Administración de memoria

Sistemas Operativos
DC - UBA - FCEN

24 de abril de 2025

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Motivación

- La CPU sólo puede ejecutar instrucciones que lee de memoria principal.
- Los programas y datos viven en almacenamiento secundario, así que hay que cargarlos en memoria principal para ejecutarlos.
- Con multiprogramación, ¡se cargan muchos programas!
  - Ningún proceso debería saber cómo está siendo compartida la memoria.
  - Los procesos no deben poder "molestarse" entre sí.
  - No queremos que esto degrade la eficiencia de uso de CPU y memoria.
- Necesitamos **memory management.**

Administración de memoria

El sistema operativo es responsable de:

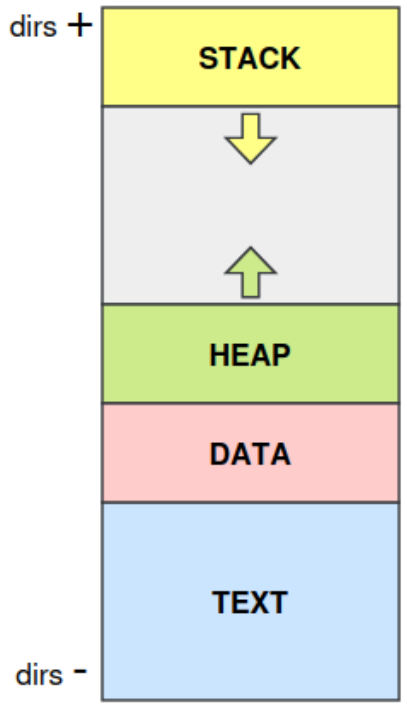
- Saber qué partes de la memoria están en uso.
- Saber qué proceso usa cada parte de la memoria.
- Asignar y liberar espacios de memoria.

Repaso

¿Qué pinta tiene la memoria de un proceso?

La memoria de un proceso es una colección de segmentos o secciones.

- Código (text): código de programa.
- Data: contiene la porción de datos inicializada de un programa.
- A medida que se necesita, el stack crece para las direcciones más bajas, el heap crece para las direcciones más altas.



Si tenemos multiples procesos, ¿cómo podemos particionar la memoria?

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Asignación de memoria contigua

Esquema de memoria contigua

Se asignan los procesos a partes de memoria de tamaño variable, donde cada una contiene exactamente un proceso.

- El sistema operativo mantiene una estructura (puede ser un bitmap o una lista enlazada) indicando qué partes de la memoria están disponibles y cuáles están ocupadas.
- Cuando un proceso entra, el SO se fija sus requerimientos de memoria y la cantidad de memoria disponible.
- Cuando un proceso termina, libera su memoria.

Asignación de memoria contigua

Estrategias

- Los espacios de memoria disponible se ven como un conjunto de "huecos" de varios tamaños dispersos por la memoria. Cuando un proceso llega y necesita memoria, el sistema busca un "hueco" con espacio suficiente para este proceso.
- Estrategias para asignar memoria:
  - **First-fit.** Asigna el primer "hueco" lo suficientemente grande.
  - **Best-fit.** Asigna el "hueco" más chico que sea lo suficientemente grande.
  - **Worst-fit.** Asigna el "hueco" más grande.

