Administración de memoria: Memoria virtual

Gunnar Wolf

Facultad de Ingeniería, UNAM
Instituto de Investigaciones Económicas, UNAM

Índice

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

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

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

Esquema general empleando memoria virtual

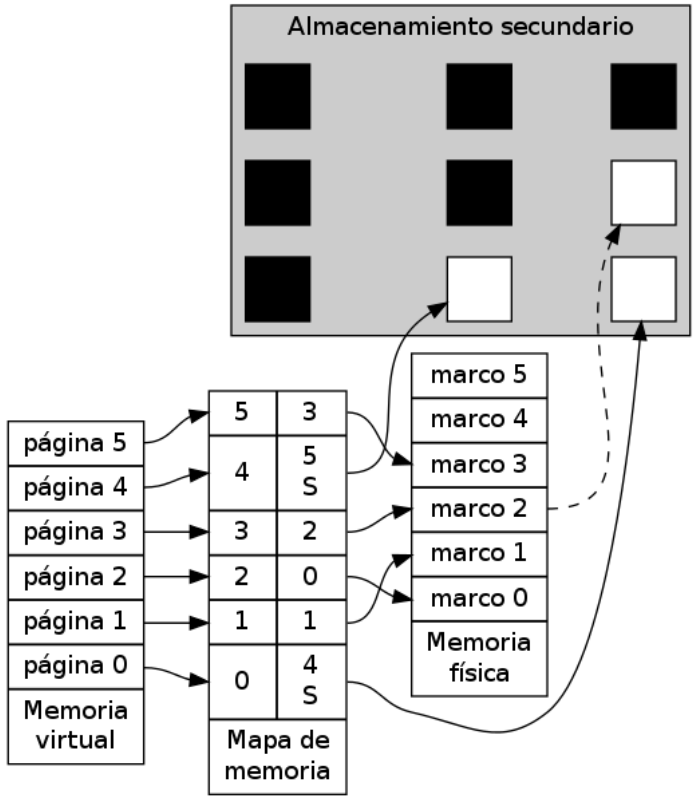

Figura: Esquema general de la memoria, incorporando espacio en almacenamiento secundario, representando la memoria virtual

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*

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

Transparencia al proceso

- Es importante recalcar que cuando hablamos de memoria virtual, ésta se mantiene *transparente al proceso*
- El proceso puede dedicarse a cumplir su tarea, el sistema operativo *paginará* la memoria según haga falta
- Es *posible* hacer ciertas indicaciones de preferencia, pero en general no es el caso

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

Índice

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

Deja dormir al *código durmiente*

- En el transcurso de la vida de un proceso, porciones importantes de su memoria se mantienen *durmientes* — Código que sólo se emplea eventualmente
  - Respuesta a situaciones de excepción
  - Exportación de un documento a determinado formato
  - Verificación de sanidad al cerrar el programa
  - Estructuras inicializadas con espacio para permitir que crezcan
  - ...
- Las páginas en que están dichos datos no son necesarias durante la ejecución normal
  - El paginador puede *posponer* su carga hasta cuando sean necesarias
  - Si es que alguna vez son requeridas

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

Entonces, ¿sobre demanda?

- Todo el código que ejecute o referencie directamente el procesador *tiene* que estar en memoria principal
  - Pero no tiene que estarlo *antes* de ser referenciado
  - Para ejecutar un proceso, sólo requerimos cargar la porción necesaria para *comenzar* la ejecución
- Podemos emplear a un paginador *flojo*
  - Sólo ir cargando a memoria las páginas conforme van a ser utilizadas
  - Las páginas que no sean requeridas nunca serán cargadas a memoria

¿Paginador *flojo*?

Flojo: Concepto usado en diversas áreas del cómputo

Flojo (*Lazy*) Busca hacer el trabajo mínimo en un principio, y diferir para más tarde tanto como sea posible

Ansioso (*Eager*) Busca realizar todo el trabajo que sea posible *desde un principio*

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

¿Cómo hacemos *flojo* al paginador?

- Estructura de MMU muy parecida a la del TLB
- La *tabla de páginas* incluirá un *bit de validez*
  - Indica si la página está presente o no en memoria
  - Si no está presente, causa un *fallo de página*

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

Respuesta a un fallo de página

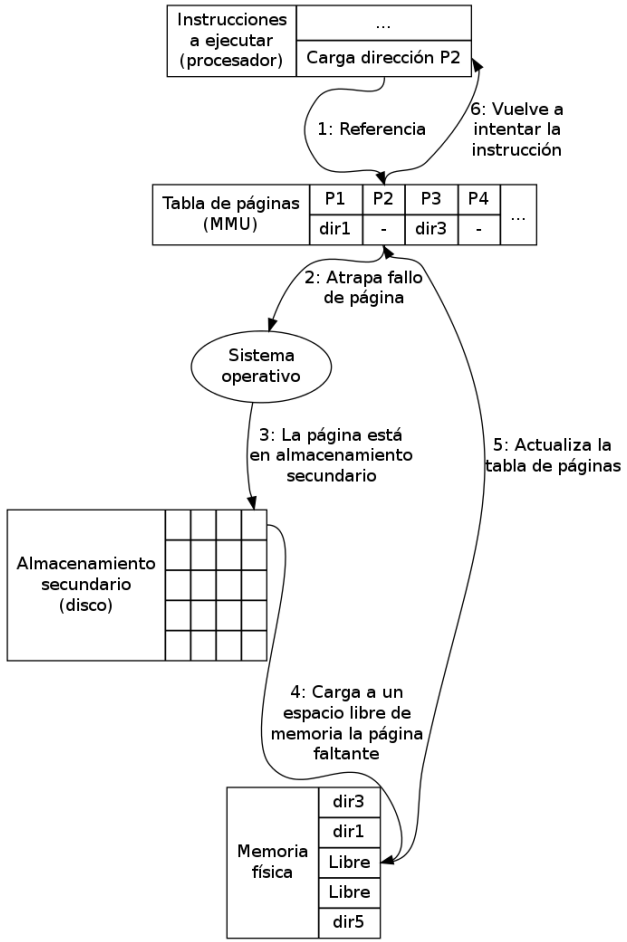

Figura: Pasos que atraviesa la respuesta a un fallo de página

Pasos para atender a un fallo de página

1. Verificar en PCB: ¿Esta página ya fue asignada al proceso? (¿es válida?)
2. Si no es válida, se termina el proceso
3. Buscar un marco disponible
   - P.ej. en una tabla de asignación de marcos
4. Solicita el al disco la lectura de la página hacia el marco especificado
   - Continúa ejecutando otros procesos
5. Cuando finaliza la lectura, actualiza PCB y TLB para indicar que la tabla está en memoria
6. Termina la suspensión del proceso.
   - Continúa con la instrucción que desencadenó el fallo.
   - El proceso continúa como si la página siempre hubiera estado en memoria

