Sistemas Operativos

Memoria virtual

Curso 2024

Facultad de Ingeniería, Universidad de la República, Uruguay

Agenda

1. Introducción
2. Memoria virtual: implementación
3. Análisis de desempeño
4. Reemplazo de páginas
5. Asignación de frames
6. Hiperpaginación

Introducción

Memoria virtual

La técnica de memoria virtual está motivada por la necesidad de ejecutar simultáneamente múltiples programas que demandan más memoria de la disponible en el sistema.

La idea es mantener en memoria física solamente la memoria que el proceso está utilizando. El resto se almacena en disco.

Implica una separación de los espacios de direccionamiento de usuario (memoria lógica) de la memoria física.

El programador se despreocupa de las limitaciones de memoria que impone el sistema.

Memoria virtual

La idea es mantener en memoria física solamente la memoria que el proceso está utilizando. El resto se almacena en disco.

En general no es necesario cargar en memoria:

- Códigos para manejar condiciones de error inusuales.
- Arreglos, matrices, listas y tablas para las que se solicita más memoria de la que realmente necesitan/usan.
- Opciones y funcionalidades de los programas que se usan con probabilidad baja.

Memoria virtual

La técnica de memoria virtual abstrae la memoria principal en un gran arreglo uniforme de bytes.

Aporta varias ventajas:

- Los programas no están limitados por la cantidad de memoria física disponible.
- Ocupa menos memoria física y permite ejecutar más programas simultáneamente (mejora el uso y rendimiento de la CPU, sin empeorar los tiempos de respuesta).
- Se requiere menos E/S para cargar o intercambiar programas de usuario en la memoria (mejora la velocidad de ejecución).

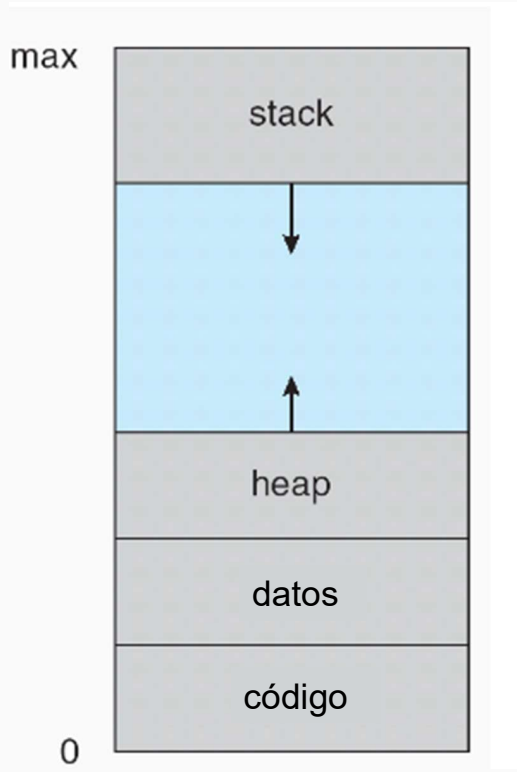
Si bien es una técnica muy potente, su uso descuidado puede generar una degradación importante del desempeño del sistema.

Memoria virtual

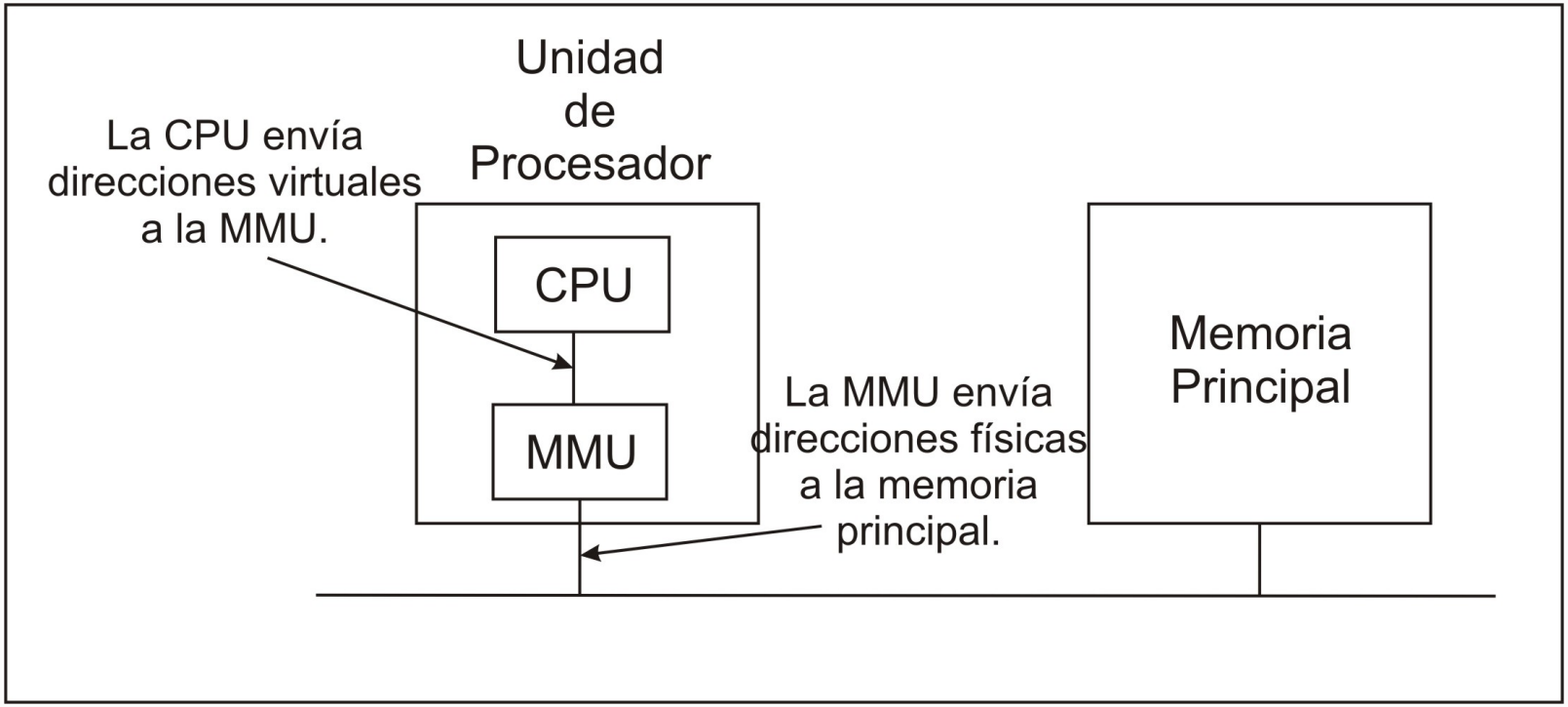
Cada proceso tiene un espacio de direccionamiento virtual (*virtual address space*).

Las direcciones de memoria que genera el proceso son absolutas en el espacio de direccionamiento.

La unidad de administración de memoria (MMU) se encarga de realizar la traducción de direcciones virtuales a físicas.

