

Memoria virtual 10

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Memoria virtual

Contenido

- Generalidades.
- Paginación por demanda.
- *Copy-on-Write*.
- Reemplazo de páginas.
- *Thrashing*.
- Asignación de memoria del *kernel*.
- Otras consideraciones.
- Ejemplos de sistemas operativos.

Memoria virtual

- La memoria virtual es una técnica que permite la ejecución de procesos que **no están completamente en memoria**.
- Abstrae la memoria principal en una matriz de almacenamiento extremadamente grande y uniforme, **separando la memoria lógica vista por el programador de la memoria física**.
- También permite que los procesos compartan archivos, bibliotecas y memoria compartida.
- Sin embargo, no es fácil de implementar y puede disminuir sustancialmente el rendimiento del sistema si no se utiliza con precaución.

3

SOYD 2020 □ Gustavo C. Distel

Memoria virtual

Generalidades

- Los temas de memoria principal descritos son necesarios debido a un requisito básico: las **instrucciones a ejecutar deben estar en memoria física**.
- Este requisito limita el tamaño de un programa al de la memoria física. Sin embargo, en muchos casos no se necesita todo el programa. Por ej.:
 - En general tienen código para manejar condiciones de error inusuales. Debido a que rara vez ocurren, este código casi nunca se ejecutará.
 - A menudo las matrices, listas y tablas tienen asignada más memoria de la que realmente necesitan. Se puede declarar una matriz de **100** por **100** elementos, aunque rara vez sea mayor que **10** por **10** elementos.
 - Ciertas opciones y características de un programa rara vez se usan (por ej.: el *help*).
- Incluso en aquellos casos en los que se necesita todo el programa, es posible que no sea necesario todo al mismo tiempo.

4

SOYD 2020 □ Gustavo C. Distel

Memoria virtual

Generalidades

- La capacidad de ejecutar programas que están parcialmente en memoria tiene los siguientes beneficios:
 - Un programa no está limitado por la cantidad de memoria física. Los usuarios pueden **escribir programas para un espacio de direcciones virtuales extremadamente grande**, simplificando la tarea de programación.
 - Debido a que cada programa consume menos memoria física, **se pueden ejecutar más programas al mismo tiempo**, con un aumento en la utilización y el rendimiento de la *CPU*, y sin incrementar el tiempo de respuesta o el tiempo de retorno.
 - Es necesario **menos E/S** para cargar o intercambiar porciones de programas en memoria, por lo que cada programa se ejecuta más rápido.

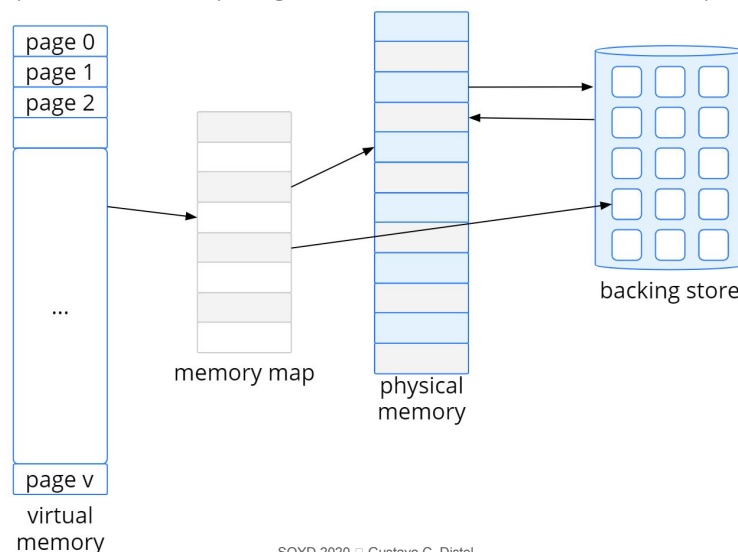
5

SOYD 2020 □ Gustavo C. Distel

Memoria virtual

Generalidades

- La **memoria virtual** implica la separación de la memoria lógica percibida por los desarrolladores de la memoria física.
- Esta separación proporciona una memoria virtual extremadamente grande para los programadores, aunque solo se disponga de una memoria física más pequeña.



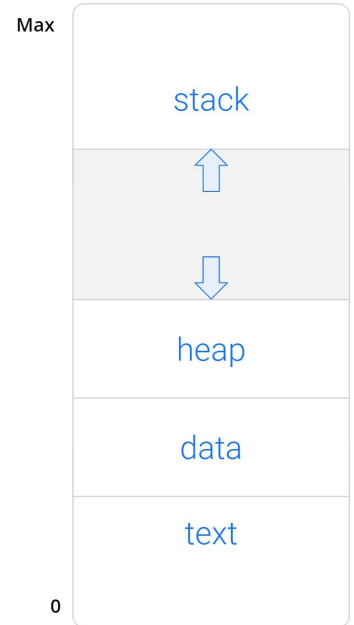
6

SOYD 2020 □ Gustavo C. Distel

Memoria virtual

Generalidades

- El **espacio de direcciones virtuales** de un proc. es la vista lógica (o virtual) de cómo se almacena en memoria; esta vista comienza en una determinada dirección lógica y crece en memoria contigua.
- Sin embargo, la memoria física está organizada en marcos y los marcos asignados a un proc. pueden ser no contiguos.
- El espacio entre el *heap* y el *stack* es parte del espacio de direcciones virtuales.
- Los espacios de direcciones virtuales que incluyen huecos se conocen como espacios de direcciones **dispersos (*sparse*)**.
- La ventaja es que los huecos se pueden llenar a medida que crecen los segmentos o si se vinculan bibliotecas dinámicamente durante la ejecución de un programa.



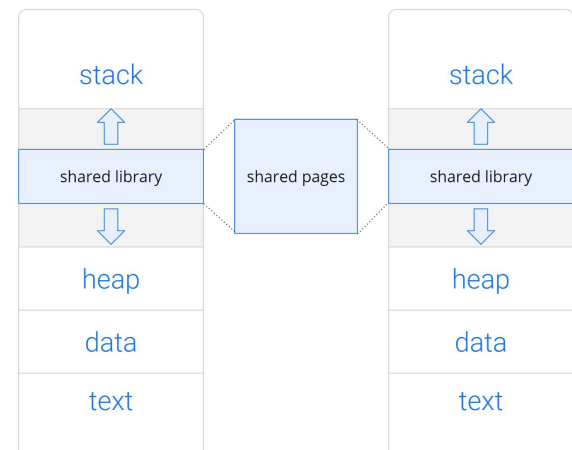
7

SOYD 2020 □ Gustavo C. Distel

Memoria virtual

Generalidades

- La memoria virtual permite que dos o más procesos compartan archivos y memoria mediante el uso compartido de páginas, por ej.:
 - Compartir bibliotecas del sistema, como la estándar **C**.
 - Permitir que un proceso cree una región de memoria que pueda compartir.
 - También se pueden compartir páginas durante la creación de un proceso al usar **fork()**, acelerando así su creación.