Paginación *puramente* sobre demanda

Llevar este proceso al extremo: Sistema de *paginación puramente sobre demanda* (*Pure demand paging*)

- Al iniciar la ejecución de un proceso, lo hace *sin ninguna página en memoria*
  - El registro de siguiente instrucción apunta a una dirección que no ha sido cargada
- De inmediato se produce un fallo de página
  - El sistema operativo responde cargando esta primer página
- Conforme avanza el flujo del programa, el proceso va ocupando el espacio real que empleará

Efecto de la paginación sobre demanda

- Al no cargarse todo el espacio de un proceso, puede iniciar su ejecución más rápido
- Al no requerir tener en la memoria física a los procesos completos, puede haber más procesos en memoria de los que cabrían antes
  - Aumentando el *grado de multiprogramación*

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

Midiendo el impacto en la ejecución

- El impacto en la ejecución de un proceso puede ser muy grande
- Un acceso a disco es varios miles de veces más lento que un acceso a memoria
- Podemos calcular el tiempo de acceso efectivo (t_e) a partir de la probabilidad de que en un proceso se presente un fallo de página ($0 \leq p \leq 1$)
- Conociendo el tiempo de acceso a memoria (t_a) y el tiempo que toma atender a un fallo de página (t_f):

$$t_e = (1 - p)t_a + pt_f$$

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

Resolviendo con valores actuales

- t_a ronda entre los 10 y 200ns
- t_f está cerca de los 8ms
  - Latencia del disco duro: 3ms
  - Tiempo de posicionamiento de cabeza: 5ms
  - Tiempo de transferencia: 0.05ms
- Si sólo uno de cada mil accesos a memoria ocasiona un fallo ($p = \frac{1}{1000}$):

$$t_e = (1 - \tfrac{1}{1000}) \times 200ns + \tfrac{1}{1000} \times 8,000,000ns$$
$$t_e = 199{,}8ns + 8000ns = 8199{,}8ns$$

Ahora sí: El impacto de la paginación sobre demanda

- Esto es, el tiempo efectivo de acceso a memoria es *40 veces* más lento que si no empleáramos paginación sobre demanda
- Podríamos mantener la penalización por degradación por debajo del 10 % del tiempo original
- Pero para que $t_e \leq 220$, tendríamos que reducir a $p \leq \frac{1}{399{,}990}$
- No olviden: No (necesariamente) es tiempo muerto
  - Multiprogramación: Mientras un proceso espera a que se resuelva su fallo de página, otros pueden continuar ejecutando

Ahora sí: El impacto de la paginación sobre demanda

- Esto es, el tiempo efectivo de acceso a memoria es *40 veces* más lento que si no empleáramos paginación sobre demanda
- Podríamos mantener la penalización por degradación por debajo del 10 % del tiempo original
- Pero para que $t_e \leq 220$, tendríamos que reducir a $p \leq \frac{1}{399{,}990}$
- No olviden: No (necesariamente) es tiempo muerto
  - Multiprogramación: Mientras un proceso espera a que se resuelva su fallo de página, otros pueden continuar ejecutando

Acomodo de las páginas en disco

- El cálculo presentado asume que el acomodo de las páginas en disco es óptimo
- Si hay que agregar el espacio que una página ocupa en un *sistema de archivos*, t_f fácilmente aumenta
  - Navegar estructuras de directorio
  - Posible fragmentación en espacio de archivos $\rightarrow$ la memoria va quedando esparcida por todo el disco
  - Mayores movimientos de la cabeza lectora
  - Problema prevalente en los sistemas tipo Windows
- Respuesta: *Partición de intercambio*, dedicada 100 % a la paginación
  - Mecanismo empleado por casi todos los sistemas Unix

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

Índice

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

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
- $\rightarrow$ Tenemos que llevar a cabo un *reemplazo de páginas*

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

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

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

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)

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

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

Y el reemplazo... ¿en dónde?

- Requerimos de un segundo parámetro
- Para analizar un algoritmo con una cadena de referencia, tenemos que saber *cuántos marcos* tiene nuestra computadora hipotética
  - Lo que buscamos es la *cantidad de fallos de página*
  - Depende directamente de los marcos disponibles
  - Y del tamaño (en páginas de memoria) de nuestro proceso

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

Casos límite respecto a los marcos disponibles

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

Datos base para los algoritmos

- A continuación veremos varios algoritmos de reemplazo de páginas
- Para el análisis, asumiremos una memoria con 3 marcos
- Y la siguiente cadena de referencia:

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

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

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

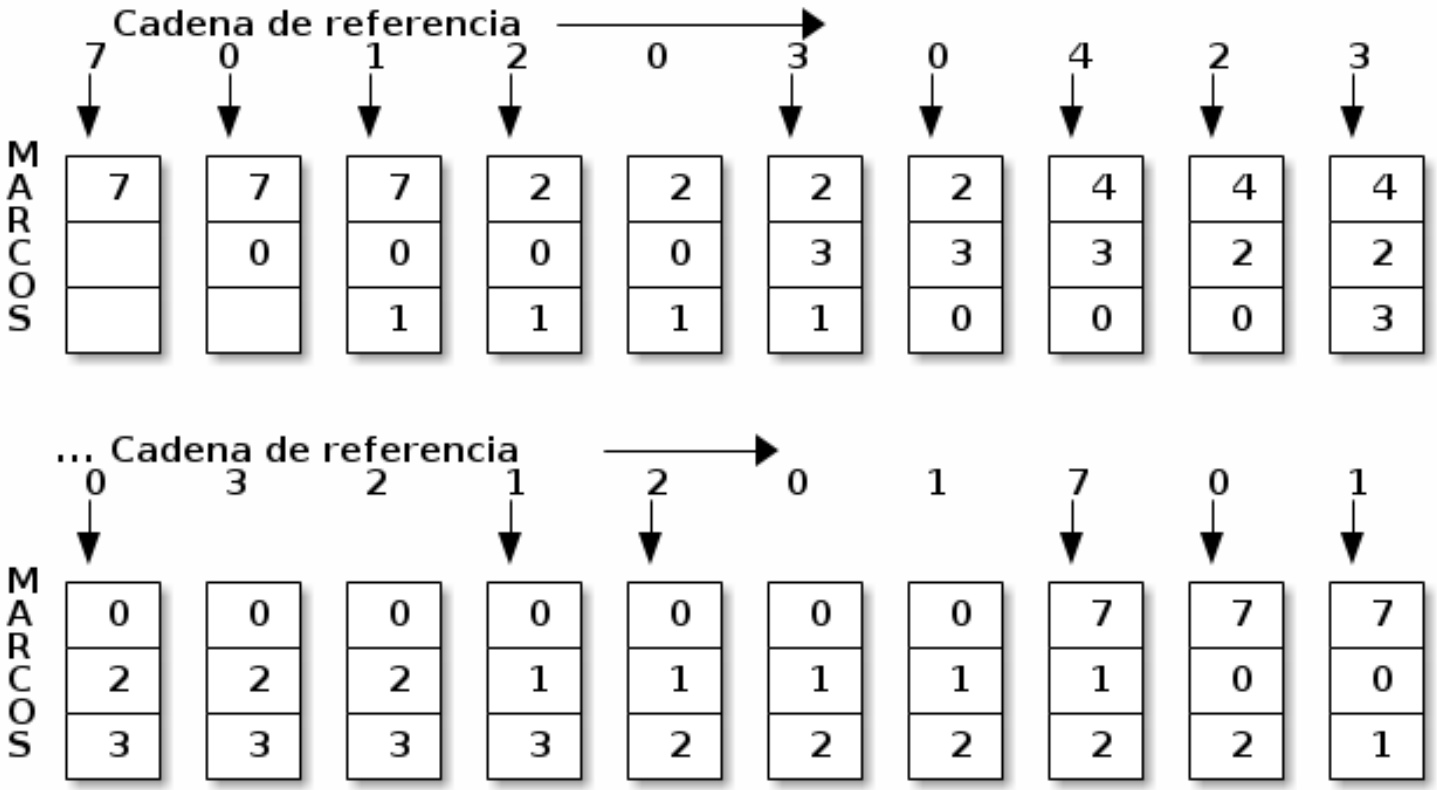

Figura: Algoritmo FIFO de reemplazo de páginas: 15 fallos

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

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

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

Anomalía de Belady: Expectativas de comportamiento

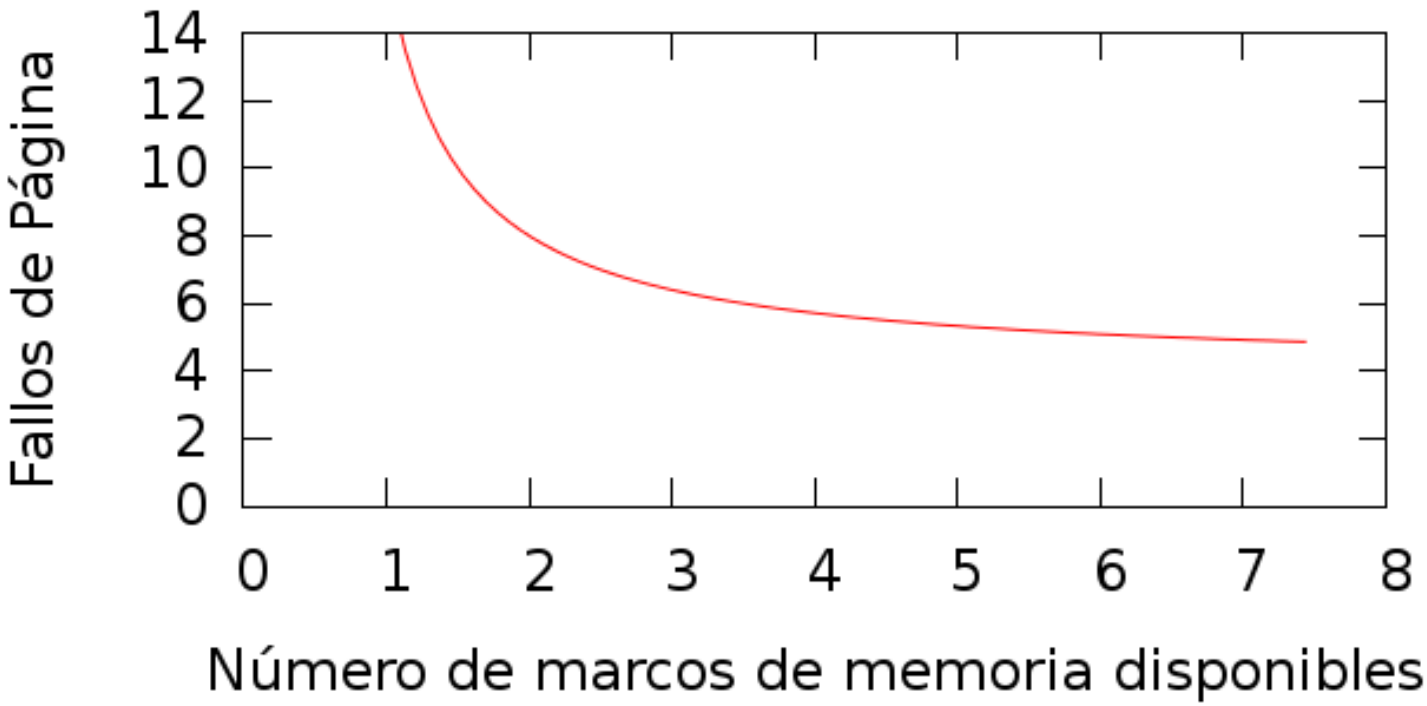

Figura: Relación ideal entre el número de marcos y la cantidad de fallos de página