Asignación de memoria contigua

Ejercicio

Enunciado

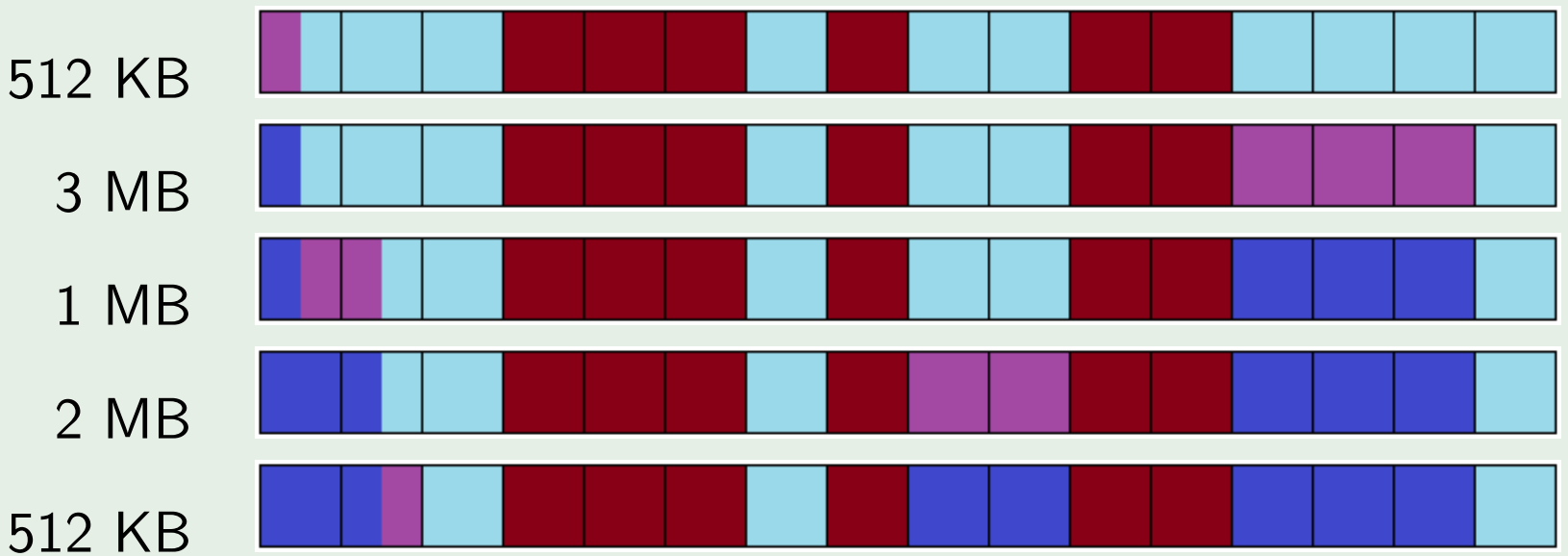
Tengo un sistema con 16 MB de memoria **sin particionar** que direcciona a byte. El estado actual de la memoria es el siguiente (cuadrado = 1MB):

Llegan los siguientes pedidos de memoria en ese orden:
512 KB, 3 MB, 1 MB, 2MB, 512 KB.
Indicar qué bloques se asignan para cada pedido utilizando *first-fit*, *best-fit* y *worst-fit*.