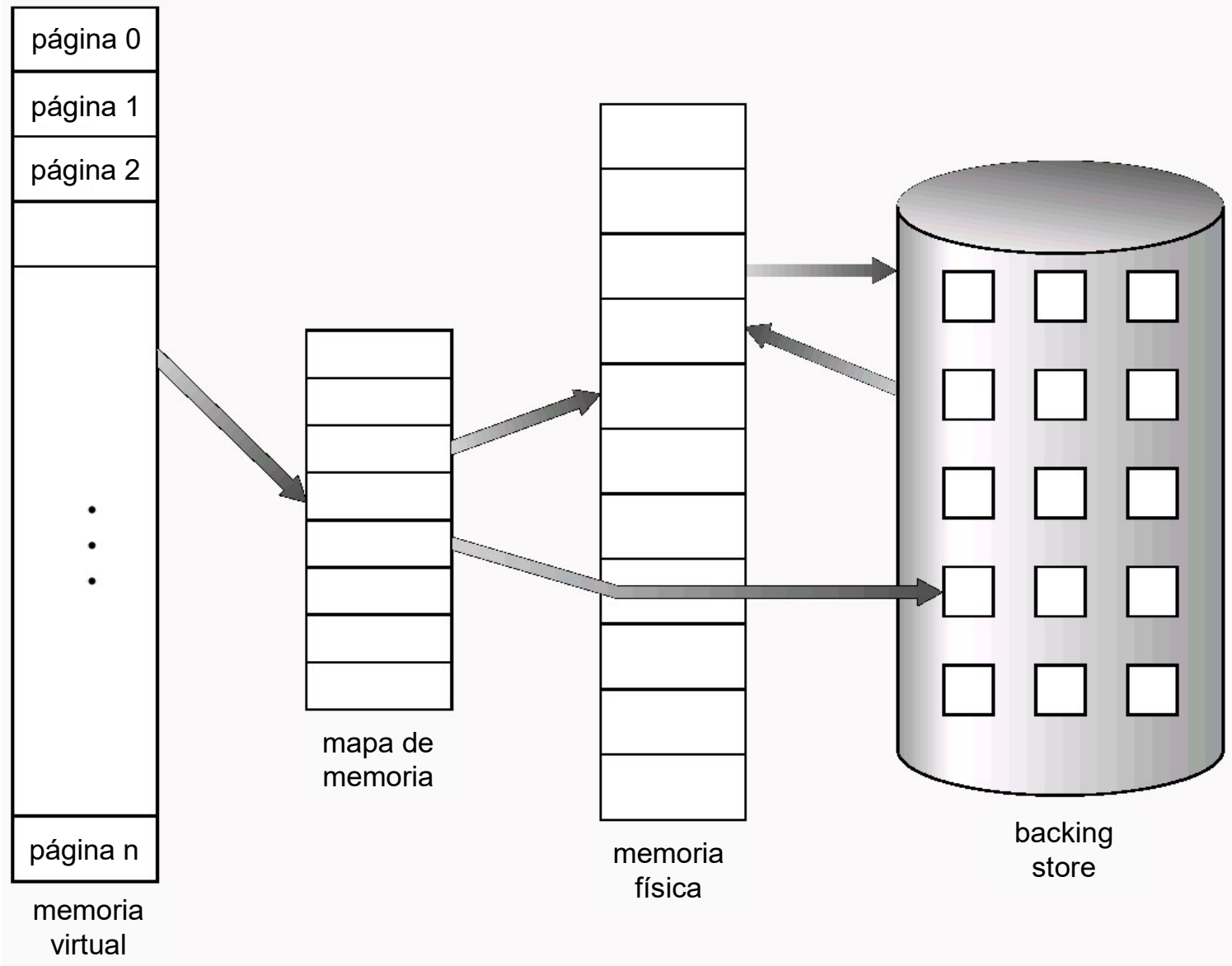


Memoria virtual



Memoria virtual: implementación

Esquema de la implementación de memoria virtual



Memoria virtual: implementación

La memoria virtual se implementa mediante la técnica de paginación bajo demanda.

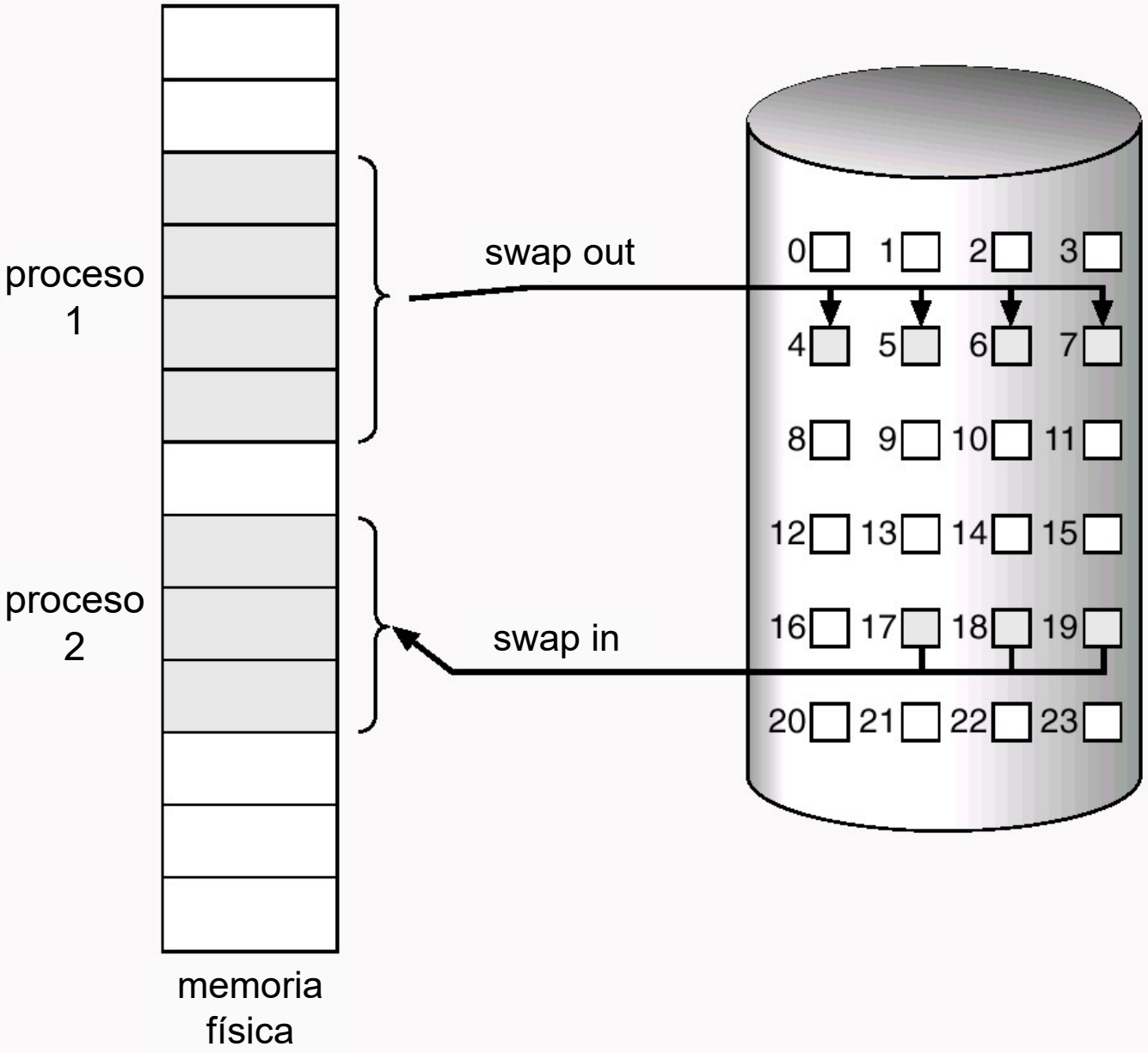
Se aplica un esquema de segmentación, con los segmentos divididos en páginas.