Anomalía de Belady: Comportamiento de FIFO

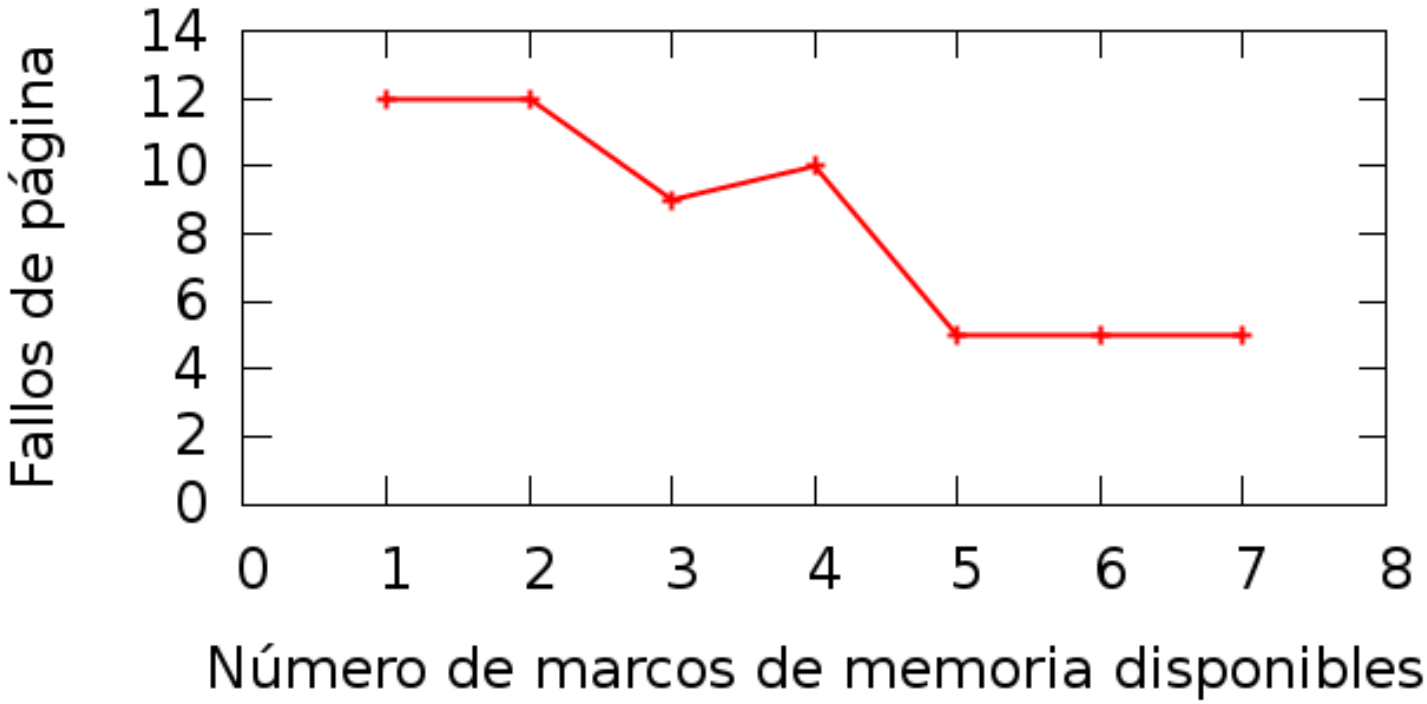

Figura: El algoritmo FIFO presenta la anomalía de Belady con la cadena de referencia 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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

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

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

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

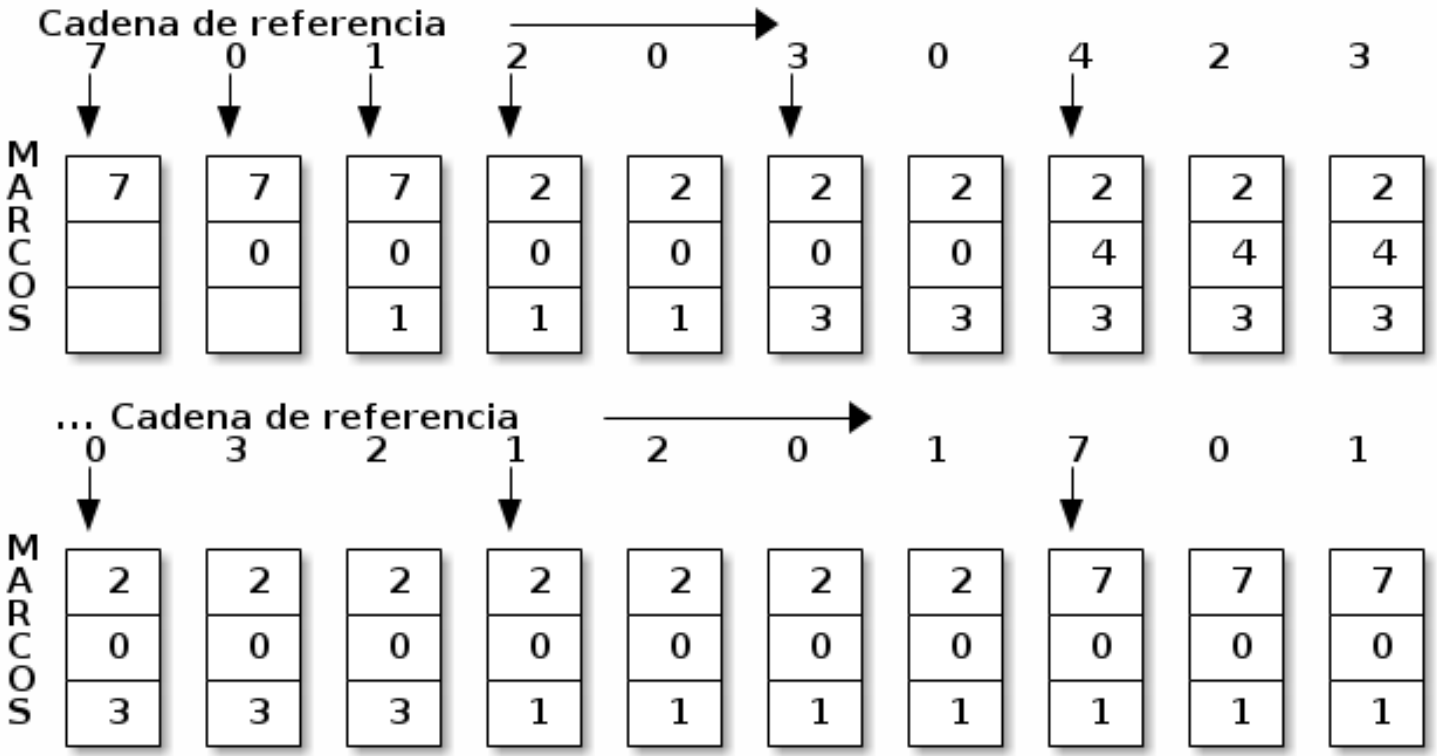

Figura: Algoritmo óptimo de reemplazo de páginas (OPT): 9 fallos

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

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

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)