8

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Conceptos básicos

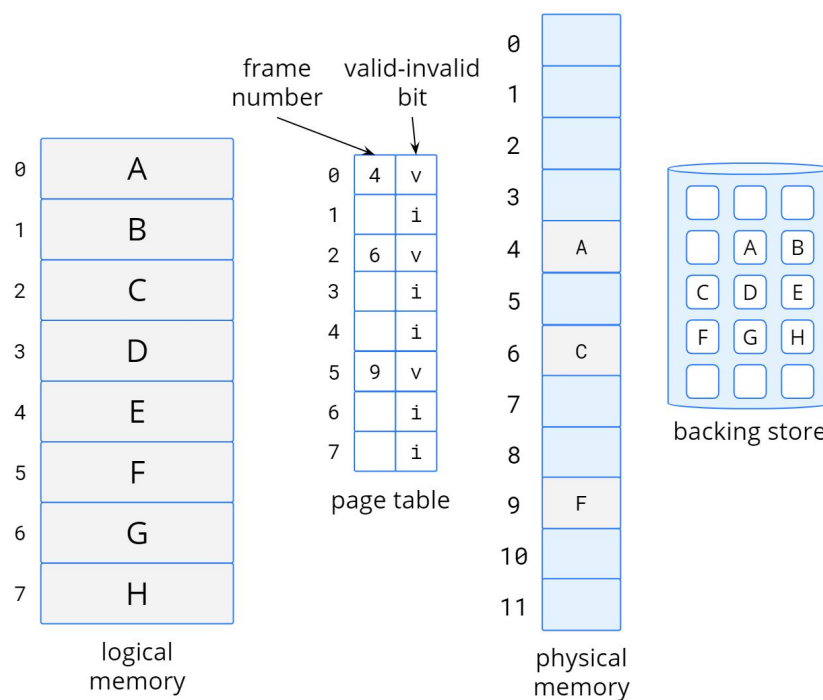
- El concepto general detrás de la paginación por demanda es cargar una página en memoria solo cuando es necesario (cuando es **demandada**).
- Como resultado, mientras se ejecuta un proceso, algunas páginas estarán en memoria y otras estarán en almacenamiento secundario.
- Se necesita soporte de *HW* para hacer esta distinción; *bits válido-invalido*:
 - **Válido**: la página asociada es legal y está en memoria.
 - **Inválido**: la página no es válida (es decir, no está en el espacio de direcciones lógicas del proceso), o es válida pero está en almacenamiento secundario.
- La entrada de la TPs para una página que no está actualmente en memoria simplemente se marca como **inválida**.
- Marcar una página como **inválida** no tendrá ningún efecto si el proceso nunca intenta acceder a esa página.

9

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Conceptos básicos



10

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Conceptos básicos

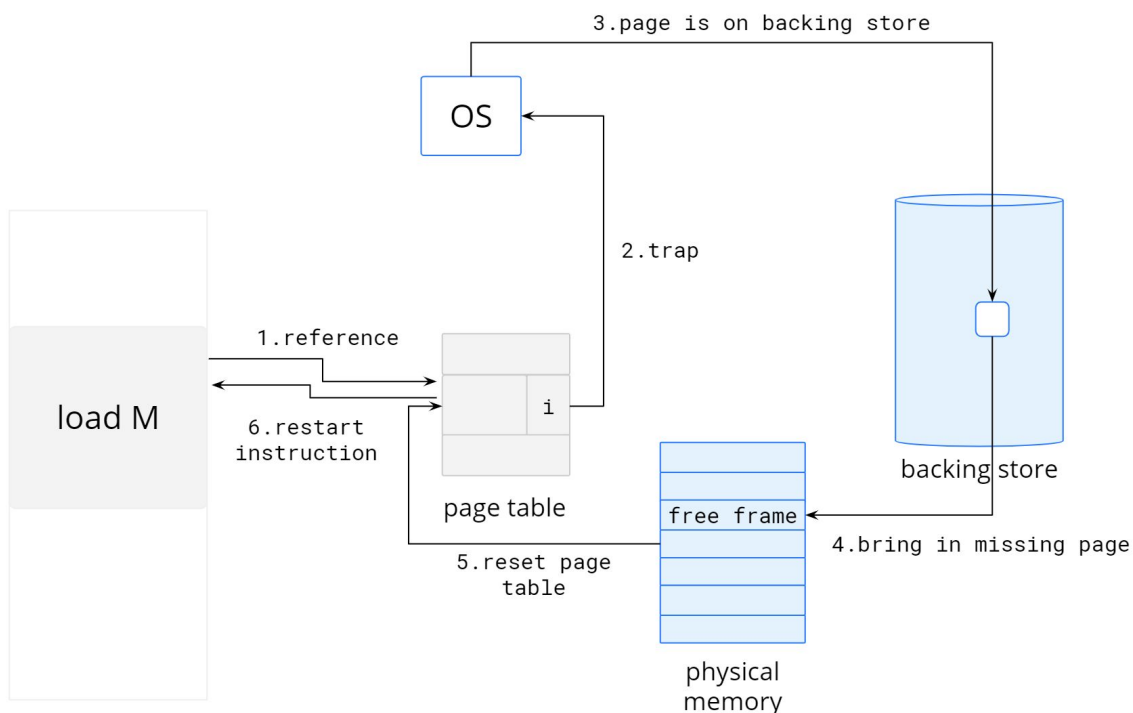
- El acceso a una página marcada como **inválida** provoca un **page fault** → *trap* al SO.
- Procedimiento para manejar un **page fault**:
 - 1. Se verifica una tabla interna del proceso (generalmente en el *PCB*) para determinar si la referencia es a un acceso **válido** o **inválido**.
 - 2. Si la referencia es **inválida** se finaliza el proceso. Si es **válida** pero aún no se ha traído esa página, se trae.
 - 3. Se busca un marco libre (tomando uno de la lista de marcos libres).
 - 4. Se planifica una operación en el almacenamiento secundario para leer la página deseada al marco recién asignado.
 - 5. Cuando se completa la lectura, se modifica la tabla interna mantenida con el proceso y la tabla de páginas para indicar que la página ahora está en memoria.
 - 6. Se reinicia la instrucción que fue interrumpida por el *trap*.
- El proceso ahora puede acceder a la página como si siempre hubiera estado en memoria.

