Sistemas operativos. Memoria Virtual

April 30, 2012

Contenidos I

Memoria Virtual

- Es la posibilidad de ejecutar un programa que no está totalmente en memoria
- Tiene sentido pues hay zonas del programa que puede que no sean referenciadas nunca
  - condiciones de error poco frecuentes
  - opciones poco usadas
  - variables sobredimensionadas

Memoria Virtual

- Ejecutar un programa que no está totalmente en memoria permite que
  - Los programas puedan ser más grandes que la memoria física instalada en la máquina
  - Se pueda aumentar el grado de multiprogramacíon
  - Sea necesaria menos e/s para intercambiar programas
- La forma mas usual de implementarla es con paginación bajo demanda
- Puede implementarse con segmentación bajo demanda pero puede requerir hacer compactaciones (IBM OS/2 hasta 1.3 lo hacía)

Paginación bajo demanda

- Es un sistema paginado con intercambio en donde se intercambia páginas cuando no son necesarias
- El HARDWARE debe suministrar una tabla de páginas con un bit que indique la presencia (o no) de la página en memoria de manera que
  - Se referencia una página cuya entrada en la tabla de páginas tiene el bit de presencia a 1: se accede normalmente
  - Se referencia una página cuya entrada en la tabla de páginas tiene el bit de presencia a 0: se produce una *excepción*

Paginación bajo demanda

- Si no se referencian páginas que no están en memoria el proceso se ejecuta normalmente..
- Cuando se produce la excepción el control se transfiere a S.O.
  - El S.O. salva el estado de la CPU
  - Determina que la excepción es un fallo de página
  - Asigna un marco libre, localiza la página en disco e inicia la transferencia
  - Cuando la transferencia se completa, el S.O. actualiza la tabla de páginas del proceso
  - Cuando se reanuda el proceso, reintenta la instrucción que produjo el fallo de página (pues no se llegó a ejecutar)

Paginación bajo demanda

- Hace falta también una zona de almacenamiento secundario donde almacenar las páginas
  - Hacerlo en un archivo es más flexible, pero hay que sufrir las indirecciones del sistema de archivos
  - Hacerlo en una partición o en un disco dedicado es más rápido, pero se pierde flexibilidad
- Se pueden ejecutar programas más grandes que la memoria física a costa de una merma en la velocidad
- Dado que a menudo hay que traer páginas a memoria, surge el problema de cuales reemplazar
- Ademaś de los distintos algoritmos hay que considerar si se hace reemplazo local o global
- Otro aspecto a tener en cuenta son los criterios de asignación de marcos a los procesos

Rendimiento de la paginación bajo demanda

- El servicio del fallo de página comprende las siguientes tareas
  a. Servir excepción del fallo de página
     - Trasferir el control al S.O.
     - Salvar estado del proceso
     - Determinar que la excepción es un fallo de página
     - Localizar la página en el intercambio
     - Asignar marco libre
  b. Transferencia de la página
     - Esperar en cola de dispositivo
     - Esperar tiempo de búsqueda y latencia
     - Transferir página al marco
     - Actualizar tabla de páginas
  c. Reanudar proceso
     - Restablecer estado de proceso
     - Reanudar ejecución

Rendimiento de la paginación bajo demanda: ejemplo

- Supongamos un sistema actual que opera a 2GHz, donde el tiempo de acceso a memoria es del orden de 5 ns (valor típico de una memoria DDR2), y donde el dispositivo donde está el intercambio tiene un tiempo de búsqueda promedio de 9,5 ms y una velocidad de transferencia de 40Mb/segundo
- En este caso podemos estimar una cota superior de 10ms para servir el fallo de página (9,5 ms la búsqueda, 0,1 ms la transferencia de una página de 4K y quedan 0,4 ms que son mas que suficientes para la latencia y el resto de las tareas)
- Si la probabilidad de un fallo de página es de 10^{-6} (se produce un fallo de página cada millón de referencias a memoria), el tiempo de acceso efectivo a memoria pasaría a ser
  $t.a.e. = (1 - 10^{-6})x5ns + 10^{-6}10ms \approx 5ns + 10ns = 15ns$

Rendimiento de la paginación bajo demanda

- Aumentar la memoria física del sistema hace disminuir la probabilidad de un fallo de página
- A medida que disminuye la probabilidad de un fallo de página disminuye el tiempo de acceso efectivo y por tanto aumenta la velocidad aparente de ejcución
- La memoria virtual permite ejecutar un proceso que no reside totalmente en memoria a costa de una merma el la velocidad de ejecución
- Para que el funcionamiento de un sistema con memoria virtual sea optimo hay que conseguir que el número de fallos de página sea mínimo