Asignación de memoria contigua

Ejercicio

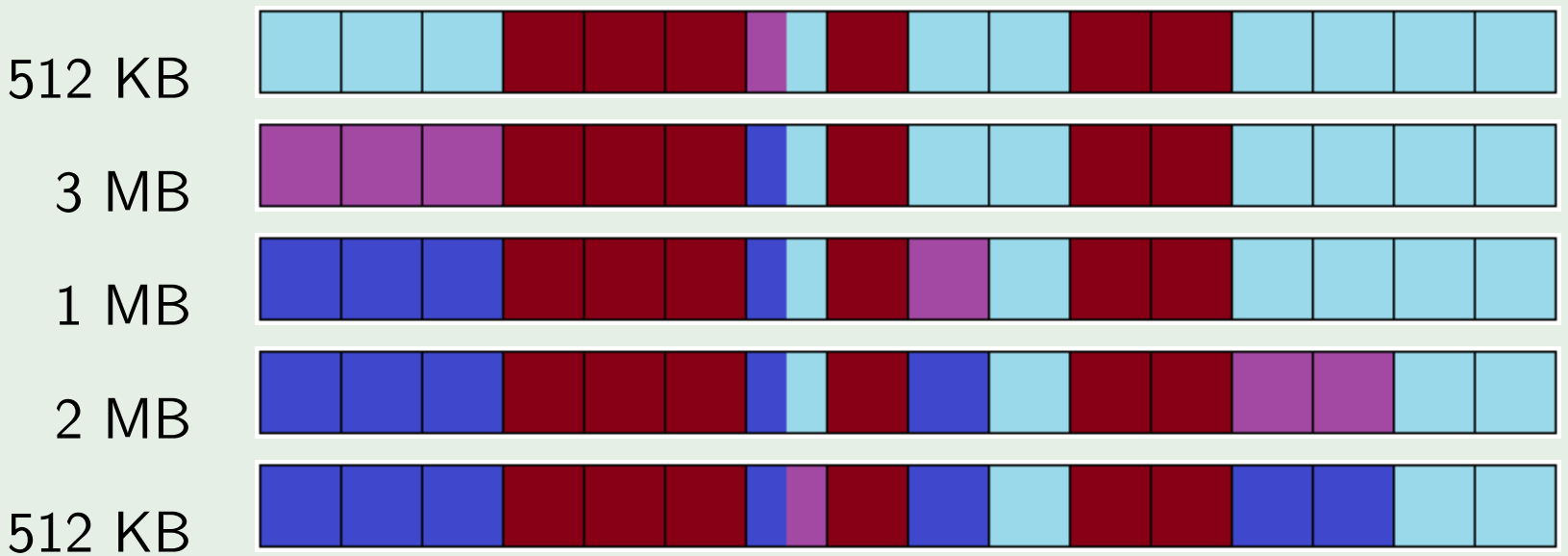
Solución First-Fit



Asignación de memoria contigua

Ejercicio

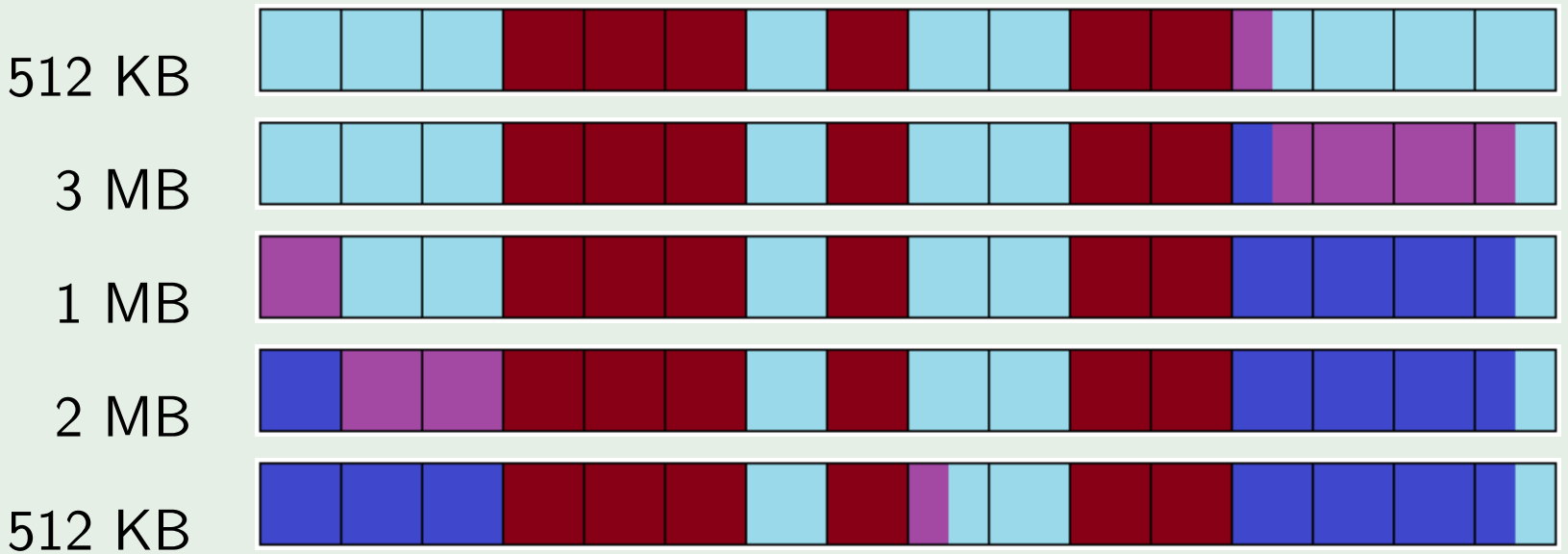
Solución Best-Fit



Asignación de memoria contigua

Ejercicio

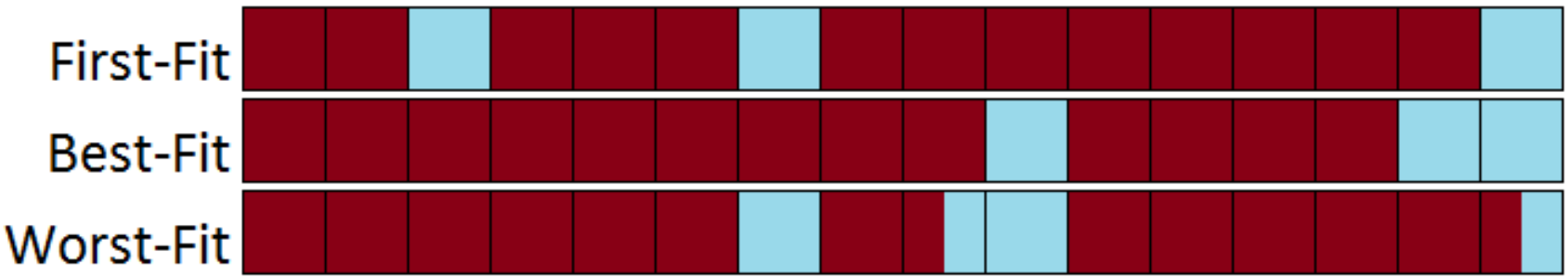
Solución Worst-Fit



Asignación de memoria contigua

Ejercicio

Entonces, ¿Cuál es mejor?



Asignación de memoria contigua

Problemas

- Se debe conocer el tamaño de cada proceso antes de cargarlo.
- Una vez cargado, el proceso no puede cambiar de lugar en la memoria.
- Mientras los procesos se cargan y quitan de memoria, el espacio de memoria libre se fragmenta en pedacitos.

Fragmentación Externa

Cuando hay suficiente memoria para satisfacer un pedido pero los espacios disponibles no son contiguos.

- ¿Cómo evitar esto?
  1. **Partir la memoria física en bloques de tamaño fijo** y asignar memoria en unidades basadas en el tamaño de bloque.
  2. **Permitir que el espacio de direcciones físicas de los procesos no sea contiguo**, permitiendo que a un proceso se le asigne memoria siempre que haya disponible.

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Memoria Virtual