11

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Conceptos básicos



12

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Conceptos básicos

- **Paginación por demanda pura:** no traer nunca una página a memoria hasta que sea requerida.
 - Un proceso se comienza a ejecutar sin páginas en memoria.
 - La primera instrucción a ejecutar provocará un *page fault*.
 - Después de que esta página se lleva a memoria, el proceso continúa ejecutándose, provocando *page faults* según sea necesario hasta que cada página que necesita esté en memoria.
 - En un determinado momento puede ejecutarse sin más *page faults*.
- Los programas tienden a tener una **localidad de referencia**, descrita más adelante, que resulta en un rendimiento razonable de la paginación de la demanda.
- El *HW* para soportar la paginación por demanda es el mismo que el *HW* de paginación y swapping:
 - **Tabla de páginas:** para marcar una entrada con un *bit* válido-inválido u otro valor de *bits*.
 - **Memoria secundaria:** contiene las páginas no presentes en memoria principal, se denomina *swap device*, y la sección utilizada para paginación es el *swap space*.

13

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Lista de marcos libres

- La mayoría de los SOs mantienen una lista de marcos libres para satisfacer las solicitudes de los *page faults*.



- Los SOs generalmente asignan marcos libres utilizando una técnica denominada **zero-fill-on-demand**.
- Los marcos **zero-fill-on-demand** se "ponen a cero (**zero-out**)" antes de ser asignados, borrando así su contenido anterior (considerar las posibles implicaciones de **seguridad** de no borrar el contenido de un marco antes de reasignarlo).
- Cuando se inicia un sistema, toda la memoria disponible se coloca en la lista de marcos libres.
- A medida que se solicitan marcos libres el tamaño de la lista se reduce. En algún momento, la lista cae a cero o cae por debajo de un cierto umbral, en cuyo punto debe rellenarse nuevamente.

14

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Performance de la paginación por demanda

- La paginación por demanda puede afectar significativamente el rendimiento de un sistema, por lo que se calcula el **tiempo de acceso efectivo**.
- Considerar por ej. que el tiempo de acceso a memoria (**ma**) de **10 nanosegundos**, mientras no haya *page faults*:
 - **tiempo de acceso efectivo = tiempo de acceso a memoria.**
- Sin embargo, si ocurre un *page fault*, primero se debe leer la página del almacenamiento secundario y luego acceder a la palabra deseada.
- Sea **p** la probabilidad de un *page fault* ($0 \leq p \leq 1$). Se espera que **p** sea cercano a **0**, es decir, esperaríamos tener solo unos pocos *page faults*.
 - **Tiempo de acceso efectivo = $(1 - p) \times ma + p \times \text{page fault time}$.**
- Para calcular el tiempo de acceso efectivo, se debe saber cuánto tiempo se necesita para resolver un *page fault*.

15

SOYD 2020 □ Gustavo C. Distel

Paginación por demanda

Performance de la paginación por demanda

- Existen al menos tres componentes principales involucrados en el tiempo de servicio de un *page fault*:
 - **1.** Servir la interrupción del *page fault*.
 - **2.** Leer la página.
 - **3.** Reiniciar el proceso.
- Con un tiempo promedio de servicio de *page fault* de **8 milisegundos** y un tiempo de acceso a memoria de **200 nanosegundos**, el tiempo de acceso efectivo en **nanosegundos** es:
 - **$effective\ access\ time = (1 - p) \times (200) + p (8\ milisegundos)$**
 - **$= (1 - p) \times 200 + p \times 8,000,000$**
 - **$= 200 + 7,999,800 \times p.$**
- Por lo tanto el tiempo de acceso efectivo es directamente proporcional a la **tasa de *page fault***.

16

SOYD 2020 □ Gustavo C. Distel

Copy-on-Write

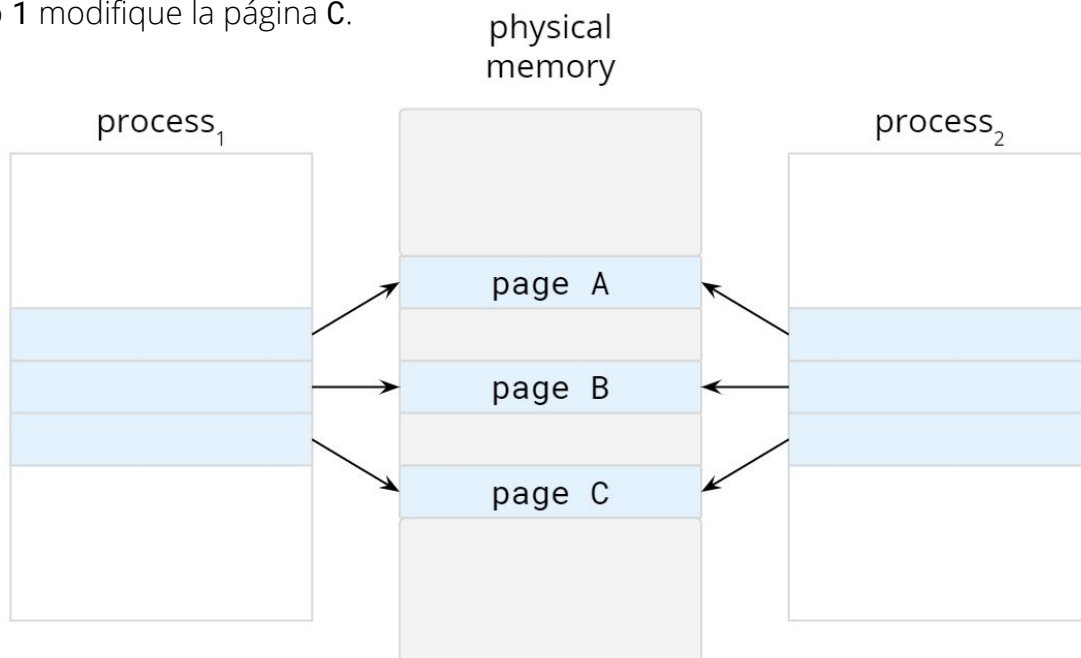
- La creación de procesos con **fork()** puede evitar la paginación por demanda mediante una técnica similar al uso compartido de páginas.
- Esta técnica proporciona una creación rápida y minimiza el número de páginas nuevas que deben asignarse al proceso recién creado.
- Tradicionalmente, **fork()** funciona creando una copia del espacio de direcciones del padre al hijo, duplicando las páginas que pertenecen al padre.
- Sin embargo, teniendo en cuenta que muchos procesos hijos invocan **exec()** luego de la creación, la copia del espacio de direcciones del padre puede ser innecesaria.
- En su lugar se puede usar **copy-on-write**, que permite que los procesos padre e hijo compartan inicialmente las mismas páginas.
- Las páginas compartidas están marcadas **copy-on-write**, lo que significa que si cualquiera de los procesos escribe en éstas, se crea una copia de la página compartida.

17

SOYD 2020 □ Gustavo C. Distel

Copy-on-Write

- **copy-on-write**: a continuación se muestran el contenido de la memoria física antes de que el proceso 1 modifique la página C.