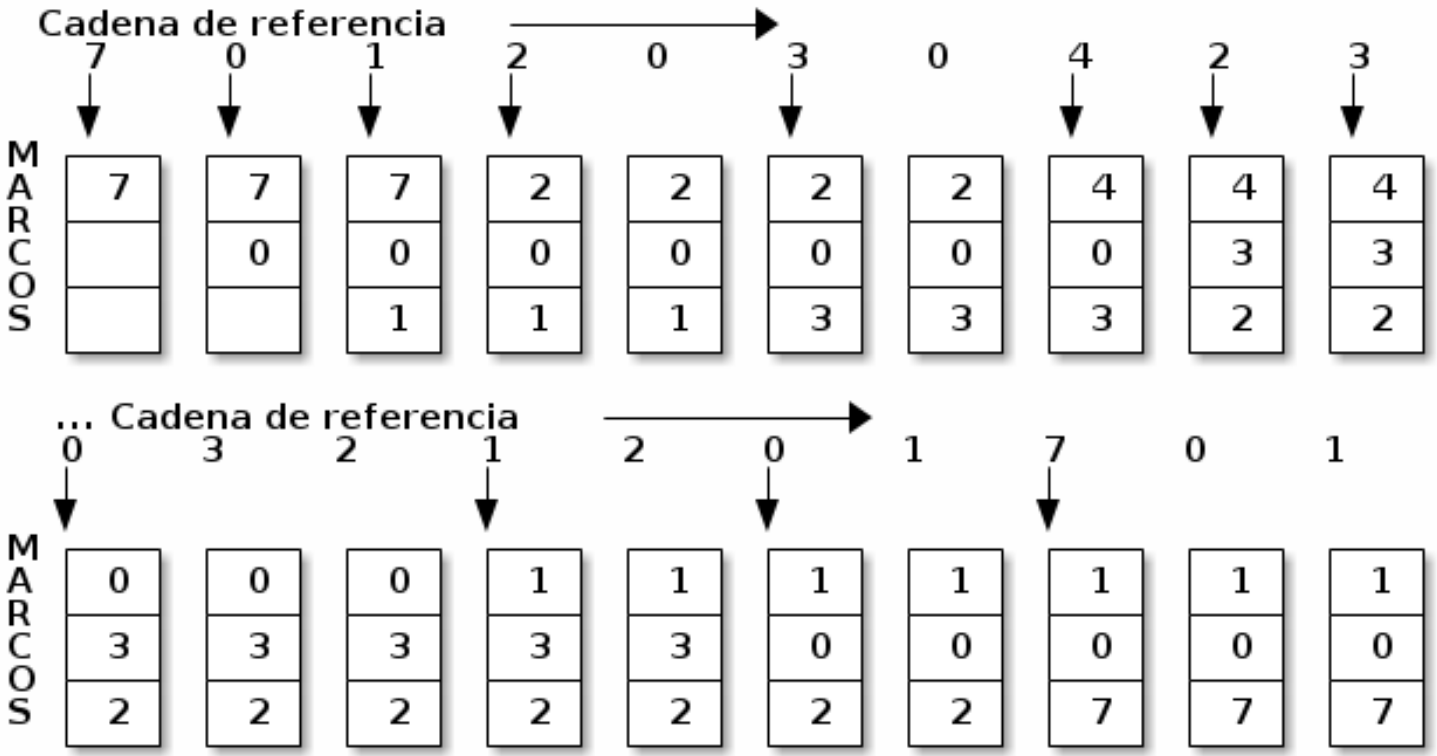

Figura: Algoritmo reemplazo de páginas menos recientemente utilizadas (LRU): 12 fallos

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 contrador 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*

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)

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

Más / menos frecuentemente utilizado (MFU / LFU) (1)

- Dos algoritmos contrapuestos, basados (como LRU) en mantener un contador
  - Miden la *cantidad* de referencias que se han hecho a cada página
- Lógica base:

  MFU Si una página fue empleada muchas veces, probablemente va a ser empleada muchas veces más

  LFU Si una página casi no ha sido empleada, probablemente recién fue cargada, y será empleada en el futuro cercano

Más / menos frecuentemente utilizado (MFU / LFU) (2)

- La complejidad de estos algoritmos es tan alta como LRU, y su rendimiento es menos cercano a OPT
  - Casi no son empleados

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

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

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

Bits adicionales (*columna*) de referencia

- Mecanismo derivado del anterior, dando más granularidad
- Se maneja una *columna* de referencia, de varios bits de ancho
  - Periódicamente, en vez de reiniciar a 0, el valor de todas las entradas se *recorre* a la derecha, descartando el bit más bajo
  - El acceso a un marco hace que se encienda su bit más alto
- Ante un fallo de página, se elige entre los marcos con valor de referencia más bajo

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

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

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 órden 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.

Segunda oportunidad mejorada (2)

- Emplea una lógica como la de *segunda oportunidad*, pero considerando el costo de E/S
- Puede requerir dar hasta cuatro *vueltas* para elegir a la página víctima
  - Aunque cada vuelta es más corta

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

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
- Cuando tenga que traer una página de disco, siempre habrá dónde ubicarla sin tener que hacer una transferencia

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

Ejemplo: Tres sistemas Linux (1)

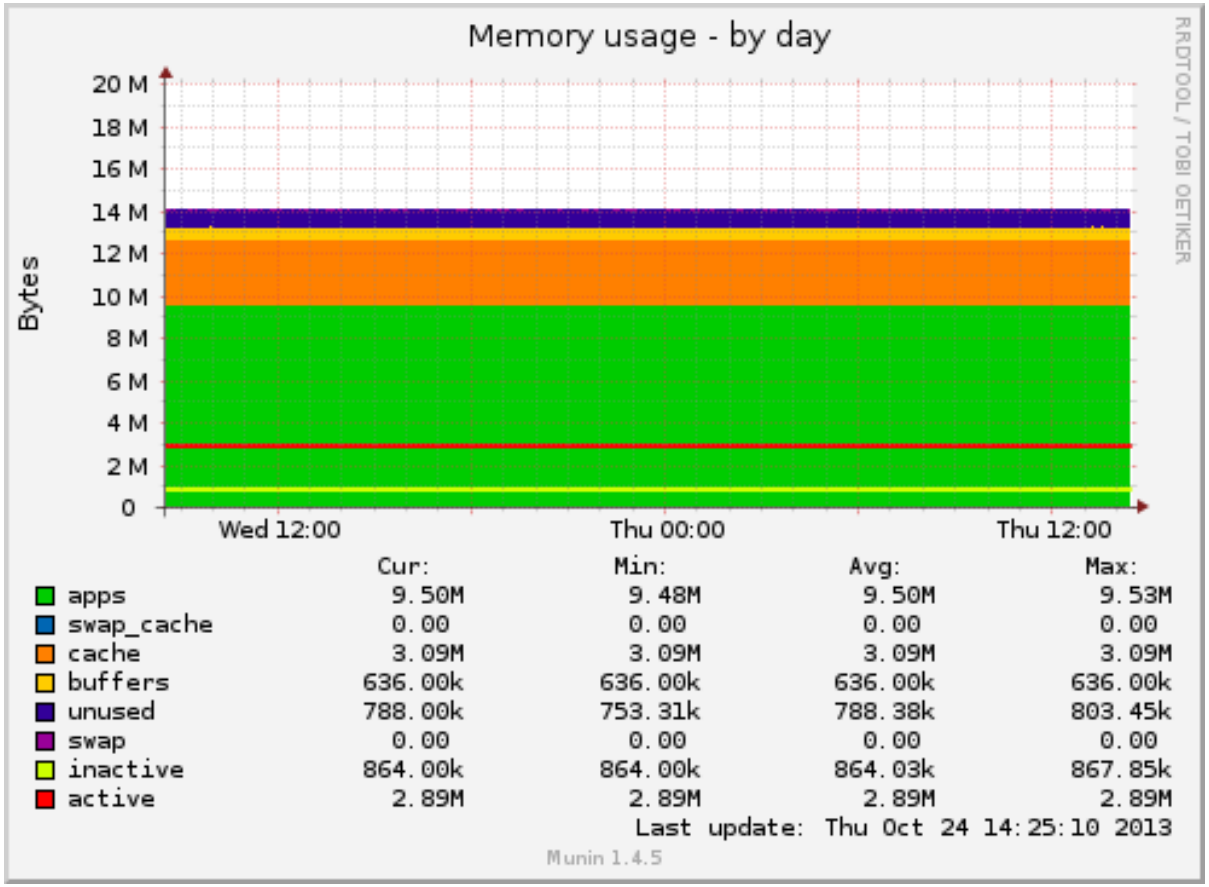

Figura: Manejo de memoria (24 horas) en un sistema embebido (16MB RAM)

