoritmo designa a una *página víctima* para su liberación
 - Veremos más adelante algunos algoritmos para esto
- El paginador graba a disco los contenidos de esta página y la marca como libre
 - Actualizando el PCB y TLB del proceso al cual pertenece
- Puede continuar la carga de la página requerida
 - ¡Ojalá esto significara que no dañaría el sistema de...



- Puede continuar la carga de la página requerida
- ¡Ojo! Esto significa que se *duplica* el tiempo de transferencia en caso de fallo de página (t_f)



The screenshot shows a presentation slide with the following elements:

- Header: Gunnar Wolf | Administración de memoria: Memoria virtual
- Navigation icons: back, forward, search, etc.
- Section titles (partially visible): Concepto, Paginación sobre demanda, Reemplazo de páginas, Asignación de marcos.
- Main content area: A large black bar containing the text "Manteniendo a t_f en su lugar".

- Con apoyo del MMU podemos reducir la probabilidad de esta duplicación en t_f
- Agregamos un *bit de modificación* o *bit de página sucia* a la tabla de páginas
 - Apagado cuando la página se carga a memoria
 - Se enciende cuando se realiza un acceso de escritura a esta página
- Al elegir una página víctima, si su *bit de página sucia* está encendido, es necesario grabarla a disco
 - Pero si está apagado, basta actualizar las tablas del proceso afectado
 - Ahorra la mitad del tiempo de transferencia



¿Cómo elegir una página víctima?

- Para elegir una víctima para paginarla al disco empleamos un *algoritmo de reemplazo de páginas*
- Buscamos una característica: Para un patrón de accesos dado, obtener el *menor número* de fallos de página
 - Diferentes patrones de acceso generan diferentes resultados para cada algoritmo
 - Nos referiremos a estos patrones de acceso como *cadena de referencia*

Para los ejemplos presentados a continuación, nos basaremos en los presentados en *Operating Systems Concepts Essentials*

(Silberschatz, Galvin y Gagné, 2011)



The screenshot shows a presentation slide with the following elements:

- Top navigation bar: Gunnar Wolf and Administración de memoria: Memoria virtual.
- Left sidebar menu: Concepto, Paginación sobre demanda, Reemplazo de páginas, Asignación de marcos.
- Main title: Eligiendo una cadena de referencia.

- La cadena de referencia debe representar un patrón típico (para la carga que deseemos analizar) de accesos a memoria
- Muchas veces son tomados de un volcado/trazado de ejecución en un sistema real
 - El conjunto resultante puede ser enorme
 - Simplificación: No nos interesa el acceso independiente a cada dirección de memoria, sino que a cada página
 - Varios accesos consecutivos a la misma página no tienen efecto en el análisis



► Algoritmos

Por ejemplo, a partir de la cadena de referencia:

1, 4, 3, 4, 1, 2, 4, 2, 1, 3, 1, 4

- En una computadora con ≥ 4 marcos, sólo se producirían cuatro fallos
 - Los necesarios para la *carga inicial*
- Extremo opuesto: Con un sólo marco, tendríamos 12 fallos
 - Cada página tendría que cargarse siempre desde disco
- Casos que se pueden estudiar: 2 o 3 marcos



Primero en entrar, primero en salir (FIFO) (1)

- Nuevamente, el algoritmo más simple y de obvia implementación
- Al cargar una página, se toma nota de cuándo fue cargada
- Cuando llegue el momento de reemplazar una página vieja, se elige la que se haya cargado hace más tiempo

Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Primero en entrar, primero en salir (FIFO) (2)

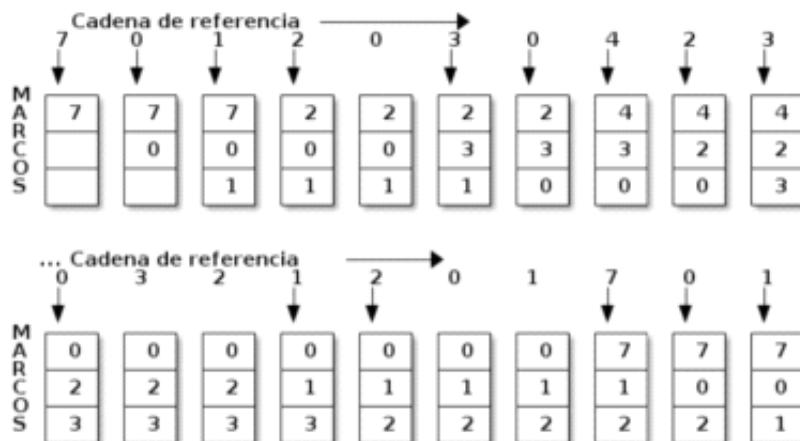


Figura: Algoritmo FIFO de reemplazo de páginas: 15 fallos

Primero en entrar, primero en salir (FIFO) (3)