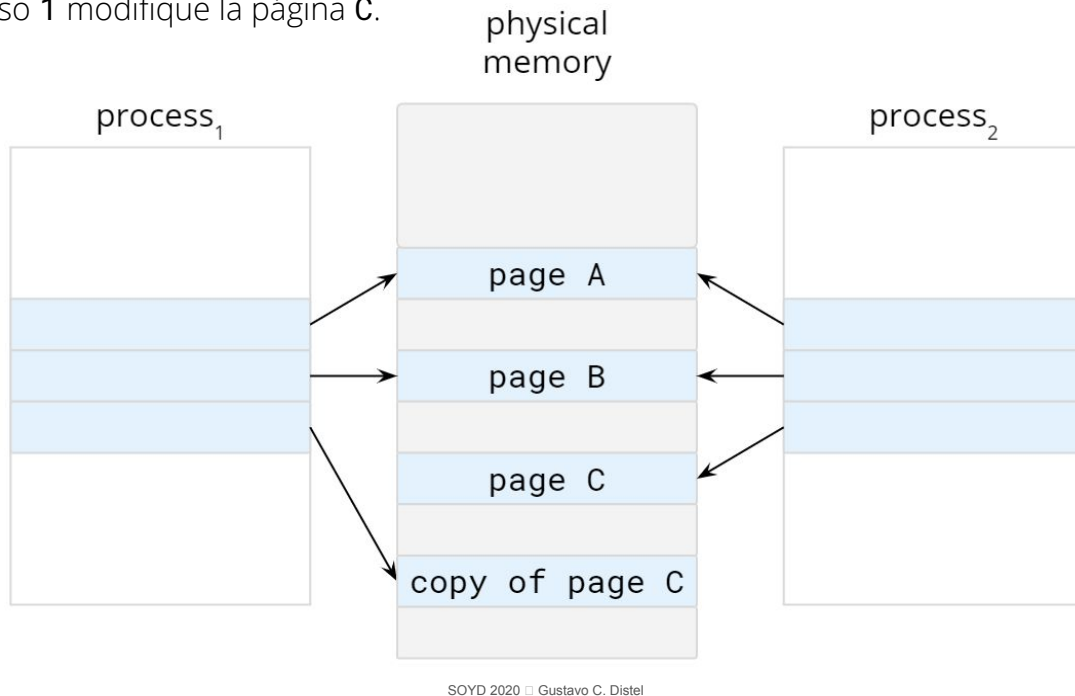


18

SOYD 2020 □ Gustavo C. Distel

Copy-on-Write

- **copy-on-write**: a continuación se muestran el contenido de la memoria física después de que el proceso 1 modifique la página C.



19

Copy-on-Write

- Por ej., sea un proceso hijo que intenta modificar una página que tiene partes de la pila con páginas marcadas **copy-on-write**.
 - El SO obtendrá un marco de la lista de marcos libres y creará una copia de esta página, asignándola al espacio de direcciones del proceso hijo.
 - El proceso hijo modificará su página copiada y no la página que pertenece al proceso padre.
- Al utilizar **copy-on-write** solo se copian las páginas que se modifican, las páginas no modificadas se comparten por los procesos padre e hijo.
- Las páginas que se pueden modificar se marcan como **copy-on-write**. Las páginas que no pueden modificarse (páginas que contienen código ejecutable) se comparten por el padre y el hijo.
- **copy-on-write** es una técnica común utilizada por varios SOs, incluidos **Windows**, **Linux** y **macOS**.

20

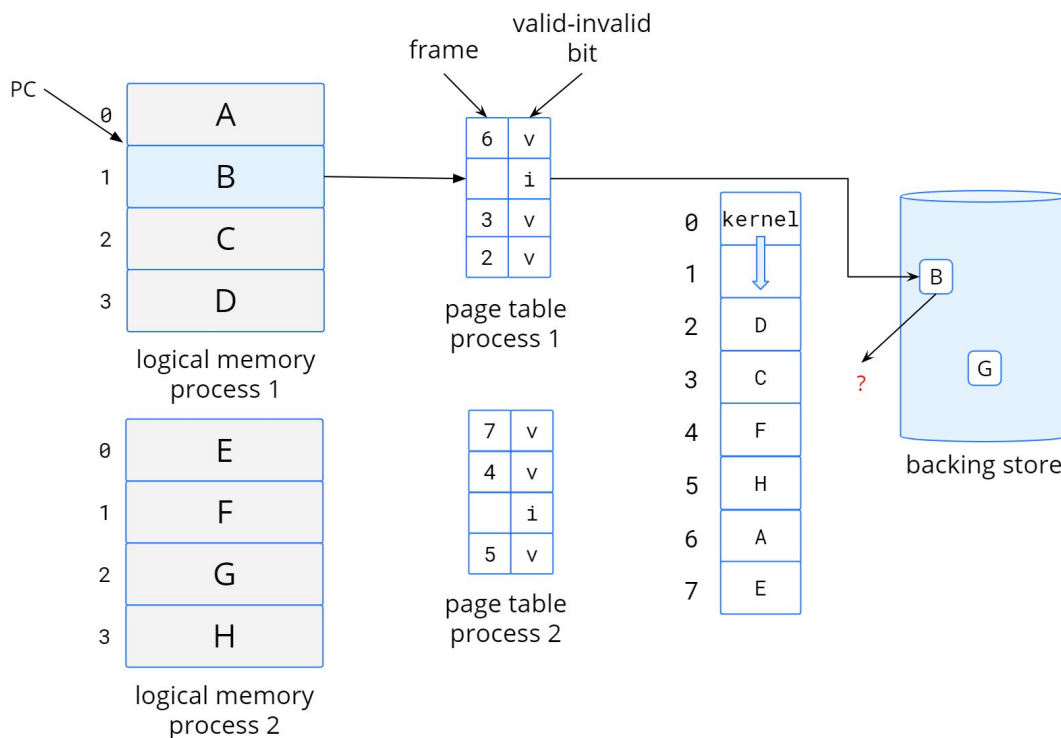
Reemplazo de páginas

- Si se aumenta el grado de multiprogramación, se puede producir **una sobre asignación de la memoria**.
- En un sistema con **40** marcos, si se ejecutan **6** procesos con **10** páginas c/u pero en realidad c/u usa **5** páginas, se tiene mayor utilización y rendimiento de la *CPU*, con **10** marcos excedentes.
- Sin embargo, es posible que c/u de estos procesos de repente intente usar las **10** páginas, lo que resulta en la necesidad de **60** marcos cuando en realidad solo hay **40** disponibles.
- Además, la memoria del sistema no se usa solo para páginas de programas (*Buffers* de E/S).
- La sobreasignación de memoria se manifiesta de la siguiente manera:
 - Mientras se ejecuta un proceso, se produce un *page fault*.
 - El SO determina dónde reside la página deseada en el almacenamiento secundario, pero no hay marcos libres (imagen).
- El SO puede terminar el proceso o utilizar estándar *swapping*, pero no son las mejores opciones.
- La mayoría de los SOs combinan *swapping* de páginas con **reemplazo de páginas**.