Los procesos residen en disco y se les asigna memoria principal cuando se cargan para ejecutar.

No se carga todo el proceso en memoria. Se implementa un cargador perezoso (lazy swapper), que carga las páginas a medida que se van necesitando.

Se utiliza un espacio de swap, que permite mantener en el sistema más memoria utilizada de la física disponible.

Memoria virtual: implementación



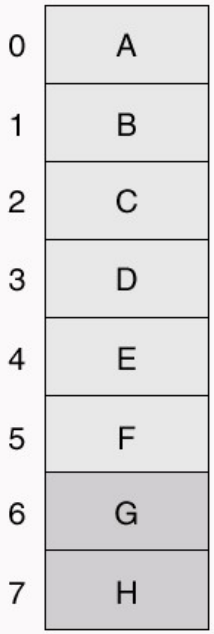
Memoria virtual: implementación

La paginación bajo demanda requiere conocer las páginas que están activas en memoria: se utiliza el valid-invalid bit.

La tabla de páginas indica qué páginas están cargadas en memoria principal.

Cada entrada en la tabla de páginas incluye el número de frame y el valid-invalid bit.

Memoria virtual: implementación



memoria virtual

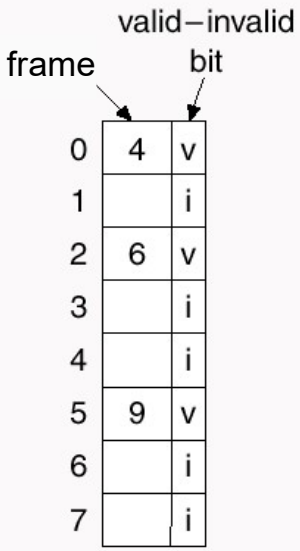
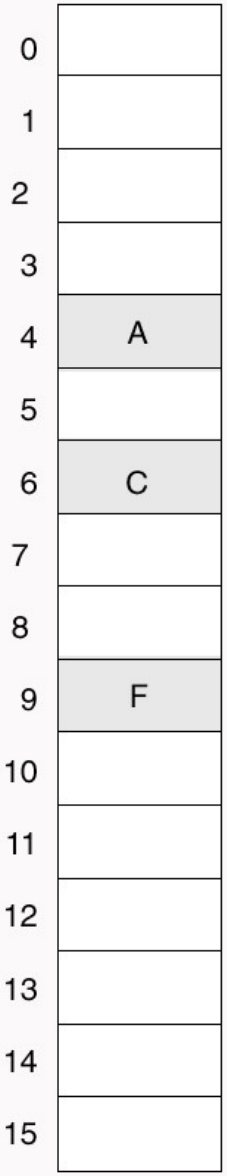
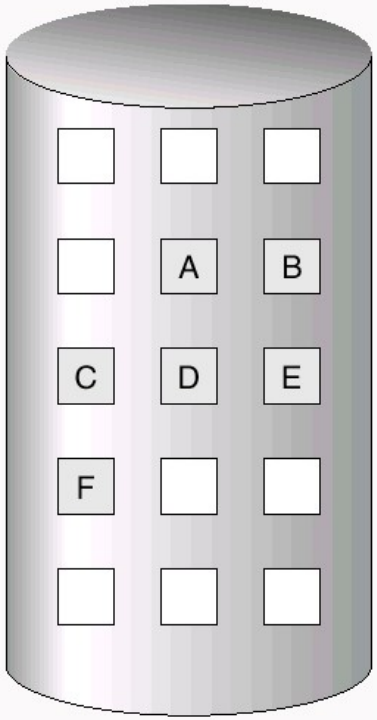


tabla de páginas



memoria física



Memoria virtual: implementación

A la porción de memoria de un proceso que está cargada en memoria principal se le denomina memoria residente.

El acceso a memoria residente por parte de un proceso es un acceso normal, pero el acceso a memoria no residente genera un fallo de página (page fault).

El fallo de página genera una interrupción que activa una rutina de atención del sistema operativo, para cargar la página en memoria principal.

Memoria virtual: acceso a memoria

Paso 1: Verificar que el proceso referencia una página válida

El proceso puede generar cualquier dirección en su espacio de direccionamiento (virtual), pero no todas las direcciones son válidas. Una referencia a una posición no asignada puede intentar acceder a una página no válida para al proceso.

Para verificar se usa una tabla interna del proceso o el campo de memoria límite en el PCB. Si la dirección no es válida, se genera un segmentation fault y el sistema operativo finaliza el proceso.

Si el acceso fue correcto en el espacio virtual, se busca en la tabla de páginas el frame correspondiente, verificando el bit de validez. Si es válido, se traduce la dirección y se accede al frame en la memoria principal.