- **Direcciones virtuales:** Las direcciones de memoria son virtuales (de mentirita), el SO las mapea a direcciones de memoria físicas (reales).
- **Registro de traducciones:** El SO mantiene un registro de las traducciones de direcciones virtuales a físicas y realiza la traducción en cada acceso a memoria.
- **Protección:** Debido a que el SO intercepta cada referencia a memoria, puede restringir el acceso a ciertas direcciones, protegiendo la memoria de otros procesos o del propio sistema operativo.

Memoria Virtual

MMU

- Problema: interceptar y traducir cada referencia a memoria es caro.
- Solución: soporte de hardware.

MMU

La *memory management unit* (MMU) es un componente de hardware que traduce las direcciones dinámicamente durante cada referencia a memoria.

- Ahora tenemos dos puntos de vista de la memoria:
  - La dirección virtual es la que ve el programa.
  - La dirección física es la verdadera ubicación en memoria.

¿Cómo hacen SO+MMU para traducir de direcciones virtuales a físicas?

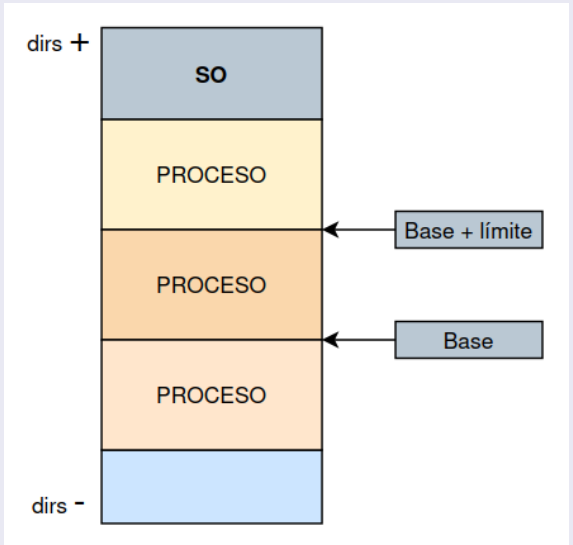
Estrategias de pasaje de direcciones

Repasemos tres formas de encarar este problema.