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

Ejemplo: Tres sistemas Linux (2)

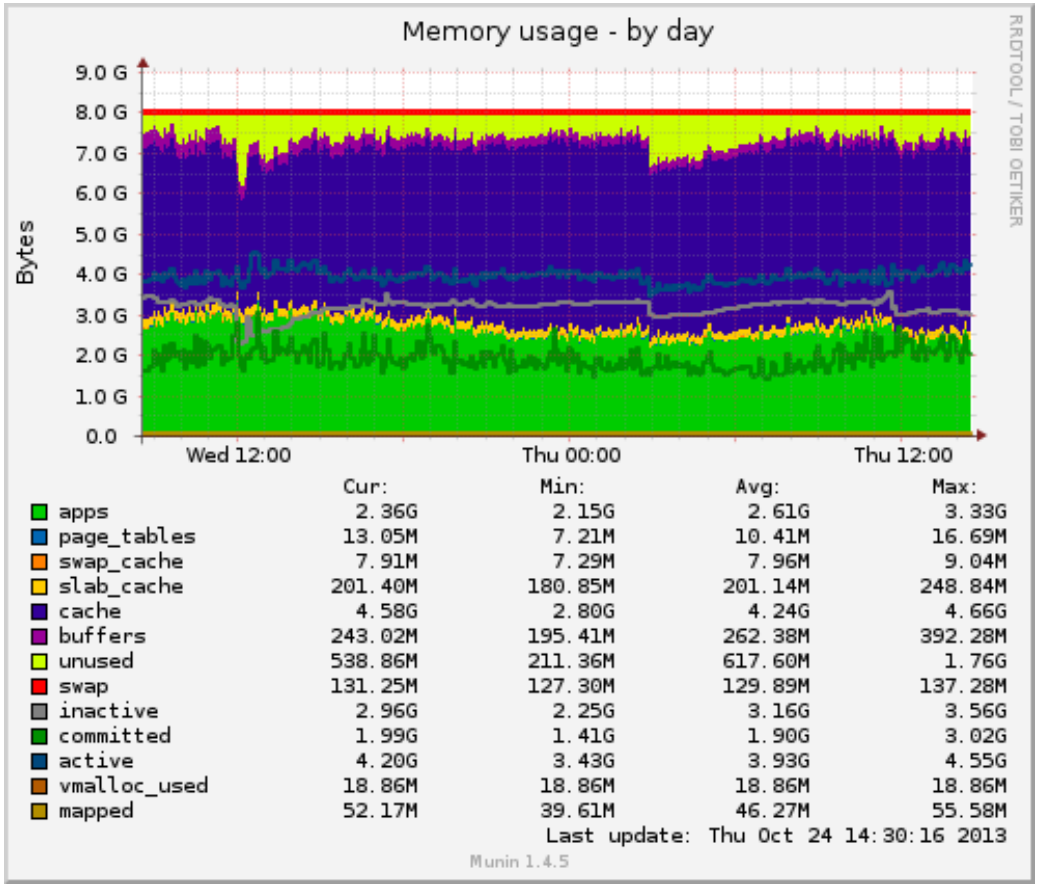

Figura: Manejo de memoria (24 horas) en un servidor medio (8GB RAM)

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

Ejemplo: Tres sistemas Linux (3)

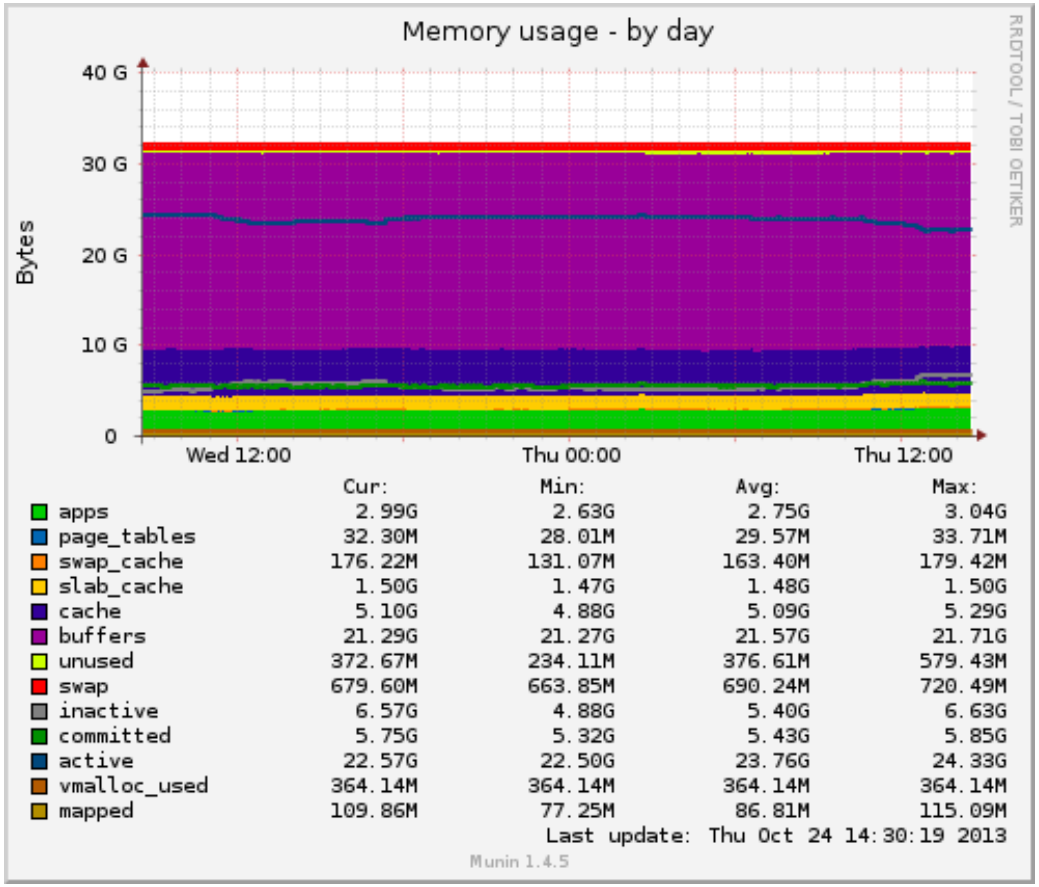

Figura: Manejo de memoria (24 horas) en un servidor grande (32GB RAM)