- Típicamente programado empleando una lista ligada circular
 - Cada elemento que va recibiendo se agrega como el último elemento
 - Tras agregarlo, se “empuja” al apuntador para convertirlo en la cabeza
- Desventaja: No toma en cuenta la historia de las últimas solicitudes
 - La cantidad de patrones de uso que le pueden causar un bajo desempeño es alto
 - Todas las páginas tienen la misma probabilidad de ser reemplazadas, independientemente de su frecuencia de uso



Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Anomalía de Belady

- En general, asumimos que a mayor cantidad de marcos de memoria disponibles, menos fallos de página se van a presentar
- La *Anomalía de Belady* ocurre cuando un incremento en el número de marcos disponibles lleva a más fallos de página
 - Depende del algoritmo y de la secuencia de la cadena de referencia
- FIFO es vulnerable a la anomalía de Belady



Anomalía de Belady: Expectativas de comportamiento

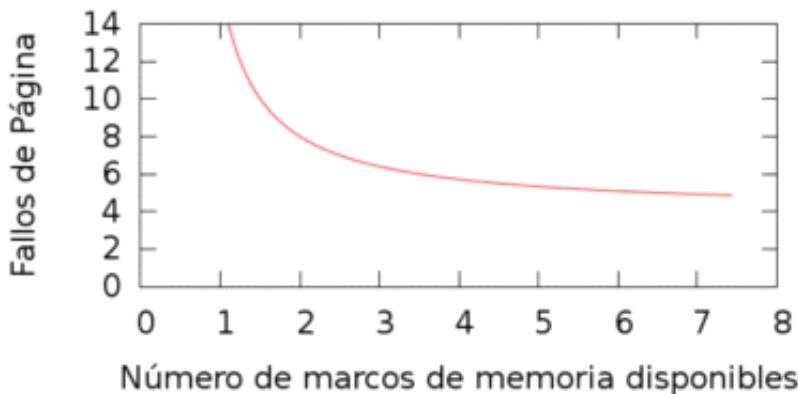


Figura: Relación ideal entre el número de marcos y la cantidad de fallos de página



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Anomalía de Belady: Comportamiento de FIFO

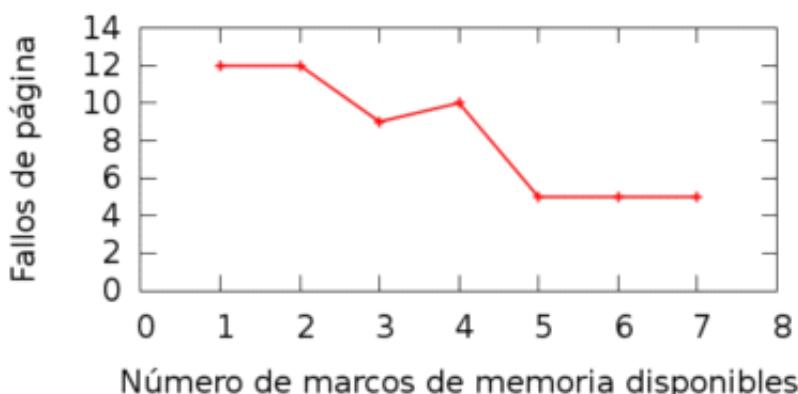
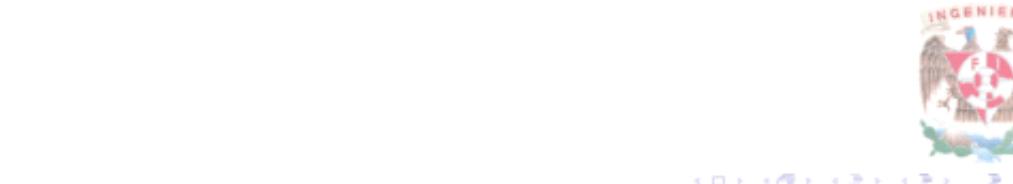


Figura: El algoritmo FIFO presenta la anomalía de Belady con la cadena de referencia 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Algoritmo óptimo (OPT) o mínimo (MIN) (1)