- Base y límite
- Segmentación
- Paginación

Base y límite

- Base: dirección física que indica el inicio, corresponde a la dirección virtual 0. Es controlado por el SO, el proceso sólo ve direcciones virtuales.
- Límite: una dirección de memoria virtual mayor que la más alta permitida.
- Para cada referencia a memoria, si la dir virtual es mayor que el límite: es una referencia inválida (*trap*), si no: se suma la base a la dir virtual provista para producir la dir física.



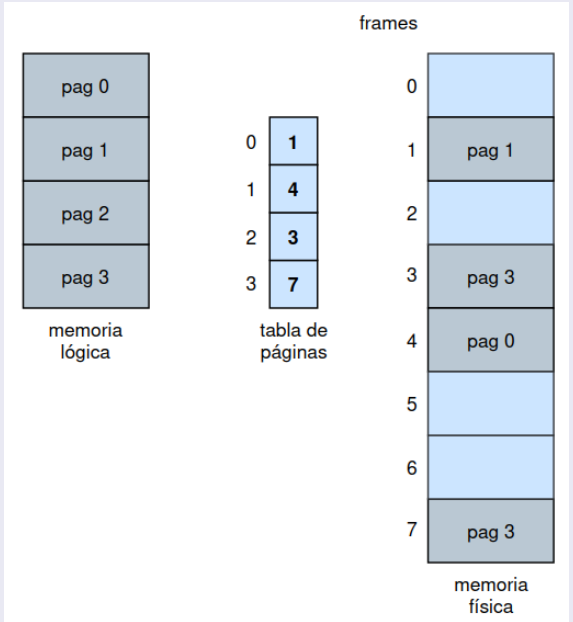
- El SO puede actualizar base y límite de un proceso si hace falta.
- Pros: barato, requiere poco espacio, separación entre dirs virtuales y físicas.
- Cons: el espacio físico debe ser contiguo, hay fragmentación, no soporta regiones de sólo lectura.

Segmentación

- Cada proceso se divide en múltiples áreas de memoria de tamaño variable llamadas segmentos. Por ejemplo: un segmento para código, un segmento para datos/heap, un segmento para pila.
- El inicio de cada segmento está fijo en una dirección virtual específica.
- El SO mapea cada segmento individualmente: cada segmento tiene sus propios base+límite, y se mantiene un mapeo de segmentos para cada proceso.
- También se almacena un bit de protección para cada segmento: indica permiso de escritura por parte del proceso.
- Continúa el problema de la fragmentación.

Paginación

- La memoria física se particiona en bloques de tamaño fijo llamados marcos o frames, y la memoria virtual se particiona en bloques del mismo tamaño llamados páginas (tamaño más común: 4KB).
- Las páginas de un proceso se cargan en marcos de memoria.
- Permite que el espacio de direcciones de memoria física de un proceso sea no contiguo.



- Cada proceso tiene su tabla de páginas con una entrada para cada página virtual, mapeándola al número de página física, más otra información como bit de protección.

Paginación

- ¡No más fragmentación externa! cualquier marco libre puede ser asignado a un proceso que lo necesite.
- Pero como los frames se asignan como unidades, el último frame asignado puede no estar completamente lleno.

Fragmentación interna

Cuando queda memoria sin utilizar dentro de una partición asignada.

Memoria virtual

¿Y si nos quedamos sin memoria?

- Si un proceso ocupa n páginas, en un primer enfoque al menos n frames deben estar disponibles en memoria para poder asignarle.
- ¿Qué hacemos si no tenemos suficiente memoria?
  - ¿Prohibir que otro programa pida memoria hasta que se libere un poco? No es lo ideal...
  - Otra idea: "fletamos" alguna página.La bajamos a disco y usamos el *frame* liberado. Si más tarde vuelvo a necesitar esa página, la vuelvo a cargar.
    Así es como la memoria física parece más grande de lo que es.

La memoria virtual permite la **ejecución de procesos que están sólo parcialmente en memoria**. Así, los programas pueden ser más grandes que la cantidad de memoria física disponible. Además, como cada programa toma menos memoria física, más programas pueden correr al mismo tiempo.