Memoria virtual: acceso a memoria

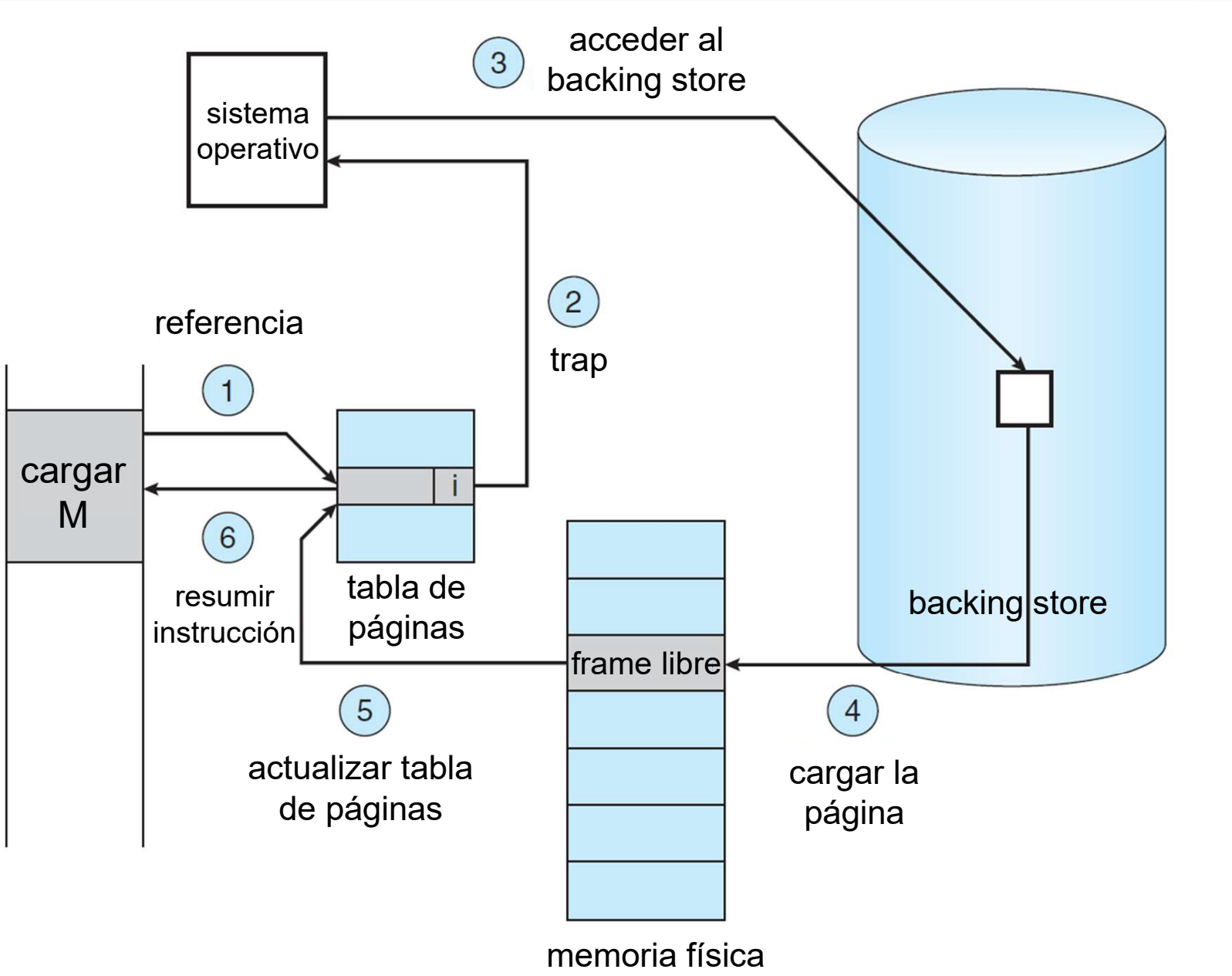
Paso 2: Fallo de página

Si el bit de validez indica que la página no se encuentra en memoria, se genera un fallo de página.

Se invoca a una rutina de atención del fallo de página:

3. Busca un frame libre en memoria principal. Si no hay un frame libre, ejecuta un algoritmo de reemplazo.
4. Se realiza un operación de lectura sobre el disco que guarda la página y se carga en el frame libre.
5. Se actualiza la tabla de páginas y la información en el PCB del proceso para marcar que la página está disponible en memoria principal.
6. Se retorna el control a la instrucción que fue interrumpida debido al fallo de página.

Memoria virtual: acceso a memoria



Esquemas de paginación a demanda

Paginación a demanda pura: nunca se carga en memoria una página hasta que es requerida.

El proceso inicia su ejecución sin páginas en la memoria.
En la primera instrucción del proceso se produce un page fault.
Después de traer esa página a la memoria, el proceso continúa ejecutándose, fallando si es necesario, hasta que cada página que necesita esté en la memoria. Luego ejecuta sin fallos.

Esquemas de paginación a demanda

Un proceso puede acceder a varias páginas de memoria al ejecutar una instrucción (una página para la instrucción y muchas para los datos): es posible que ocurran múltiples fallos de página por instrucción, reduciendo el rendimiento del sistema. En general no sucede por la localidad de las referencias, y la paginación a demanda pura tiene un rendimiento razonable.

Paginación previa: se cargan otras páginas, además de la(s) requerida(s). El sistema operativo intenta predecir qué páginas requerirá el proceso y las carga de manera proactiva en la memoria principal.

Paginación a demanda: soporte

Soporte requerido para implementar la paginación a demanda:

- (lógico) Tabla de páginas, incluyendo marca de bit válido-inválido y bits de protección.
- (hardware) Backing store para el swap de páginas.

La paginación a demanda requiere resumir una instrucción después de un fallo de página. Se guarda el estado (registros, flags, contador de instrucciones, etc.) del proceso interrumpido cuando ocurre el fallo, para resumir el proceso exactamente en el mismo lugar y estado, con la página accesible en memoria.

Si el fallo se da al buscar una instrucción, se resume recuperando la instrucción. Si el fallo se da al buscar un operando, se busca y decodifica la instrucción y luego se busca el operando.

Análisis de desempeño

Desempeño de la técnica de paginación a demanda

La paginación a demanda puede generar un alto impacto en el rendimiento del sistema.

La métrica utilizada para determinar su efecto es el tiempo de acceso efectivo (Effective Access Time, EAT):

$$EAT = ((1 - p) \times \text{ma}) + (p \times \text{TRPF})$$

p es la probabilidad de que ocurra un fallo de página, TRPF es el tiempo requerido para resolver un fallo y ma el tiempo del acceso a memoria.

Desempeño de la técnica de paginación a demanda

El tiempo requerido para resolver un fallo de página incluye el tiempo para las siguientes tareas:

- generar la interrupción para el sistema operativo
- determinar que la interrupción es de un fallo de página
- salvar el estado del proceso
- ejecutar el planificador (aprovechar la CPU mientras espera)
- solicitar la lectura de la página a disco
- esperar la transferencia desde disco
- atender la notificación de completitud de la operación de E/S
- salvar el estado de otro (posible) proceso en ejecución
- corregir la tabla de páginas
- esperar que se asigne la CPU nuevamente
- restaurar el estado al proceso.

Desempeño de la técnica de paginación a demanda

Los componentes principales de la demora del fallo de página son:

1. Dar servicio a la interrupción de fallo de página
2. Leer la página
3. Reanudar el proceso

Codificando apropiadamente, las tareas 1 y 3 se pueden reducir a varios cientos de instrucciones (100 μs cada tarea). Pero cambiar la página tarda unos 8 ms, por el acceso al disco (latencia $\sim$ 3 ms, búsqueda $\sim$ 5 ms y transferencia $\sim$ 0,1 ms).

El tiempo total de paginación es del orden de 8 ms.
Cuando una cola de procesos está esperando el dispositivo, debe agregarse el tiempo de espera del dispositivo.

Ejemplo de análisis de desempeño

Sea TRPF = 8 ms y ma = 200 ns, se tiene:

$$EAT = ((1 - p) \times 200) + (p \times (8{,}000{,}000))$$

$$EAT = 200 + (7{,}999{,}800 \times p)$$

El tiempo de acceso efectivo es directamente proporcional a la probabilidad de fallo de página.

Si un acceso cada 1.000 provoca un fallo de página, EAT es 8,2 μs. El sistema se enlentece en un factor de 40 !

Si se permite un 10 % de degradación del sistema:

$$220 > 200 + (7{,}999{,}800 \times p)$$

$$20 > 7{,}999{,}800 \times p$$

$$p < 0,0000025$$

Debe darse un fallo cada 400.000 accesos.

Reemplazo de páginas

Reemplazo de páginas

En un sistema cargado, el uso de la memoria principal por parte de los procesos puede agotar el recurso.

Cuando se necesita traer a memoria principal una página y no hay un frame libre, se debe buscar un frame a reemplazar (frame víctima).