- Interes casi puramente teórico
- Elegimos como página víctima a aquella que *no vaya a ser utilizada por un tiempo máximo*



Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Algoritmo óptimo (OPT) o mínimo (MIN) (2)

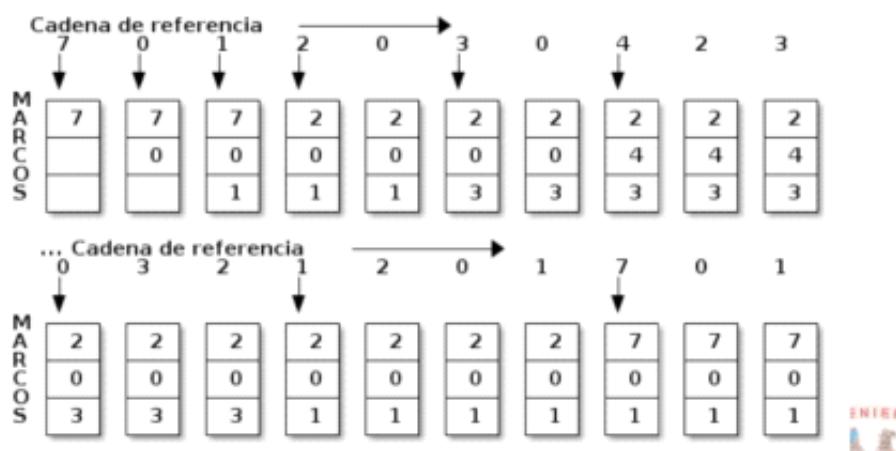


Figura: Algoritmo óptimo de reemplazo de páginas (OPT): 9 fallos

Algoritmo óptimo (OPT) o mínimo (MIN) (3)

- Óptimo demostrado, pero no aplicable
- Requiere conocimiento a priori de las necesidades del sistema
 - Si es de por sí impracticable en los despachadores, lo es mucho mas al hablar de un área tan dinámica como la memoria
 - Recuerden: Millones de accesos por segundo
- Principal utilidad: Brinda una cota mínima
 - Podemos ver qué tan cercano resulta otro algoritmo respecto al caso óptimo



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Menos recientemente utilizado (LRU) (1)

- Lo hemos mencionado ya en varios puntos de la administración de memoria
- Busca acercarse a OPT *prediciendo* cuándo será el próximo uso de cada una de las páginas
 - Basado en su historia reciente
- Elige la página que *no ha sido empleada* desde hace más tiempo



Menos recientemente utilizado (LRU) (2)



Figura: Algoritmo reemplazo de páginas menos recientemente utilizadas (LRU): 12 fallos

Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Menos recientemente utilizado (LRU) (3)

- Para nuestra cadena de referencia, resulta en el punto medio entre OPT y FIFO
- Para una cadena S y su cadena espejo R^S , $OPT(S) = LRU(R^S)$ (y viceversa)
- Está demostrado que LRU y OPT están libres de la anomalía de Belady
 - Para n marcos, las páginas que están en memoria son un subconjunto estricto de las que estarían con $n + 1$ marcos.



Implementación ejemplo de LRU (1)

- Se agrega un contador a *cada uno* de los marcos
 - El contador se incrementa siempre que se hace referencia a una página
- Se elige como víctima a la página con el contador más bajo
 - Esto es, a la que hace más tiempo no haya sido actualizada
- Desventaja: Con muchas páginas, se tiene que recorrer *la lista completa* para encontrar la más *envejecida*



Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Implementación ejemplo de LRU (2)

- La lista de marcos es una lista doblemente ligada
- Esta lista es tratada como una lista y como un stack
 - Cuando se hace referencia a una página, se mueve a la cabeza (arriba) del stack — Peor caso: 6 operaciones
 - Para elegir a una página víctima, se toma la de *abajo* del stack (tiempo constante)



► Aproximaciones a LRU

Aproximaciones a LRU

- ¿Principal debilidad de LRU? Su implementación requiere apoyo en hardware mucho más complejo que FIFO
- Hay varios mecanismos que buscan *aproximar* el comportamiento de LRU
 - Empleando información menos detallada



Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Bit de referencia

- Aproximación bastante común
- Todas las entradas de la tabla de páginas tienen un *bit de referencia*, inicialmente apagado
 - Cada vez que se referencia a un marco, se enciende su bit de referencia
 - El sistema *reinicia* periódicamente a *todos* los bits de referencia, apagándolos
- Al presentarse un fallo de página, se elige por FIFO de entre el subconjunto con el bit apagado
 - Esto es, entre las páginas que no fueron empleadas en el periodo