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

Índice

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

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

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

Puramente sobre demanda: *Demasiado flojo*

- El esquema de paginación *puramente* sobre demanda puede resultar *demasiado flojo*
- Ser un poco *más ansioso* aseguraría un mejor rendimiento
- Conviene determinar un *mínimo* utilizable de marcos
  - Si asignamos por debajo del mínimo, sufre el rendimiento

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

Mínimo de marcos: Arquitectura y direccionamiento

- Hasta ahora hemos simplificado asumiendo que cada instrucción puede generar sólo un fallo de página
- Independientemente de la arquitectura, cada instrucción puede desencadenar *varias* solicitudes
  - Una solicitud, la lectura de la siguiente dirección a ejecutar (¡recuerden: von Neumann!)
  - Otra, la dirección de memoria referida
  - Por ejemplo, si el flujo *brinca* a `0x00A2C8`, y esta instrucción es `load 0x043F00`, para satisfacerla requerimos dos páginas: `0x00A` y `0x043`
  - → Requerimos un mínimo de dos páginas
- Pero... ¿Y las *referencias indirectas*?

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

Mínimo de marcos: Referencias indirectas

- *Casi* todas las arquitecturas permiten hacer *referencias indirectas* a memoria
- Una instrucción de acceso a memoria (`load`, `store`) especifica una dirección de memoria
- Y *esta dirección* guarda la ubicación de memoria
- Por ejemplo, `0x043F00` indica la carga de `0x010F80`
  - → Satisfacer al `load 0x043F00` requerirá entonces tres páginas: `0x00A`, `0x043` y `0x010`
- . . . Y algunas arquitecturas (principalmente antiguas) permitían niveles ilimitados de indirección
  - Por ejemplo, por medio de un *bit de indirección*
  - En dado caso, es imposible *asegurar* un límite máximo → Es común que el MMU haga un *conteo de referencias* para evitar caer en un ciclo sin fin

Instrucciones con operandos en memoria

- Las arquitecturas RISC introdujeron requisitos de regularidad que incluyen el que la aritmética opere exclusivamente sobre los registros del procesador
- Las arquitecturas más antiguas permiten que los operandos y resultado sean direcciones de memoria
  - ¿Antiguas? De antes de que la diferencia de velocidad entre CPU y memoria fueran tanta
  - Recordemos que la principal arquitectura actual tiene *herencia* desde 1970...
  - Si en un x86, en `0x00AC28` tenemos `ADD [edx], [ecx]`, en `EDX` el valor `0x010F80` y en `ECX` el valor `0x043F00`,
    - En el *acumulador* `EAX` obtendremos la suma del *contenido* de los dos operadores
    - $\rightarrow$ Tres referencias a memoria en una sola instrucción

El nivel *deseable* de marcos

- Con estos lineamientos determinamos ya un mínimo absoluto
  - *Muy* bajo, poniéndolo en el contexto de los sistemas actuales
- ¿Cómo puede el sistema determinar un nivel *deseable* de marcos por proceso?
  - Depende *siempre* del estado actual del sistema
- Podríamos intentar satisfacer los requisitos *totales* de uno de los procesos
  - A menor cantidad de fallos de página, mayor rendimiento
  - Pero si reducimos el *grado de multiprogramación*, reducimos el uso efectivo del procesador

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

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 marcos

Modulando la asignación proporcional

- Mínimos: El esquema debe cuidar nunca asignar por debajo del mínimo de la arquitectura
  - Si P_2 ocupara sólo 10 marcos de memoria física, en una arquitectura x86 no deberían asignársele menos de 3 marcos
- Desbalance por procesos *obesos*
  - Si P_1 crece al doble de su tamaño virtual, hay que cuidar tener *umbrales máximos* para no castigar de más a los demás procesos del sistema
- ¿Manejo de prioridades?
  - Si el sistema maneja prioridades, podrían incluirse ponderadas, otorgando proporcionalmente más marcos a los procesos con mayor prioridad

Sufrimiento ante la entrada de nuevos procesos

- El esquema de asignación proporcional sufre cuando son admitidos nuevos procesos, cambia el tamaño en memoria virtual de alguno de los existentes o (aunque menos) finalizan los que están en ejecución
- Deben recalcularse los totales, y probablemente reducir de golpe el espacio asignado a los procesos existentes

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

Desperdicio de recursos

- El patrón de uso de memoria física de un proceso no necesariamente guarda correspondencia con su tamaño en memoria virtual
- Pueden emplear mucho menores requisitos en *determinadas secciones* de su ejecución
- Recordar este punto → *Conjunto activo*

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

Ámbitos del algoritmo de reemplazo de páginas

Respondiendo a los problemas que abre la sección anterior, podemos discutir el *ámbito* en el que operará nuestro algoritmo de reemplazo de páginas

- Reemplazo local
- Reemplazo global
- Reemplazo global con prioridad

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

Reemplazo local

- Mantenemos tan estable como sea posible el cálculo de marcos de memoria por proceso
  - Cuando se presente un fallo de página, sólo se consideran aquellas pertenecientes *al mismo proceso*
- El proceso tiene asignado un espacio de memoria física
  - Lo mantendrá mientras el sistema operativo no tome alguna decisión para modificarlo

Reemplazo global

- Los algoritmos de asignación determinan el espacio asignado /al momento de su inicialización
  - Pueden *influir* en los algoritmos de reemplazo
  - P.ej. dando *mayor peso* a los marcos de un proceso que excede su asignación para ser elegidas como víctima
  - . . . O pueden operar bajo un esquema *laissez-faire*, buscando que el sistema se *auto-regule* basado en las necesidades reales momento a momento
- Operan sobre el espacio completo de memoria
  - La asignación física a cada proceso puede variar según el estado del sistema, momento a momento

Reemplazo global con prioridad

- Esquema mixto
- Permite que un proceso *sobrepase su límite*
  - Pero sólo siempre que le *robe* espacio en memoria física *sólo* a procesos de *prioridad inferior* a él
- Consistente con el comportamiento de los algoritmos planificadores
  - Siempre da preferencia a un proceso de mayor prioridad por sobre los de menor prioridad
- Puede también operar bajo *concesiones temporales*, buscando equilibrar posteriormente

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

Comparando los ámbitos de reemplazo

Reemplazo local Más rígido; no permite aprovechar las menores demandas de unos procesos para favorecer a los que tienen mayores demandas en un momento dado

Reemplazo global (ambos) Puede llevar a rendimiento inconsistente fuera del control de cada uno de los procesos

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

¿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. . .
  - $\rightarrow$ Sólo podemos prometer *tiempo real suave*

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

Hiperpaginación: Definición

- 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*)

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

¿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...

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

¿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
  - $\rightarrow$ 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?