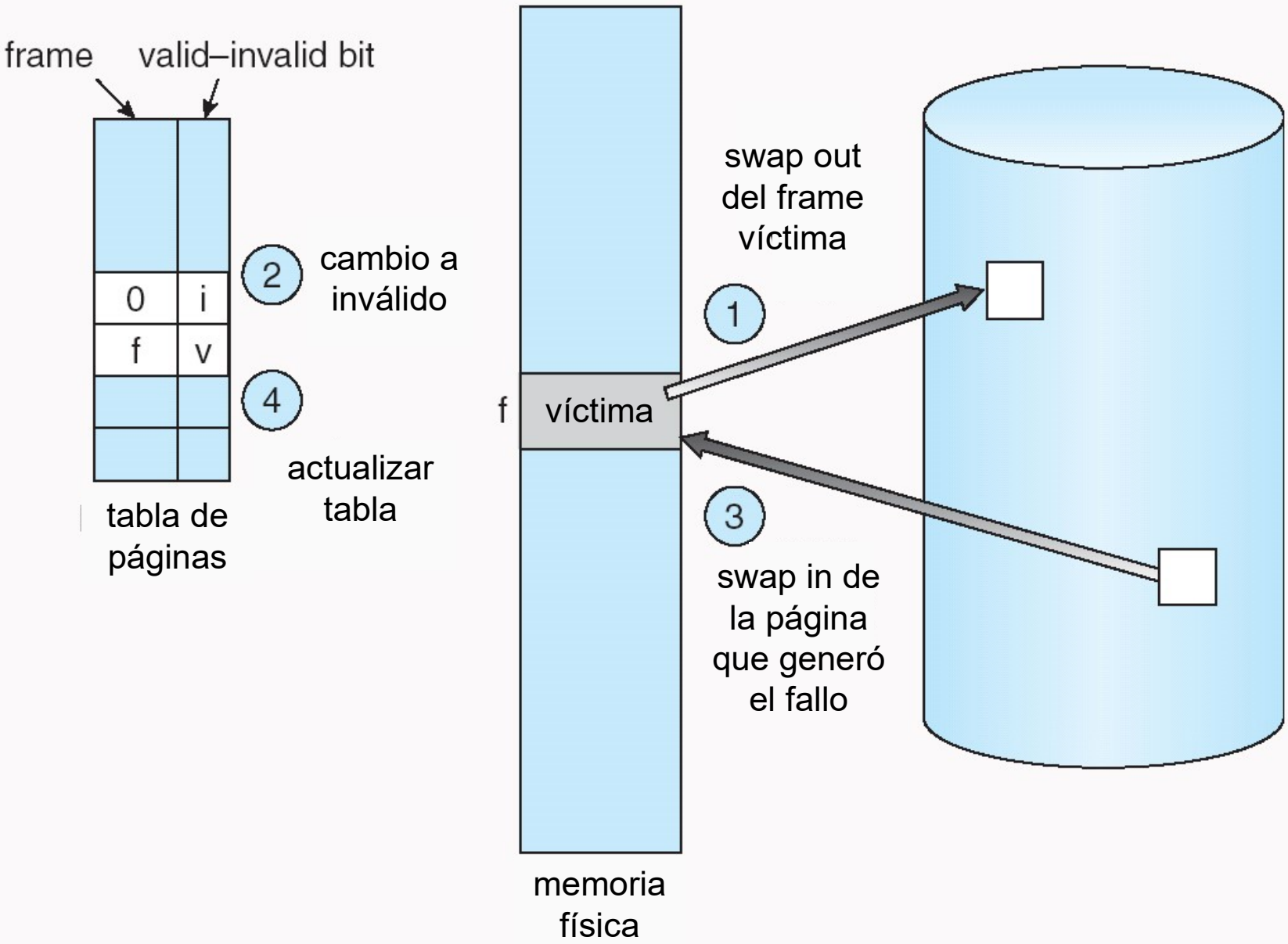
Se escribe el contenido del frame seleccionado en el espacio de swap, se actualiza la tabla de páginas y se utiliza el frame liberado para almacenar la página que generó el fallo.

Reemplazo de páginas

La rutina que atiende el fallo debe incorporar el reemplazo:

1. Elegir el frame victima, aplicando un algoritmo de reemplazo.
2. Escribir el contenido del frame victima en el swap y ajustar la tabla de página correspondiente (swap out).
3. Cargar la página que generó el fallo en el frame liberado.
4. Actualizar la tabla de páginas del proceso.

Reemplazo de páginas



Algoritmo de reemplazo FIFO (First In – First Out)

Reemplazar la página con mayor tiempo de permanencia en memoria principal

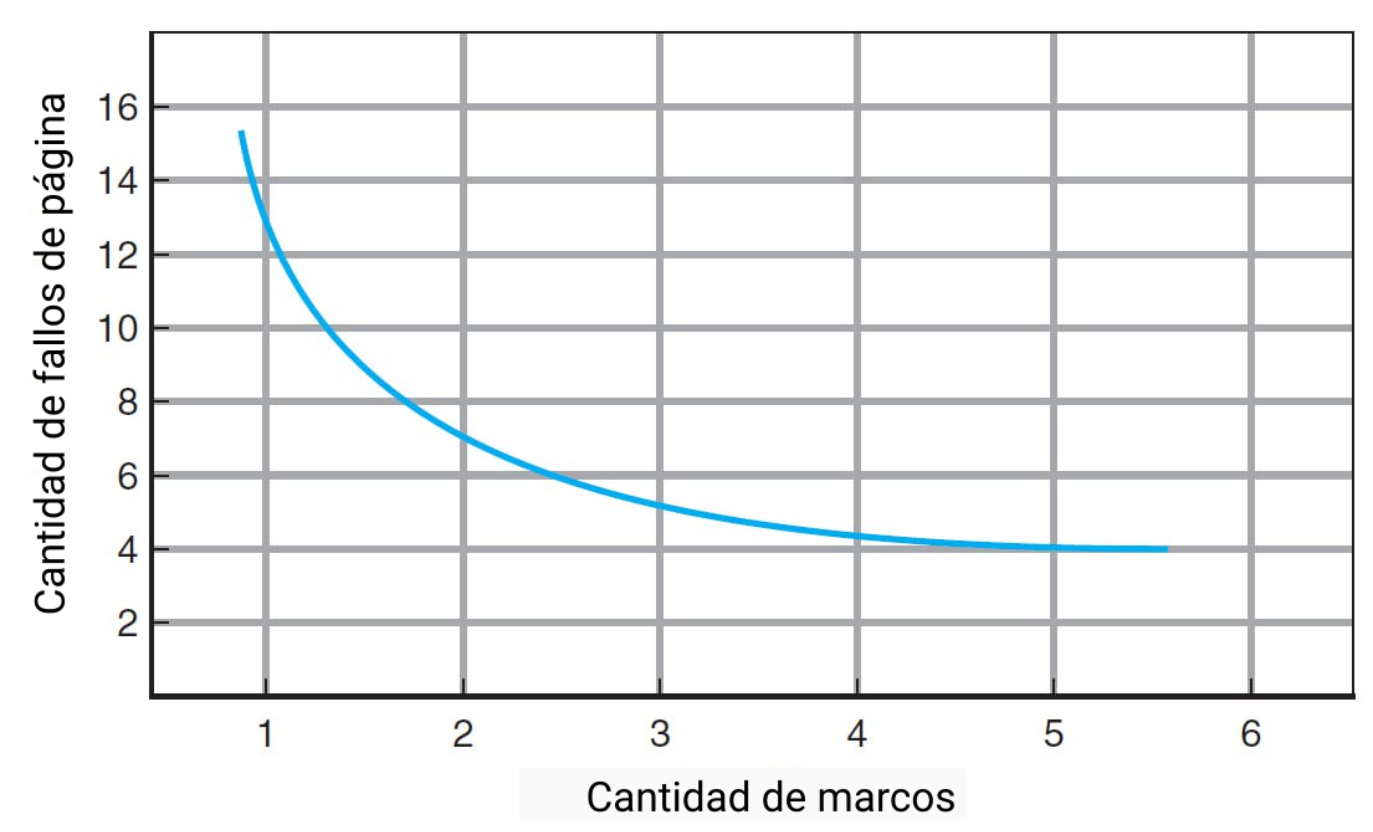
Su implementación es simple: requiere únicamente de una estructura tipo cola.