21

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas



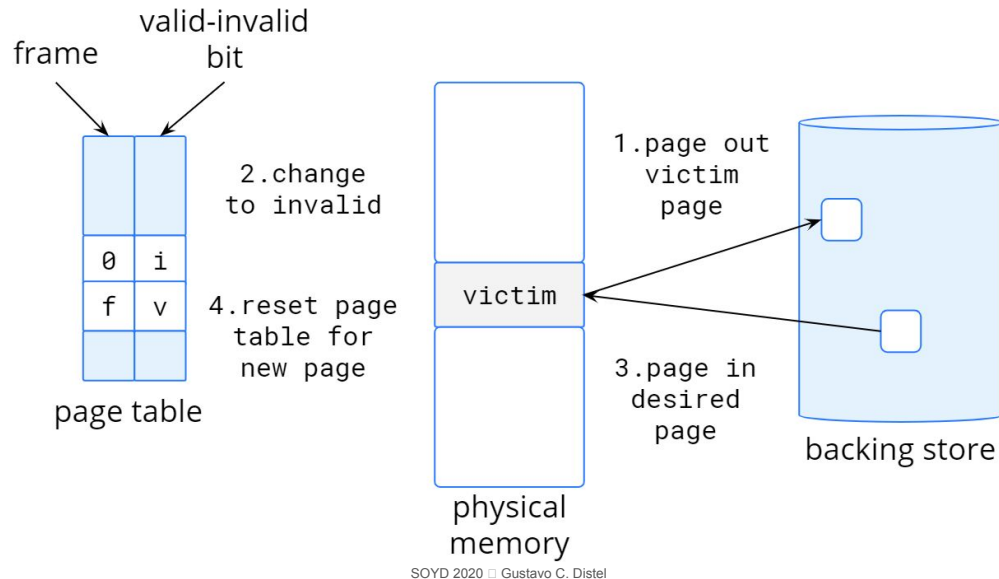
22

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas básico

- Si no hay ningún marco libre, se busca uno que no se esté utilizando y se libera.
- Se escribe su contenido en *swap* y se modifica la TP indicando que la página ya no está en memoria; a continuación se usa el marco liberado para la página por la cual el proceso falló.



23

Reemplazo de páginas

Reemplazo de páginas básico

- Se modifica la rutina de servicio de *page fault* para incluir el reemplazo de la página:
 - 1. Encontrar la ubicación de la página deseada en almacenamiento secundario.
 - 2. Encontrar un marco libre:
 - a. Si hay un marco libre, usarlo.
 - b. Si no hay un marco libre, usar un **algoritmo de reemplazo de página** para seleccionar un **marco víctima**.
 - c. Escribir el **marco víctima** en almacenamiento secundario (si es necesario); modificar la tabla de páginas y la tabla de marcos.
 - 3. Leer la página deseada en el marco recién liberado; modificar la tabla de páginas y la tabla de marcos.
 - 4. Continuar con la ejecución del proceso desde donde ocurrió el *page fault*.

24

Reemplazo de páginas

Reemplazo de páginas básico

- Si no hay marcos libres, se requieren dos transferencias de página (*page-out* y *page-in*); esto duplica el tiempo de servicio de *page fault* y aumenta el tiempo de acceso efectivo.
- Se puede reducir el *overhead* al usar un **bit de modificación** (*modify bit* o *dirty bit*) en cada página.
 - El *HW* modifica el *bit* cada vez que se escribe un *byte* en la página, indicando que ha sido modificada.
- Cuando se selecciona una página para reemplazar, se examina su **bit de modificación**.
 - **Si está seteado**; la página se ha modificado desde cuando se leyó desde el almacenamiento secundario → se debe escribir la página en el almacenamiento.
 - **Si no está seteado**; la página no se ha modificado desde cuando se leyó en memoria → no se necesita escribir la página de memoria en el almacenamiento.
- Las páginas de solo lectura (por ej., páginas con código binario), no pueden ser modificadas, por lo tanto se pueden descartar.

25

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas básico

- Se deben resolver dos problemas principales para implementar la paginación por demanda:
 - Un algoritmo de asignación de marcos.
 - Un algoritmo de reemplazo de página.
- Hay muchos algoritmos de reemplazo de página; cada SO tiene su propio esquema de reemplazo. En general, se selecciona el alg. que tenga la tasa de *page fault* más baja.
- Se evalúa un algoritmo ejecutándolo en una cadena particular de referencias de memoria (**cadena de referencia - reference string**) y calculando el número de *page faults*.
 - Solo se considera el número de página, en lugar de la dirección completa.
 - Para una referencia a una página **p**, cualquier referencia siguiente a la página **p** no causará *page fault*.

26

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas básico

- Por ej., si se traza un proceso en particular y se tiene la siguiente secuencia de direcciones:
 - 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105
- A 100 bytes por página, esta secuencia se reduce a la siguiente cadena de referencia:
 - 1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1
- Para determinar la cantidad de *page faults* para una cadena de referencia particular y un algoritmo de reemplazo de página, también necesitamos saber la cantidad de marcos de página disponibles.
 - Es claro que a medida que aumenta el número de marcos disponibles, disminuye el número de *page faults*.

27

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas FIFO (First-in, First-out)

- El algoritmo de reemplazo *FIFO* asocia cada página con el momento en que esa página fue traída a memoria. Cuando se debe reemplazar una página, se elige la página más antigua.
- No es estrictamente necesario registrar el momento en que ingresa una página.
 - Se puede crear una cola *FIFO* para mantener todas las páginas en memoria.
 - Reemplazamos la página en la cabeza de la cola.
 - Cuando una página se trae a memoria, se inserta al final de la cola.
- Ejemplo:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Pages faults	F	F	F	F		F	F	F	F	F	F			F	F			F	F	F

28

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *FIFO* (*First-in, First-out*)

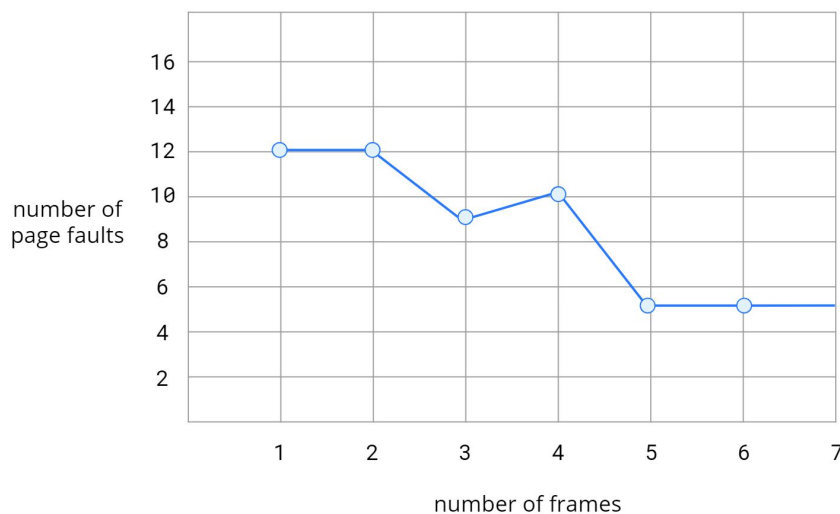
- El algoritmo de reemplazo de página *FIFO* es **fácil de entender y programar**. Sin embargo, **su rendimiento no siempre es bueno**.
- La página reemplazada puede ser un módulo de inicialización que se utilizó hace mucho tiempo y ya no es necesario, o podría contener una variable muy utilizada que se inicializó temprano y está en uso constante.
- Incluso si se selecciona una página activa para reemplazar, todo continúa funcionando correctamente.
 - Después de reemplazar una página activa por una nueva, ocurre un *page fault* de inmediato para recuperar la página activa (reemplazando a su vez otra página).
- Por lo tanto, una mala elección de reemplazo aumenta la tasa de *pages faults* y ralentiza la ejecución del proceso.
- Para ilustrar otros problemas, considere el siguiente *string* de referencia:
 - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