Segunda oportunidad (o reloj)

- Maneja un bit de referencia y un recorrido tipo FIFO
- El algoritmo avanza linealmente sobre la lista ligada circular
- Hay eventos que encienden el bit, y eventos que lo apagan:
 - Una referencia a un marco enciende su bit de referencia
 - Si elige a un marco que tiene encendido el bit de referencia, lo apaga y avanza una posición (dándole una

Segunda oportunidad (o reloj)

- Maneja un bit de referencia y un recorrido tipo FIFO
- El algoritmo avanza linealmente sobre la lista ligada circular
- Hay eventos que encienden el bit, y eventos que lo apagan:
 - Una referencia a un marco enciende su bit de referencia
 - Si elige a un marco que tiene encendido el bit de referencia, lo apaga y avanza una posición (dándole una *segunda oportunidad*)
 - Si elige a un marco que tiene apagado el bit de referencia, lo designa como página víctima
- Se le llama *de reloj* porque puede verse como una manecilla que avanza sobre la lista de marcos
 - Hasta encontrar uno con el bit de referencia apagado



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Segunda oportunidad mejorada (1)

- Si agregamos al bit de referencia un bit de *modificación*, nos mayor expresividad, y puede ayudar a elegir a una página víctima *más barata*. En orden de preferencia:
 - (0,0) El marco no ha sido utilizado ni modificado. Buen candidato.
 - (0,1) Sin uso reciente, pero está *sucio*. Hay que escribirlo a disco.
 - (1,0) Está *limpio*, pero tiene uso reciente, y es probable que se vuelva a usar pronto
 - (1,1) Empleado recientemente y *sucio*. Habría que grabarlo a disco, y tal vez vuelva a requerirse pronto. Hay que evitar reemplazarlo.



Algoritmos con manejo de buffers

- De uso cada vez más frecuente
- No esperan a que el sistema requiera reemplazar un marco, buscan *siempre tener espacio disponible*
 - Algoritmos ansiosos, no flojos
 - Operan basados en *umbrales* aceptables/deseables
- Conforme la carga lo permite, el SO busca las páginas sucias más proclives a ser paginadas
 - Va copiándolas a disco y marcándolas como limpias



sucias mas proclives a ser paginadas

- Va copiándolas a disco y marcándolas como limpias
- Cuando tenga que traer una página de disco, siempre habrá dónde ubicarla sin tener que hacer una transferencia



Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Ejemplo: Tres sistemas Linux (1)

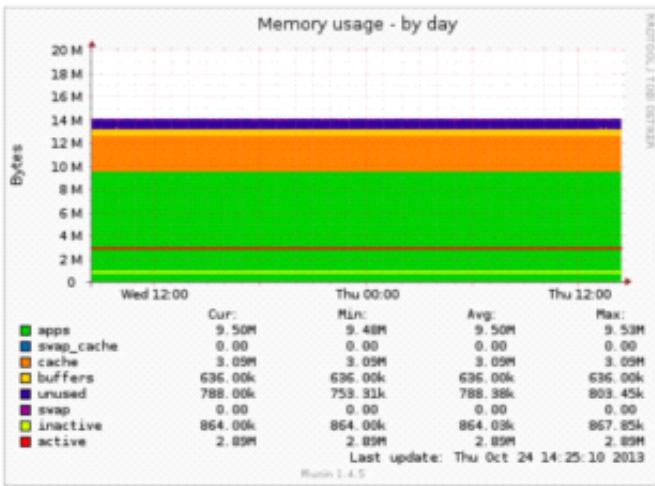


Figura: Manejo de memoria (24 horas) en un sistema embebido (16MB RAM)



Asignación de Marcos

Thursday, 23 April 2020 5:58 PM

Viendo el lado opuesto del problema

- Vimos ya cómo *retirar* marcos asignados
- ¿Cómo conviene *asignar* los marcos a los procesos?
- Definamos algunos parámetros para nuestros ejemplos
 - Un sistema con 1024KB de memoria física
 - 256 páginas de 4KB cada una
 - El sistema operativo ocupa 248KB (62 páginas); 194 páginas para los procesos a ejecutar