El algoritmo FIFO sufre de anomalía de Belady: Los fallos de página se incrementan al asignar más frames.

Fallos de página y número de frames

Agregar memoria física aumenta la cantidad de frames.

El resultado esperado es una curva descendente: el número de fallos de página cae cuando aumenta el número de frames.



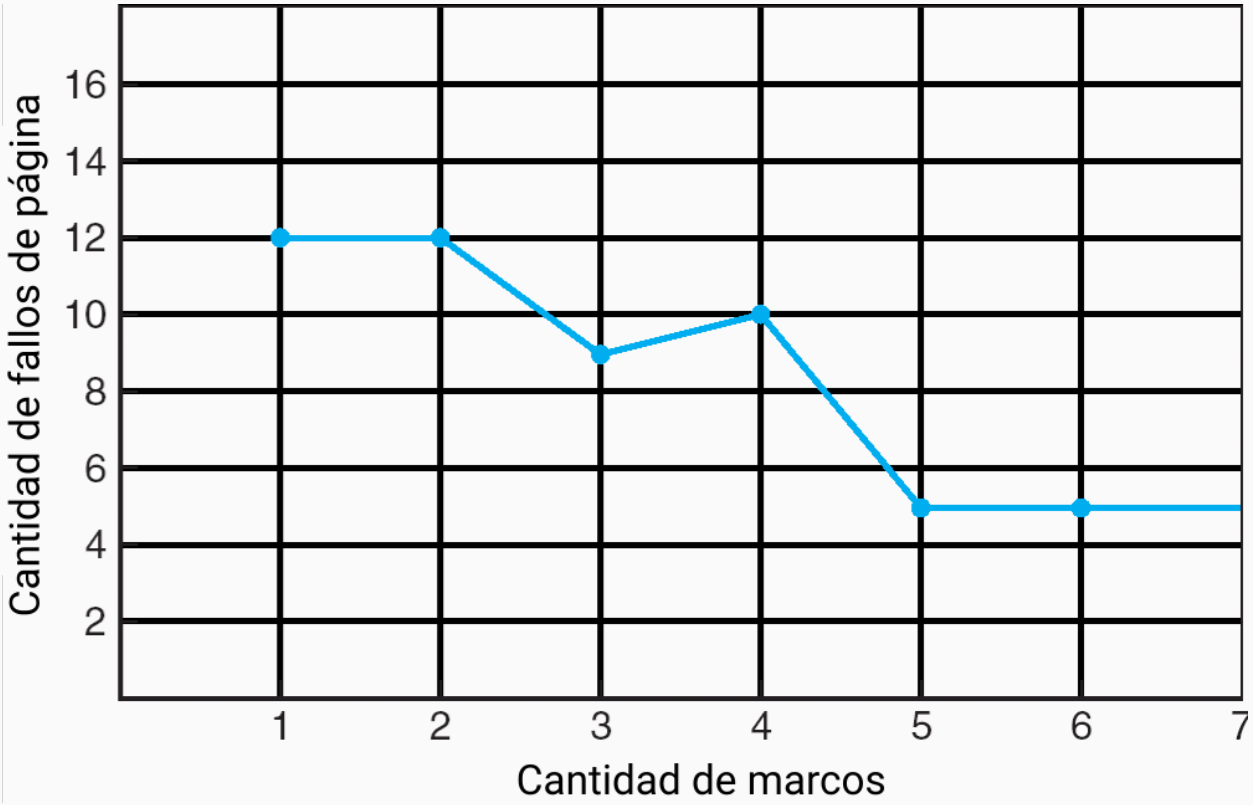
Anomalía de Belady para reemplazo FIFO

Sea la siguientes secuencia de acceso a página:

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

En 3 frames ocurren 9 fallos de página.

En 4 frames ocurren 10 fallos de página.



Algoritmo de reemplazo óptimo

El algoritmo con la tasa de fallos de página más baja (de todos los algoritmos posibles). Nunca sufre la anomalía de Belady.

Reemplazar la página que no se utilizará durante más tiempo

No es posible implementarlo en la práctica, ya que requiere conocimiento del futuro (saber de antemano a que páginas accederá un proceso).

El algoritmo óptimo se usa para análisis comparativos.
Por ejemplo, es útil saber que un algoritmo (no óptimo) está a 12 % del óptimo en el peor caso y 5 % en promedio.

Algoritmo de reemplazo segunda oportunidad (second chance)

En el algoritmo FIFO una página se reemplaza sin tener en cuenta las referencias que tuvo.

Una alternativa es incluir una marca para cada página que se accede (bit de referencia), luego de ser cargada en memoria.

Second chance: verifica el bit de referencia del frame víctima.
Si el bit de referencia es 1, se le da una segunda oportunidad a la página, se la mueve al final de la cola y se resetea su bit de referencia. Se continua con la siguiente página en la cola. Si el bit de referencia es 0, la página se selecciona para ser reemplazada.

Su implementación es menos eficiente que FIFO, pero permite reducir la cantidad de fallos de página.

Algoritmo de reemplazo No Recientemente Usada (Not Recently Used, NRU)

Algoritmo de segunda oportunidad mejorado: se asocia a cada página un bit de referencia y uno de modificación.

El bit de referencia se actualiza cada vez que se opera sobre la página (lectura o escritura). El bit de modificación se activa cada vez que se escribe.

Cada cierto tiempo, se genera una interrupción para limpiar el bit de referencia.

Las páginas se dividen en cuatro clases:

- Clase 0: no referenciada, no modificada.
- Clase 1: no referenciada, modificada.
- Clase 2: referenciada, no modificada.
- Clase 3: referenciada, modificada.

Algoritmo de reemplazo No Recientemente Usada (Not Recently Used, NRU)

Se reemplaza una página de la clase más baja que no sea vacía.

Es un algoritmo sencillo de implementar.

Para ser eficiente debe tener soporte de hardware, ya que en cada referencia a memoria se deben actualizar los bits de referencia y de modificación.

Algoritmo de reemplazo Menos Recientemente Usada (Least Recently Used, LRU)

Uno de los algoritmos que más se aproxima al óptimo.

A cada página se le asocia el tiempo en que fue referenciada.
La página reemplazada es la que fue accedida hace más tiempo.

Es óptimo respecto al uso hacia atrás de las páginas y es muy utilizado por los sistemas operativos actuales.

Para su implementación requiere soporte a nivel de hardware.

No sufre la anomalía de Belady.

Implementación de LRU

Cómo ordenar los frames de acuerdo a su tiempo de último uso.

Contador lógico. Agrega a la tabla de páginas el campo tiempo de uso, que se actualiza en cada acceso. Se debe buscar en la tabla de páginas para encontrar la página LRU y escribir en la memoria (tiempo de uso) para cada acceso a la memoria. Los tiempos deben mantenerse al cambiar la tabla de páginas.

Pila de números de página. Cada vez que se accede a una página, se coloca en el tope. La página LRU queda en la parte inferior de la pila. Para eliminar páginas intermedias se implementa con una lista doblemente enlazada, con punteros de tope y de cola. Eliminar una página y colocarla en el tope requiere cambiar seis punteros en el peor caso. La actualización es más cara, pero no se requiere búsqueda: el puntero de cola indica la página LRU.