29

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *FIFO* (*First-in, First-out*)



- El número de fallos para 4 marcos (10) es mayor que el número de fallos para 3 (9).
- **Anomalía de Belady:** para algunos algoritmos de reemplazo de página, la tasa de *page fault* puede aumentar a medida que aumenta el número de marcos asignados.

30

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *OPT (Optimal)*

- Reemplaza la página que no se utilizará durante el período de tiempo más largo.
- Este algoritmo garantiza la tasa de *page faults* más baja posible para un número fijo de marcos; además no sufre de la anomalía de Belady.

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	2	2	3	2	2	2	2	2	2	2	7	7	7
		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Pages faults	F	F	F	F		F		F			F			F				F		

- Solo **9** *page faults* (mucho mejor que *FIFO*, que da **15** *page fault*).
- Desafortunadamente, el algoritmo óptimo de reemplazo de página es difícil de implementar, ya que requiere un conocimiento futuro de la cadena de referencia (idem planificación *S/F*).
- El algoritmo óptimo se utiliza principalmente para realizar estudios de comparación con otros alg.

31

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *LRU (Least recently used)*

- La distinción entre los algoritmos *FIFO* y *OPT* (aparte de mirar hacia atrás vs. hacia adelante en el tiempo) es que:
 - el alg. *FIFO* usa el tiempo en que una página fue traída a la memoria, mientras que,
 - el alg. *OPT* usa el tiempo en el cual se va a usar una página.
- Si utilizamos el pasado reciente como una aproximación del futuro cercano, entonces podemos reemplazar la página que no se ha utilizado durante el período de tiempo más largo.
- El reemplazo *LRU* asocia cada página con el momento de su último uso.
 - Cuando se debe reemplazar una página, *LRU* elige la página que no se ha utilizado durante el período de tiempo más largo.
- Podemos pensar en esta estrategia como el algoritmo óptimo de reemplazo de página que mira hacia atrás en el tiempo, en lugar de hacia adelante.

32

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *LRU* (*Least recently used*)

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Pages faults	F	F	F	F		F		F	F	F	F			F		F		F		

- 12 *page faults* (mejor que *FIFO* con 15). Además no sufre de la anomalía de Belady.
- La política de *LRU* se usa a menudo como un algoritmo de reemplazo de página y se considera adecuada. El principal problema es cómo implementarla.
- Un algoritmo de reemplazo de página *LRU* puede requerir asistencia de *HW*.
 - El problema es determinar un orden para los marcos definidos por el último uso.
- Dos implementaciones son factibles:

33

SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *LRU* (*Least recently used*)

- Contadores:
 - Se agrega a cada entrada de la TP un campo de **tiempo de uso** y se agrega a la *CPU* un reloj o contador lógico, el cual incrementa en cada referencia a memoria.
 - A su vez, cuando se referencia una página, el contenido del registro del reloj se copia al campo **tiempo de uso**.
 - De esta manera, siempre tenemos el "tiempo" de la última referencia a cada página.
 - Se reemplaza la página con el menor valor de tiempo.
 - Se requiere una búsqueda en la TP para encontrar la página *LRU* y una escritura en memoria por cada acceso.
 - Los tiempos también deben mantenerse cuando se cambian las tablas de páginas (debido a la planificación de la *CPU*).
 - También se debe considerar el desbordamiento del reloj.

34

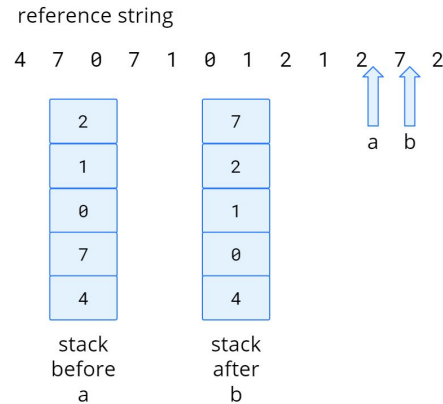
SOYD 2020 □ Gustavo C. Distel

Reemplazo de páginas

Reemplazo de páginas *LRU* (*Least recently used*)

- Pila:

- Mantener una pila con los números de página.
 - Cada vez que se hace referencia a una página, se elimina de la pila y se coloca en la parte superior.
- La página más utilizada recientemente siempre está en la parte superior, y la página menos utilizada en la parte inferior.
- Debido a que las entradas deben eliminarse del medio de la pila, es mejor implementarla utilizando una lista doblemente enlazada con un puntero a la cabeza y uno a la cola.
- Cada actualización es costosa, pero no se busca un reemplazo; el puntero de la cola apunta a la parte inferior de la pila, que es la página *LRU*.



35

SOYD 2020 □ Gustavo C. Distel

Thrashing

- Considere lo que ocurre si un proceso no tiene suficientes marcos para su ejecución.
 - El proceso provocará un *page fault*, reemplazando alguna página.
 - Dado que todas sus páginas están activas, deberá reemplazar una página que necesitará de inmediato.
 - En consecuencia, falla una y otra vez, reemplazando las páginas que deberá traer nuevamente.
- Este alto nivel de paginación se denomina *thrashing*.
 - Un proceso se encuentra en *thrashing* si se la pasa más tiempo paginando que ejecutando.
 - El *thrashing* provoca graves problemas de rendimiento.

36

SOYD 2020 □ Gustavo C. Distel

Thrashing

Causa

- El SO monitorea la utilización de la *CPU*.
 - Si la utilización de la *CPU* es baja, se aumenta el grado de multiprogramación introduciendo un nuevo proceso en el sistema.
- A su vez, se utiliza un **alg. global de reemplazo de página**, que reemplaza las páginas sin tener en cuenta el proceso al que pertenecen.
- Ahora supongamos que un proceso entra en una nueva fase de ejecución y necesita más marcos;
 - Comenzará a realizar *page faults* y a quitar marcos a otros procesos.
- Sin embargo, estos otros procesos también necesitan esas páginas, por lo que también fallan, quitando marcos de otros procesos.
- A su vez, los procesos que fallan deben usar el dispositivo de paginación para intercambiar (*swap*) páginas.
- A medida que se encolan en el dispositivo de paginación, **la cola de listos se vacía y la utilización de la *CPU* disminuye**.