< > < > < > < > < > < >

Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

Vuelta a la paginación puramente sobre demanda

- En un esquema de paginación puramente sobre demanda, cada fallo de página que se va generando lleva a que se asigne el marco correspondiente
- Se van asignando los marcos conforme son requeridos, hasta que hay 194 páginas ocupadas por procesos
 - Entonces, entran en escena los algoritmos de *reemplazo de páginas* que ya vimos
 - Claro está, cuando un proceso termina, sus marcos vuelven a la lista de marcos libres



Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación igualitaria

- Buscando un reparto *justo* de recursos, se divide el total de memoria física disponible entre el número de procesos
- Volviendo a nuestra computadora ejemplo (256 marcos; 62 marcos asignados al sistema, 194 a los procesos):
 - Si tenemos 4 procesos en ejecución, dos tendrán derecho a 49 marcos y dos a 48
 - Los marcos no pueden dividirse: es imposible asignar 48.5

Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación igualitaria

- Buscando un reparto *justo* de recursos, se divide el total de memoria física disponible entre el número de procesos
- Volviendo a nuestra computadora ejemplo (256 marcos; 62 marcos asignados al sistema, 194 a los procesos):
 - Si tenemos 4 procesos en ejecución, dos tendrán derecho a 49 marcos y dos a 48
 - Los marcos no pueden dividirse; es imposible asignar 48.5 a cada uno
- El esquema es justo, pero deficiente
 - Si tenemos un gestor de bases de datos P_1 con 2048KB (512 marcos) de memoria virtual, y un proceso de usuario P_2 que sólo requiere 112KB (28 páginas)...
 - Ambos recibirán lo mismo — Y P_2 desperdiciará 20 páginas



Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

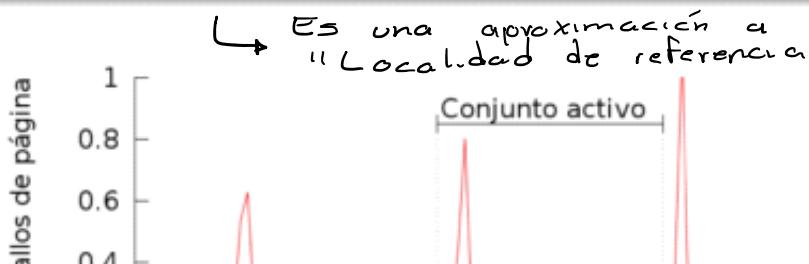
Asignación proporcional

- Brinda a cada proceso una porción del espacio de memoria física proporcional a su uso de memoria virtual
- Si además de los dos procesos descritos tenemos a P_3 con 560KB (140 páginas) y P_4 con 320KB (80 páginas) de memoria virtual
 - Uso total de memoria virtual:
 $V_T = 512 + 28 + 140 + 80 = 760$ páginas
 - Sobreuso de memoria física cercano al 4:1 respecto a las 194 páginas disponibles
- Cada proceso recibirá $F_P = \frac{V_P}{V_T} m$
 - F_P : Espacio de memoria física que recibirá
 - V_P : Cantidad de memoria virtual que emplea,
 - m : Total de marcos de memoria física disponibles
- P_1 : 130 marcos; P_2 : 7 marcos; P_3 : 35 marcos; P_4 : 20



Sistemas de
Tiempo Real "Duro" \Rightarrow No maneja o usa
memoria virtual

El conjunto activo



El conjunto activo

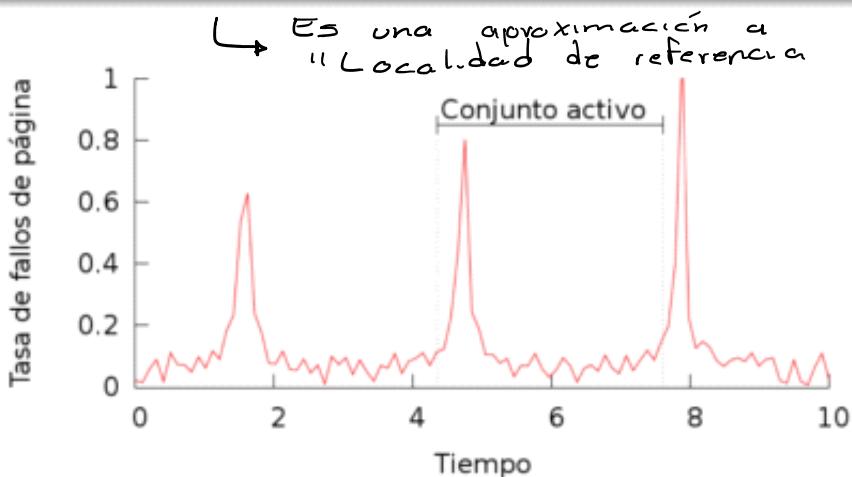


Figura: Los picos y valles en la cantidad de fallos de página de un proceso definen a su *conjunto activo* (Silberschatz, p.349)