Asignación de frames

Asignación de frames

Los procesos en ejecución concurrente realizan sus pedidos de memoria y el sistema operativo trata de resolver las solicitudes de forma de lograr un uso adecuado del recurso.

Si el sistema operativo no aplica una estrategia de asignación, un proceso que requiera mucha memoria puede colapsar el sistema.

Es necesario aplicar estrategias de asignación adecuadas para todos los procesos que comparten la memoria del sistema.

Estrategia de asignación de frames

Asignación equitativa. Dividir la cantidad de frames entre los procesos en ejecución.

La estrategia no es eficiente, ya que los procesos no tienen las mismas necesidades de memoria.

Asignación proporcional. Pondera la cantidad de frames asignados según el uso de memoria.

Sea S_i la cantidad de memoria virtual del proceso p_i.
La memoria virtual de todos el sistema es $S = \sum S_i$.

La estrategia de asignación ponderada asigna al proceso p_i $f_i = m \times S_i/S$ frames, siendo m la cantidad de frames del sistema.

Asignación de frames: estrategias locales y globales

Los algoritmos de reemplazo de páginas se clasifican en:

- Reemplazo global: un proceso reemplaza un frame cualquiera, inclusive si es utilizado por otro proceso.
- Reemplazo local: un proceso reemplaza únicamente los frames que tiene asignado.

En el reemplazo local, la cantidad de frames asignados a un proceso no varía. En el reemplazo global si.

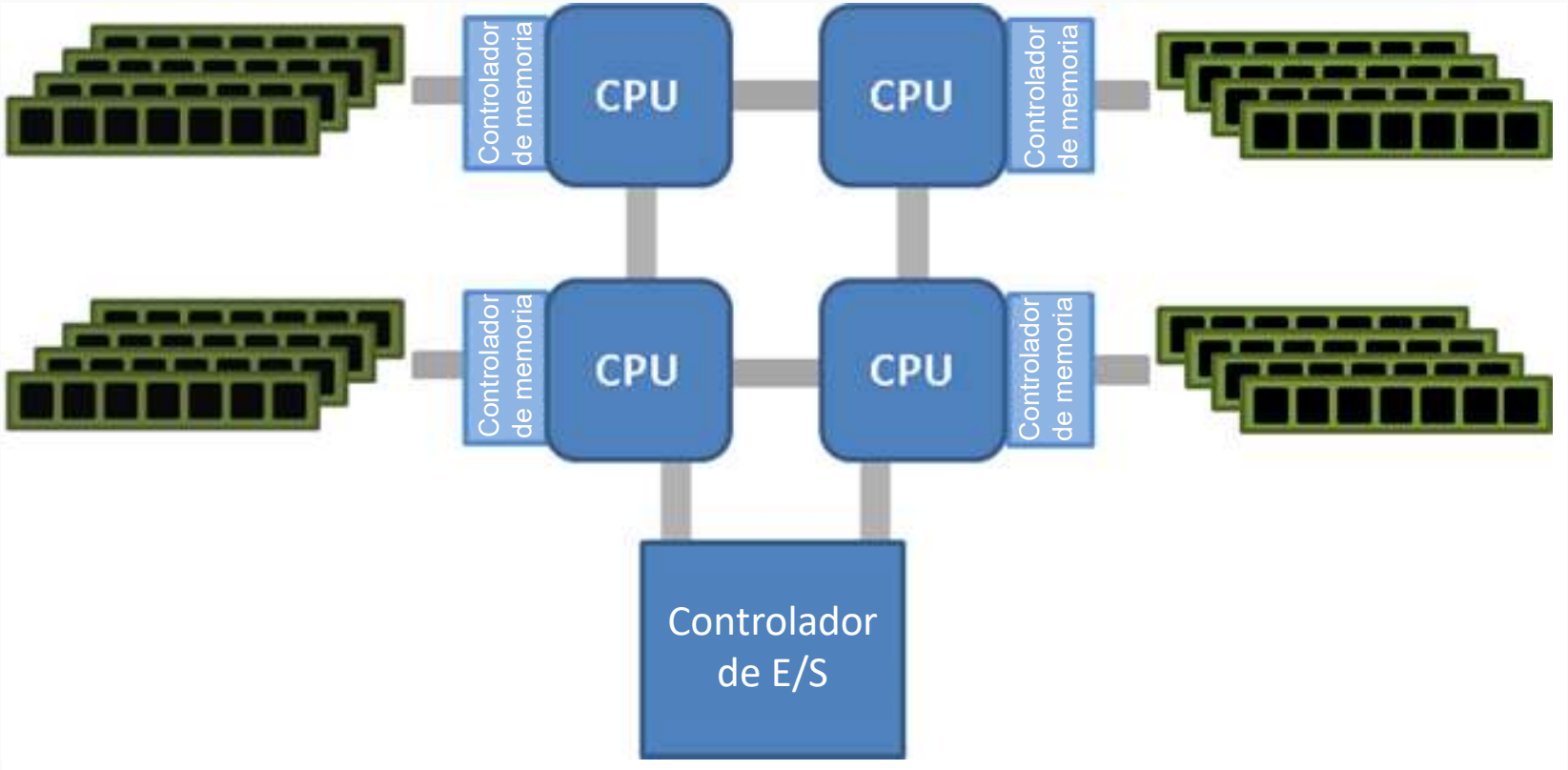
Con reemplazo global, los fallos de página de un proceso afectan la eficiencia de ejecución de otros.

Con reemplazo local, se ocupan frames que pueden ser poco usados y no son reemplazados.

El reemplazo global es la estrategia más utilizada.

Acceso no uniforme a memoria

Sistemas non-uniform memory access (NUMA): los tiempos de acceso a la memoria varían significativamente, porque memorias y CPUs están en diferentes placas base.



Acceso no uniforme a memoria

Se aplican estrategias para asignar frames “lo más cerca posible” (con latencia mínima) de la CPU en la que ejecuta el proceso.

El planificador almacena la última CPU en la que se ejecutó cada proceso. Se puede programar cada proceso “en su CPU anterior” y el sistema puede asignar frames cercanos, para obtener mejor uso de caché y menores tiempos de acceso a la memoria.

Para gestionar hilos se necesitan estructuras avanzadas.
Ejemplo: lgroups (grupos de latencia) en el kernel de Solaris, que reúnen CPU y memoria cercanas. Se intenta ejecutar todos los threads y asignar toda la memoria de un proceso dentro de un lgroup, o en lgroups cercanos. Minimiza la latencia de la memoria y maximiza el hit rate de la caché de la CPU.

Hiperpaginación

Hiperpaginación

La estrategia de asignación de frames limita a los procesos la cantidad de frames que pueden tener en memoria.