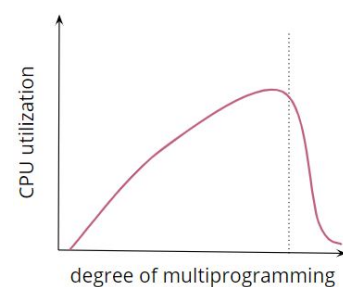
37

SOYD 2020 □ Gustavo C. Distel

Thrashing

Causa

- El planificador de la *CPU* detecta una disminución en la utilización de la *CPU* y aumenta el grado de multiprogramación.
- El nuevo proceso toma marcos de los procesos en ejecución, causando más *page faults* y una cola más larga en el dispositivo de paginación.
- Como resultado, la utilización de la *CPU* disminuye aún más, y el planificador de la *CPU* intenta aumentar aún más el grado de multiprogramación.



- **Se produjo thrashing y el rendimiento del sistema se desploma.**
- La tasa de *page fault* aumenta enormemente. Como resultado, aumenta el tiempo efectivo de acceso a memoria.
- El resultado es que no se realiza ningún trabajo, porque los procesos están dedicando todo su tiempo a la paginación.
- En este punto, para aumentar la utilización de la *CPU* y detener el thrashing, **se debe disminuir el grado de multiprogramación**.

38

SOYD 2020 □ Gustavo C. Distel

Thrashing

Causa

- Podemos limitar los efectos del *thrashing* mediante el uso de un [algoritmo de reemplazo local](#).
 - Requiere que cada proceso solo seleccione marcos de su conjunto de marcos asignados.
 - Por lo tanto, si un proceso comienza a hacer *thrashing*, no puede robar marcos de otro proceso y hacer que este último también caiga en *thrashing*.
- El problema en parte continúa, ya que los procesos en *thrashing* estarán en la cola de paginación la mayor parte del tiempo.
 - El tiempo de servicio promedio por *page fault* aumentará debido a la cola → el tiempo de acceso efectivo aumentará para todos los procesos.
- Para evitar el *thrashing* debemos proporcionar a un proceso con tantos marcos como sea necesario, observando cuántos marcos está utilizando actualmente → [modelo de localidad de ejecución del proceso](#).
 - Establece que, a medida que se ejecuta un proceso, se mueve de una localidad a otra.
 - Una localidad es un conjunto de páginas que se usan juntas activamente.

39

SOYD 2020 □ Gustavo C. Distel

Thrashing

Causa

- Un programa en ejecución generalmente se compone de varias localidades diferentes, que pueden superponerse.
- Por ej., cuando se llama a una función, se define una nueva localidad.
 - En esta localidad se hacen referencias de memoria a las instrucciones de la llamada a la función, sus variables locales y un subconjunto de las variables globales.
- Cuando sale de la función, el proceso abandona esta localidad, ya que las variables locales y las instrucciones de la función ya no están en uso activo.
- Más tarde se puede regresar a esta misma localidad.

40

SOYD 2020 □ Gustavo C. Distel

Thrashing

Working-Set Model

- El **modelo de conjunto de trabajo** se basa en el supuesto de localidad, utilizando un parámetro, Δ , para definir la **ventana del conjunto de trabajo**.
- La idea es examinar las referencias de página Δ más recientes, las cuales conforman el conjunto de trabajo.
- Si una página está activa, estará en el conjunto de trabajo, y si ya no se usa, caerá del conjunto de trabajo Δ unidades de tiempo después de su última referencia. Por lo tanto, el conjunto de trabajo es una aproximación de la localidad del programa.
- Por ej., si $\Delta = 10$ referencias de memoria, entonces el conjunto de trabajo en el tiempo t_1 es $\{1, 2, 5, 6, 7\}$. Para el tiempo t_2 , el conjunto de trabajo ha cambiado a $\{3, 4\}$.

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



SOYD 2020 □ Gustavo C. Distel

41

Thrashing

Working-Set Model

- La precisión del conjunto de trabajo depende de la selección del Δ .
 - Si el Δ es demasiado pequeño, no abarcará toda la localidad.
 - Si el Δ es demasiado grande, puede superponerse a varias localidades.
 - Si el Δ es infinito, el conjunto de trabajo es el conjunto de páginas utilizadas durante la ejecución del proceso.
- La propiedad más importante del conjunto de trabajo es entonces su tamaño. Si calculamos el tamaño del conjunto de trabajo WSS_i , para cada proceso en el sistema, entonces podemos considerar que:
 - $D = \sum WSS_i$, donde D es la demanda total de marcos.
- Cada proceso está utilizando activamente las páginas de su conjunto de trabajo. Por lo tanto, el proceso necesita WSS_i marcos.
- Si la demanda total es mayor que el número total de marcos disponibles ($D > m$) **se producirá thrashing**, porque algunos procesos no tendrán suficientes marcos.

SOYD 2020 □ Gustavo C. Distel

42

Thrashing

Working-Set Model

- Una vez que se ha seleccionado el Δ , el uso del modelo de conjunto de trabajo es simple.
 - El SO monitorea el conjunto de trabajo de cada proceso y le asigna suficientes marcos como para proporcionarle su tamaño de conjunto de trabajo.
 - Si hay suficientes marcos adicionales, se puede iniciar otro proceso.
 - Si la suma de los tamaños de los conjuntos de trabajo aumenta, excediendo el número total de marcos disponibles, el SO selecciona un proceso para suspender.
 - Las páginas del proceso se escriben (*swapped*) y sus marcos se reasignan a otros procesos.
 - El proceso suspendido se puede reiniciar más tarde.

43

SOYD 2020 □ Gustavo C. Distel

Thrashing

Working-Set Model

- Esta estrategia de conjunto de trabajo evita el *thrashing* mientras mantiene el grado de multiprogramación lo más alto posible.
- Por lo tanto, optimiza la utilización de la *CPU*.
- La dificultad con este modelo es hacer un seguimiento del conjunto de trabajo.
- La ventana del conjunto de trabajo es una ventana móvil. En cada referencia de memoria, aparece una nueva referencia en un extremo, y la referencia más antigua cae en el otro extremo.
- Una página está en el conjunto de trabajo si se hace referencia a ella en cualquier parte de la ventana.
- Se puede aproximar el modelo del conjunto de trabajo con una interrupción del temporizador de intervalo fijo y un *bit* de referencia.