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Paginación a Demanda

- En lugar de cargar el programa entero en memoria física para poder ejecutarlo, **cargar sólo las páginas que son necesarias en cada momento**.
- Como algunas páginas estarán en memoria y algunas en almacenamiento secundario, necesitamos soporte de hardware (ejemplo: bit válido-inválido).
- Intentar acceder a una página que no está en memoria causa un *page-fault*.
  - Mientras no haya *page-faults*, el tiempo de acceso efectivo es igual al tiempo de acceso a memoria.
  - Si ocurre un *page-fault*, se debe primero leer la página desde almacenamiento secundario y luego accederla en memoria.
  - Entonces, el tiempo de acceso efectivo es directamente proporcional a la tasa de *page-faults*.

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Reemplazo de páginas

Atención de page-faults

Mecanismo (resumidísimo) para atender un *page-fault* con reemplazo de páginas:

1. Encontrar la página deseada en almacenamiento secundario.
2. Encontrar un frame libre:
   1. Si hay un frame libre, usarlo.
   2. Si no hay un frame libre, usar un algoritmo de reemplazo de páginas para seleccionar un frame *víctima*.
   3. Escribir el contenido del frame *víctima* a almacenamiento secundario (si hace falta) (en general, a espacio de swap); actualizar la tabla de páginas (y demás tablas).
3. Leer la página deseada en el recién liberado frame; actualizar la tabla de páginas (y demás tablas).
4. Continuar el proceso desde donde ocurrió el *page-fault*.

Reemplazo de páginas

Costo

- Utilizando reemplazo de páginas, si no hay frames libres se requieren 2 transferencias de páginas (una para la que sale y una para la que entra). Esto duplica el tiempo para atender *page-faults* y el tiempo de acceso efectivo.
- Mejora 1: no todas las páginas deben ser escritas a disco, (ej: páginas de sólo lectura).
- Mejora 2: usar un bit de modificación (*dirty bit*), y sólo escribir la página a disco si fue modificada desde que se leyó a memoria.

¿Cómo elegimos qué página desalojar?

FIFO

Asocia a cada página el tiempo en el que fue cargada en memoria. Cuando se tiene que reemplazar una página, se elige la más antigua.

Ejercicio

Tengo un sistema con 6 páginas y sólo 4 marcos de página. La memoria comienza vacía.
Llegan los siguientes pedidos de memoria (número de página) en ese orden:

1, 2, 1, 3, 4, 3, 5, 6, 2

Indicar qué página se desaloja tras cada pedido utilizando el algoritmo FIFO.

Solución

Pag Pedidas	Frames				Pags a desalojar
1	**1**				1
2	1	**2**			1 2
1	**1**	2			1 2
3	1	2	**3**		1 2 3
4	1	2	3	**4**	1 2 3 4
3	1	2	**3**	4	1 2 3 4
5	**5**	2	3	4	2 3 4 5
6	5	**6**	3	4	3 4 5 6
2	5	6	**2**	4	4 5 6 2

¿Cómo elegimos qué página desalojar?

LRU

Asocia a cada página el tiempo de la última vez que se usó. Cuando se tiene que reemplazar una página, se elige la que hace más tiempo que no se usa.

Ejercicio

Tengo un sistema con 6 páginas y sólo 4 marcos de página. La memoria comienza vacía.
Llegan los siguientes pedidos de memoria (número de página) en ese orden:

1, 2, 1, 3, 4, 3, 5, 6, 2

Indicar qué página se desaloja tras cada pedido utilizando el algoritmo LRU.

Solución

Pag Pedidas	Frames				Pags a desalojar
1	**1**				1
2	1	**2**			1 2
1	**1**	2			2 1
3	1	2	**3**		2 1 3
4	1	2	3	**4**	2 1 3 4
3	1	2	**3**	4	2 1 4 3
5	1	**5**	3	4	1 4 3 5
6	**6**	5	3	4	4 3 5 6
2	6	5	3	**2**	3 5 6 2

¿Cómo elegimos qué página desalojar?

Second Chance

Rotar a través de las páginas hasta encontrar la que no ha sido referenciada desde la última vez que chequeamos.
Cuando una página es seleccionada para desalojar, se mira el bit de referenciada. Si es 0, se desaloja. Si es 1, se le da una segunda oportunidad, se limpia el bit de referenciada, y se actualiza el tiempo de llegada con el tiempo actual.