Gunnar Wolf Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

El conjunto activo y el espacio en memoria

→ No es fácil detectarlo

- Idealmente, en todo momento, debemos asignar a cada proceso *suficientes páginas* para mantener en memoria física su conjunto activo
- Si no es posible hacerlo, el proceso es buen candidato para ser suspendido → No cabe en memoria
- ... Pero no es fácil detectar con claridad *cuál* es el conjunto activo
 - Mucho menos predecir cuál será dentro de determinado tiempo
 - ¿Cuánto dura un proceso dentro de determinada rutina?
 - Puede requerir rastrear y verificar decenas de miles de accesos a memoria



Marcas de memoria → Asignación → Proceso

► Hiperpaginación

¿Y el tiempo real?

- Cuando presentamos al *tiempo real* (*Planificación de procesos*), mencionamos que el tiempo real duro es incompatible con sistemas basados en memoria virtual
 - Principal razón: Las demoras inducidas por la paginación
 - Podría indicarse que un proceso de tiempo real esté 100 % en memoria física (nunca candidato para paginación)
 - *Reduce fuertemente* el impacto que sufriría al pelear por recursos
 - Pero no lo resuelve por completo
 - Ni la contención en el bus, ni la inversión de prioridades...
 - → Sólo podemos prometer *tiempo real suave*



Gunnar Wolf

Administración de memoria: Memoria virtual

- Uno o más procesos tienen demasiadas pocas páginas asignadas para llevar a cabo su trabajo
 - Generan fallos de pagina con tal frecuencia que resulta imposible realizar trabajo real
 - O resulta tan lento que la percepción es de no-avance
 - El sistema pasa más tiempo intentando satisfacer la paginación que trabajando

Estamos en estado de *hiperpaginación*
En inglés, *thrashing* (literal: *paliza*)



¿Qué puede llevar a la hiperpaginación? (1)

- El sistema tiene una carga normal
 - Esquema de reemplazo global de marcos
 - Se lanza un nuevo proceso
 - Su inicialización requiere poblar estructuras a lo largo de su memoria virtual
 - O cambia de conjunto activo
 - Serie de fallos de página → El sistema responde, reemplazando a varios marcos de otros procesos

¿Qué puede llevar a la hiperpaginación? (1)

- El sistema tiene una carga normal
- Esquema de reemplazo global de marcos
- Se lanza un nuevo proceso
 - Su inicialización requiere poblar estructuras a lo largo de su memoria virtual
 - O cambia de conjunto activo
 - Serie de fallos de página → El sistema responde, reemplazando a varios marcos de otros procesos
- Mientras esto continúa operando, algunos de los procesos víctima requieren de las páginas que pasaron a disco
 - Recordemos que el disco es miles a millones de veces más lento que la memoria...

OFFICE



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos

¿Qué puede llevar a la hiperpaginación? (2)

- La utilización del procesador decrece
 - ... Porque los procesos están esperando a que su memoria esté disponible
- El sistema operativo aprovecha la situación para lanzar procesos de mantenimiento
 - Que requieren que se les asigne memoria
 - → Reducen aún más el espacio de memoria física disponible
- Se forma una cola de solicitudes de paginación (algunas veces contradictorias)
- Baja todavía más la actividad del procesador (NOOP)



¿Cómo se ve la hiperpaginación?



Figura: Al aumentar demasiado el grado de multiprogramación, el uso del CPU cae abruptamente y caemos en la hiperpaginación (Silberschatz, p.349)



Gunnar Wolf

Administración de memoria: Memoria virtual

Concepto
Paginación sobre demanda
Reemplazo de páginas
Asignación de marcos



Respondiendo a la hiperpaginación

- Los síntomas son muy claros
 - Fáciles de detectar — ¡pregúntenle a cualquier usuario!
- Reducir temporalmente el nivel de multiprogramación
 - Caímos en hiperpaginación por tener requisitos en memoria que no alcanzamos a satisfacer con la memoria física disponible
 - • El sistema puede seleccionar a uno (o más) procesos y suspenderlos por completo
 - • Incluso poner su memoria física a disposición de otros procesos
 - Hasta que salgamos del estado de hiperpaginación