44

SOYD 2020 □ Gustavo C. Distel

Asignación de memoria del *kernel*

- La memoria del *kernel* a menudo se asigna desde un grupo de memoria libre diferente a la utilizada para satisfacer los procesos en modo usuario.
- Hay dos razones principales para esto:
 - 1. El *kernel* solicita memoria para estructuras de datos de diferentes tamaños, algunos de los cuales tienen menos de una página de tamaño.
 - Como resultado, el *kernel* debe usar la memoria de forma conservadora e intentar minimizar el desperdicio debido a la fragmentación.
 - Esto es importante porque muchos SOs no someten el código del *kernel* o sus datos al sistema de paginación.
 - 2. Ciertos dispositivos de *HW* interactúan directamente con memoria física, sin el beneficio de una interfaz de memoria virtual y, en consecuencia pueden requerir memoria que reside en páginas físicamente contiguas.
 - Las páginas asignadas a los procesos en modo usuario no tienen que estar necesariamente en memoria física contigua.

45

SOYD 2020 □ Gustavo C. Distel

Asignación de memoria del *kernel*

Buddy System (descomposición binaria)

- El *buddy system* asigna memoria de segmentos de tamaño fijo que consiste en páginas físicamente contiguas.
- La memoria se asigna desde este segmento utilizando un asignador de potencia de 2, que satisface las solicitudes en unidades de tamaño con una potencia de 2 (4 KB, 8 KB, 16 KB, etc.).
- Una solicitud en unidades de tamaño menor se redondea a la siguiente potencia más alta de 2. Por ej., una solicitud de 11 KB se satisface con un segmento de 16 KB.
- Considerando un tamaño de segmento de 256 KB si el *kernel* solicita 21 KB:
 - El segmento se divide en dos *buddies* (amigos), A_L y A_R de 128 KB c/u.
 - Uno de estos se divide en dos *buddies* de 64 KB: B_L y B_R .
 - La siguiente potencia más alta de 2 de 21 KB es 32 KB, por lo que B_L o B_R se divide nuevamente en dos *buddies* de 32 KB, C_L y C_R .
 - Uno de estos *buddies* se utiliza para satisfacer la solicitud de 21 KB (C_L es el segmento asignado).

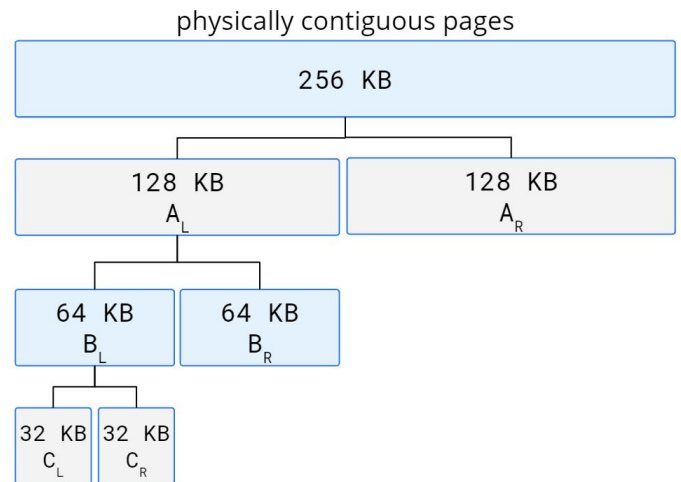
46

SOYD 2020 □ Gustavo C. Distel

Asignación de memoria del *kernel*

Buddy System (descomposición binaria)

- Una ventaja es la rapidez con la que los *buddies* adyacentes se pueden combinar para formar segmentos más grandes utilizando una técnica conocida como fusión (*coalescing*).
- Ej.: cuando el *kernel* libera C_L , se puede fusionar C_L y C_R en un segmento de **64 KB**.
- A su vez B_L puede fusionarse con su *buddy* B_R para formar un segmento de **128 KB**.
- Finalmente, podemos terminar con el segmento original de **256 KB**.
- El inconveniente del sistema de *buddies* es que redondear a la siguiente potencia más alta de 2 cause probablemente fragmentación dentro de los segmentos asignados.



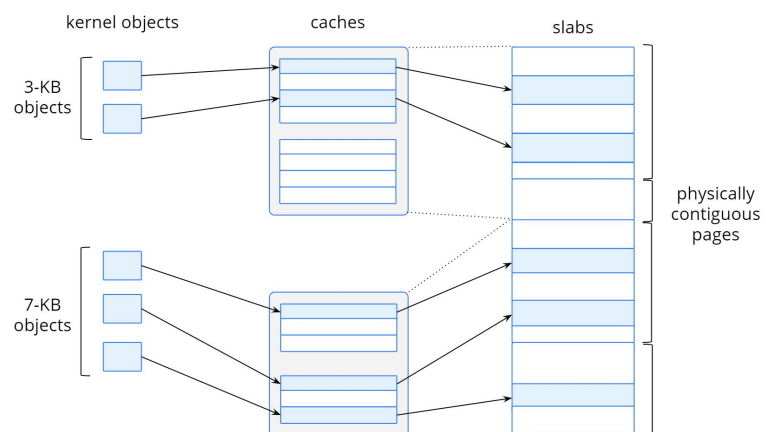
47

SOYD 2020 □ Gustavo C. Distel

Asignación de memoria del *kernel*

Slab Allocation

- Un *slab* está formado por una o más páginas físicamente contiguas.
- Un caché consta de uno o más *slabs*.
- Hay una memoria caché por cada estructura de datos del *kernel*; por ej., una para los descriptores de proceso, otra para objetos de archivo, otra para semáforos, etc.
- Cada caché se llena con objetos conformados por instancias de la estructura de datos del *kernel* que representa la caché.
- Por ej., la caché que representa semáforos almacena instancias de objetos semáforos, la caché que representa descriptores de proceso almacena instancias de objetos descriptores de proceso, y así sucesivamente.



48

SOYD 2020 □ Gustavo C. Distel

Asignación de memoria del *kernel*

Slab Allocation

- El algoritmo de asignación de *slabs* utiliza cachés para almacenar objetos del *kernel*.
- Cuando se crea una memoria caché, se le asigna una serie de objetos, que inicialmente se marcan como libres.
- El número de objetos en la caché depende del tamaño de la *slab* asociada.
- Por ejemplo, una *slab* de **12 KB** (compuesta por **3** páginas contiguas de **4 KB**) podría almacenar **6** objetos de **2 KB**.
- Cuando se necesita un nuevo objeto para una estructura de datos del *kernel*, el asignador puede seleccionar cualquier objeto libre del caché para satisfacer la solicitud.
- El objeto asignado desde el caché se marca como usado.

49

SOYD 2020 □ Gustavo C. Distel

Otras consideraciones

- *Prepaging*
- *Page Size*
- *TLB Reach*
- *Inverted Page Tables*
- *Program Structure*
- *I/O Interlock and Page Locking*

50

SOYD 2020 □ Gustavo C. Distel

Ejemplos de sistemas operativos

- Linux
- Windows
- Solaris