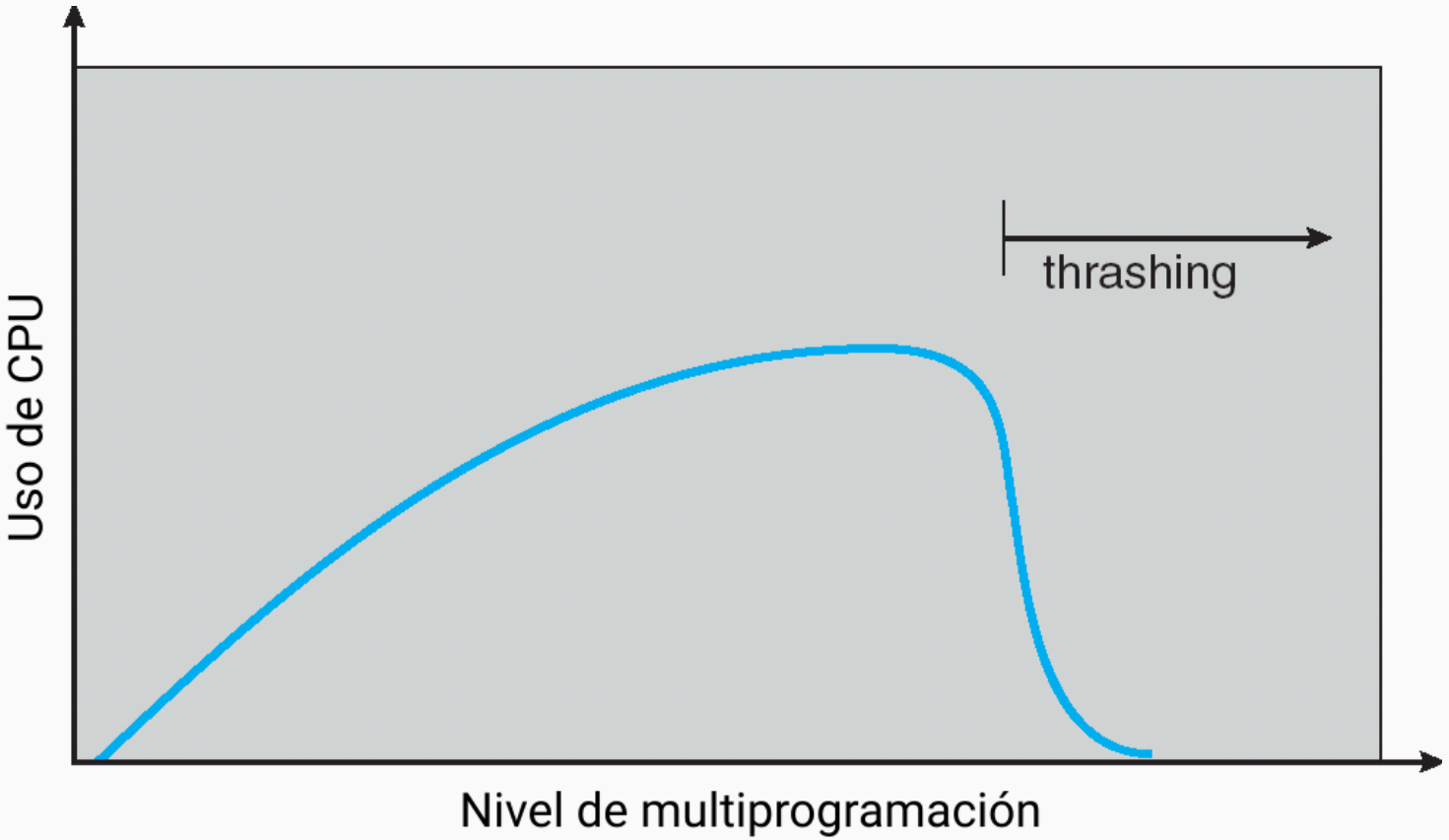
Si un proceso tiene un uso activo de un mayor número de frames que los asignados por el sistema, tendrá muchos fallos de página. Si todas sus páginas están en uso activo, reemplaza páginas que se necesitarán de inmediato.

Un proceso que usa más tiempo para paginar que para ejecutar sufre de hiperpaginación (thrashing).

La hiperpaginación degrada significativamente el rendimiento.

Hiperpaginación

La multiprogramación genera un mayor uso del procesador, pero puede llevar al sistema a un estado de degradación por la hiperpaginación.

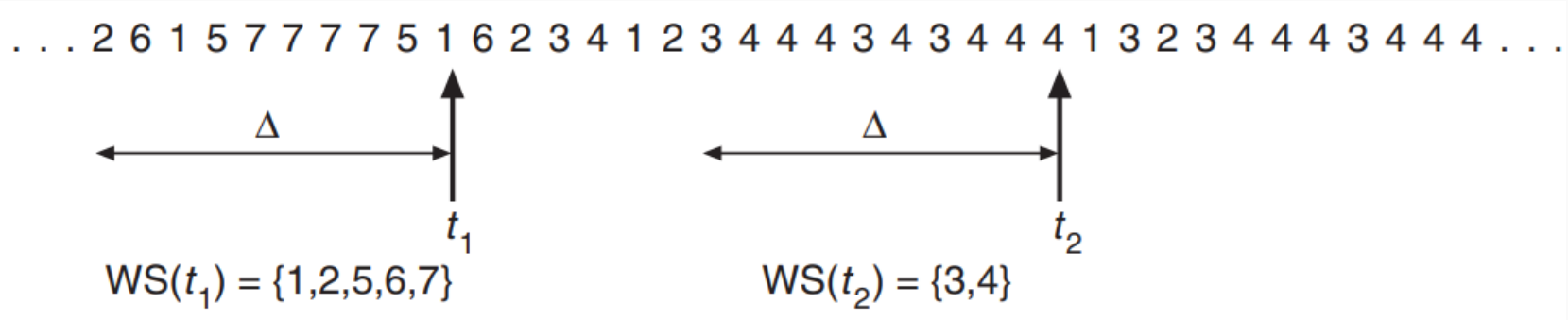


El modelo Working-Set

El modelo Working-Set (conjunto de trabajo) afronta el problema de la hiperpaginación basado en el principio de localidad.

A medida que un proceso ejecuta, utiliza un conjunto de páginas activas, que cambia a medida que ejecuta el proceso.

Se define el working-set.(WS) como el conjunto de páginas más recientemente referenciadas (en un tiempo Δ).



El modelo Working-Set

El working set representa apropiadamente la localidad.

Si Δ es demasiado pequeño, no abarcará toda la localidad; si es demasiado grande, puede superponer varias localidades

La cantidad total de frames demandada por los procesos del sistema es $D = \sum_i size(WS_i)$

Si D es mayor que la cantidad de frames disponibles en el sistema, se tendrá hiperpaginación.

El sistema operativo puede tomar medidas como suspender un proceso y mover sus frames al backing store.

El modelo Working-Set

Mantener el WS mediante las referencias que realiza un proceso es costoso (implica inserciones y bajas continuas, por cada referencia se debe verificar si la página ya está en el WS).

Una estrategia común es usar un timer que interrumpe cada tiempo fijo t, o cuando se realiza un número de referencias (menor a Δ).

Cuando interrumpe el timer, se verifican todas las páginas del sistema para armar los WS de los procesos. Se dispone del bit de referencia y Δ/t bits adicionales para cada verificación (reseteables).

En un fallo de página, se buscan páginas que no pertenecen a los WS recientes.

Estrategia de frecuencia de fallos de página

Estrategia alternativa al modelo de working-set

Monitorea la frecuencia con que se producen los fallos de pagina.

Si la frecuencia es mayor a un umbral U_h, el proceso necesita más frames.

Si la frecuencia es menor a un umbral U_l, el proceso puede liberar un frame.