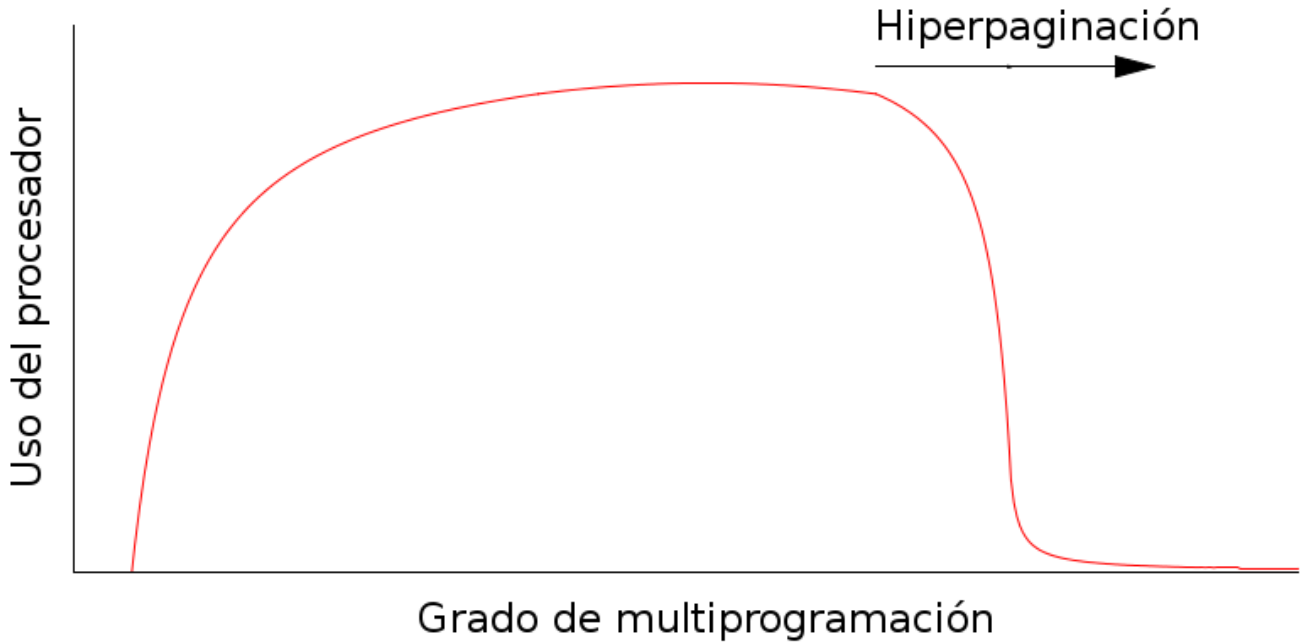

Figura: Al aumentar demasiado el grado de multiprogramación, el uso del CPU cae abruptamente y caemos en la hiperpaginación (Silberschatz, p.349)

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

¿A cuál proceso *castigar*?

- Al de menor prioridad
- Al que esté causando más fallos
- Al que esté ocupando más memoria
- ...

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

El conjunto activo

- El *conjunto activo* es una clara aproximación a la *localidad de referencia*
- El conjunto de páginas con que un proceso está trabajando *en un momento dado*
  - ¿Qué significa *un momento dado*?

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

El conjunto activo

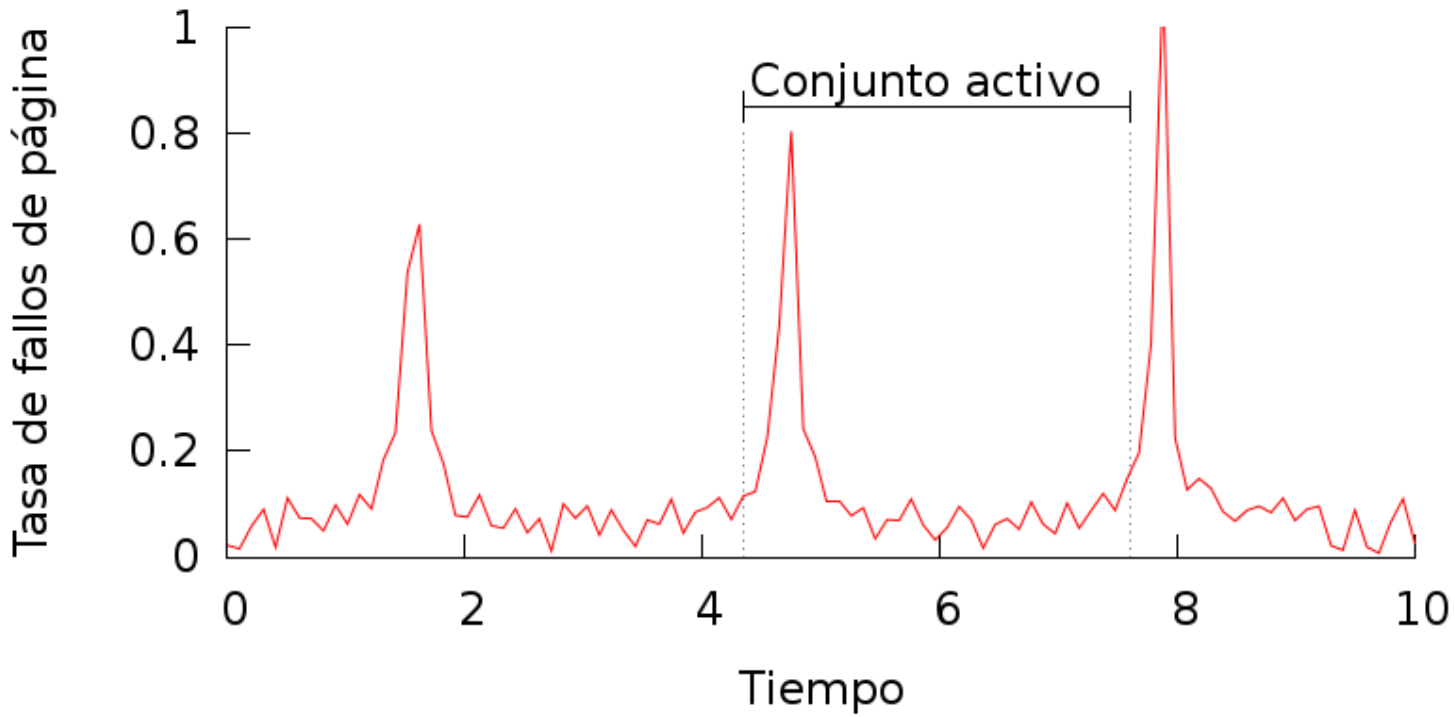

Figura: Los picos y valles en la cantidad de fallos de página de un proceso definen a su *conjunto activo* (Silberschatz, p.349)

El conjunto activo y el espacio en memoria

- 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
- . . . 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