Ejercicio

Tengo un sistema con 6 páginas y sólo 4 marcos de página. La memoria comienza vacía.
Llegan los siguientes pedidos de memoria (número de página) en ese orden:

1, 2, 1, 3, 4, 3, 5, 6, 2

Indicar qué página se desaloja tras cada pedido utilizando el algoritmo SC.

Solución

Pag Pedidas	Frames				Pags a desalojar			
1	**1**				1			
2	1	**2**			1	2		
1	**1**	2			1	2		
3	1	2	**3**		1	2	3	
4	1	2	3	**4**	1	2	3	4
3	1	2	**3**	4	1	2	3	4
5	1	2	3	4	2	3	4	1
	1	**5**	3	4	3	4	1	5
6	1	5	3	4	4	1	5	3
	1	5	3	**6**	1	5	3	6
2	**2**	5	3	6	5	3	6	2

Algoritmos de reemplazo

¿Qué algoritmo de reemplazo de páginas es es mejor?

- Los algoritmos se evalúan ejecutándolos sobre una secuencia particular de referencias a memoria y computando el número de *page-faults*.
- En general queremos el que tenga la menor tasa de *page-faults*.
- Page Fault Rate $= \frac{\text{páginas qué pedí y no estaban cargadas en memoria}}{\text{páginas totales}}$

Ejercicio

Para los ejercicios anteriores, calcular el *Page fault rate* en cada caso.

Solución

- Page Fault Rate (FIFO) $= 7 \ / \ 9$
- Page Fault Rate (LRU) $= 7 \ / \ 9$
- Page Fault Rate (SC) $= 7 \ / \ 9$

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Thrashing

Thrashing

Un proceso hace thrashing si pasa más tiempo cargando y descargando páginas que ejecutando procesos de usuario.

Ejemplo real:

1. Si el uso de CPU es muy bajo, el *scheduler* introduce más procesos al sistema.
2. Supongamos que un proceso empieza a necesitar más frames.
3. Con un enfoque de reemplazo de paginas global (reemplazando páginas sin importar a qué proceso pertenecen) el proceso empieza a fallar y a tomar frames de otros procesos.
4. Esos procesos necesitan esas páginas así que fallan y toman frames de otros procesos.
5. Y así, y así...
6. Mientras están todos esperando que se atiendan sus *page-faults*, la cola *ready* para ejecutar se vacía y decrementa el uso de CPU.
7. El *scheduler* ve esta baja en el uso de CPU e incrementa el grado de multiprogramación.
8. Y todo empeora. Y el sistema colapsa.

Thrashing

Mitigación

- En una computadora personal, un usuario puede darse cuenta de que está ocurriendo *thrashing* y matar algunos procesos "a mano".
- Desde el SO, se pueden limitar los efectos del *thrashing* usando un reemplazo de páginas local, es decir, que cada proceso sólo pueda tomar *frames* de los que ya tiene asignados, y no pueda "robarle" a otro proceso.
- Pero para prevenir el *thrashing* deberíamos proveer a un proceso de tantos *frames* como necesite.

Localidad

¿Cuántos *frames* necesita un proceso? $\rightarrow$ definir un modelo de localidad de la ejecución.

Localidad

Una localidad es un conjunto de páginas que se usan activamente al mismo tiempo.
El modelo de localidad dice que mientras un proceso ejecuta, se mueve de localidad a localidad.

- Supongamos que asignamos suficientes *frames* a un proceso para acomodar su localidad actual.
- Fallará por páginas en esa localidad hasta que estén todas en memoria, y luego ya no fallará hasta que cambie de localidad.
- Si no asignamos suficientes frames para el tamaño de su localidad actual, el proceso hará *thrashing*.

Momento para preguntas

Menú para hoy

1. Introducción
2. Memoria contigua
3. Memoria virtual
4. Paginación a Demanda
5. Reemplazo de páginas
6. Thrashing
7. Cierre

Resumen

Hoy vimos...

- Asignación de memoria contigua. ¡Produce fragmentación externa!
- Memoria virtual
  - Repaso base+límite
  - Repaso segmentación. ¡Produce fragmentación externa!
  - Repaso paginación. ¡Produce fragmentación interna!
- Paginación a demanda
- Algoritmos de reemplazo de páginas
  - FIFO
  - LRU
  - Second Chance
- Thrashing y modelo de localidad

Cómo seguimos...

Con esto se puede resolver toda la guía práctica 4.