Reemplazo de páginas

- En la gestión de la memoria virtual, toma gran importancia lo que se denomina *política de reemplazo*, que es la que decide que página de las presentes en memoria es la que va a ser sustituida
- En esta *política* se encuentran involucrados varios conceptos, que aunque distintos, están fuertemente interrelacionados
  a. La cantidad de memoria (número de marcos) física asignada a cada proceso en el sistema
  b. Si el conjunto de páginas a tener en cuenta a la hora de ser reemplazadas incluye solamente a las del proceso que ha provocado el fallo de página o a todas las residentes en memoria (de los distintos procesos)
  c. De entre las páginas consideradas, cual es la que debe seleccionarse para ser reemplazada

Asignación de marcos y reemplazo de páginas

- El punto *a)* plantea lo que se conoce como el problema de la *asignación de marcos*. Las soluciones a este problema pasan por intentar asignar a cada proceso un número de marcos variable que se adapta a la localidad del proceso (es decir a sus necesidades de memoria en las distintas fases de su ejecución)
- El punto *b)* plantea lo que se conoce como *reemplazo local* o *reemplazo global*
- El punto *c)* es lo que es normalmente referido como *política de reemplazo* o *algoritmos de reemplazo de páginas*

Reemplazo y asignación

- A la hora de asignar marcos para la ejecución de un proceso hay dos aproximaciones
  - Asignación fija: El número de marcos asignados a un proceso es fijo
  - Asignación variable. El número de marcos asignados a un proceso evoluciona con las necesidades de memoria del proceso
- El reemplazo de páginas puede ser
  - Local: Un fallo de página de un proceso solo puede reemplazar una página de dicho proceso
  - Global: Un fallo de página de un proceso puede reemplazar una página de otro proceso

Reemplazo y asignación

- No todas las combinaciones son posibles: por ejemplo no puede haber reemplazo global con una ténica de asignación fija
- La solución mas utilizada actualmente es asignación variable con reemplazo global
  - Muy sencilla de implementar
  - Se combina con la existencia de *"pool"* de marcos libres

pool de marcos libres

- La mayoría de los sistemas cuenta con un depósito (*pool*) de marcos libres
- Si hay que hacer reemplazo
  - Se trae la página nueva a uno de los marcos libres
  - La página *reemplazada* pasa a ser marcada como libre y añadida al "*pool*" de marcos libres
- No hay que esperar para traer la página nueva a memoria
- Ante un nuevo fallo, es posible que la página que ha provocado este nuevo fallo esté en memoria en el *pool* de marcos libres y no hay que traerla de disco

Algoritmos clásicos de reemplazo

- Los algoritmos clásicos de reemplazo de página son
  - **FIFO**
  - **óptimo**
  - **LRU**
- Lo mas usual es que el hardware proporcione un bit de referencia y uno de modificada (*dirty bit*) para cada página. Con esta ayuda, los algoritmos mas usuales son
  - FIFO con segunda oprtunidad
  - No Usada Recientemente
  - Ni Usada Ni Modificada Recientemente
  - otras aproximaciones LRU ...

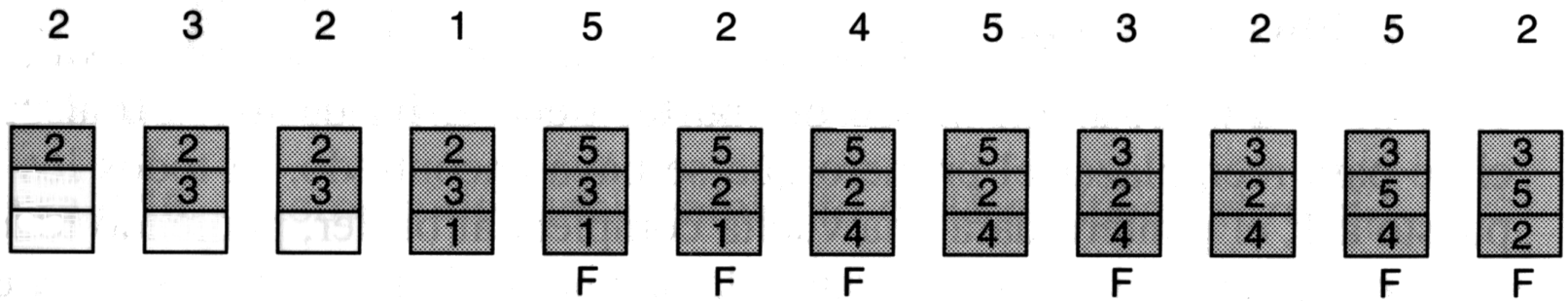
Algoritmos de reemplazo de página

Algoritmo FIFO

- Se reemplaza la primera página en entrar (FIRST IN FIRST OUT)
- Implementación muy sencilla
- Presenta la anomalía de *Belady*: para ciertos ejemplos concretos de cadenas de referencias a memoria es posible que al aumentar el número de marcos aumente el número de fallos de página
  - Probar la cadena de referencias *1 2 3 4 1 2 5 1 2 3 4 5* para tres y cuatro marcos

Algoritmo FIFO: ejemplo

- La cadena de referencia *2 3 2 1 5 2 4 5 3 2 5 2* con tres marcos produce 5 fallos

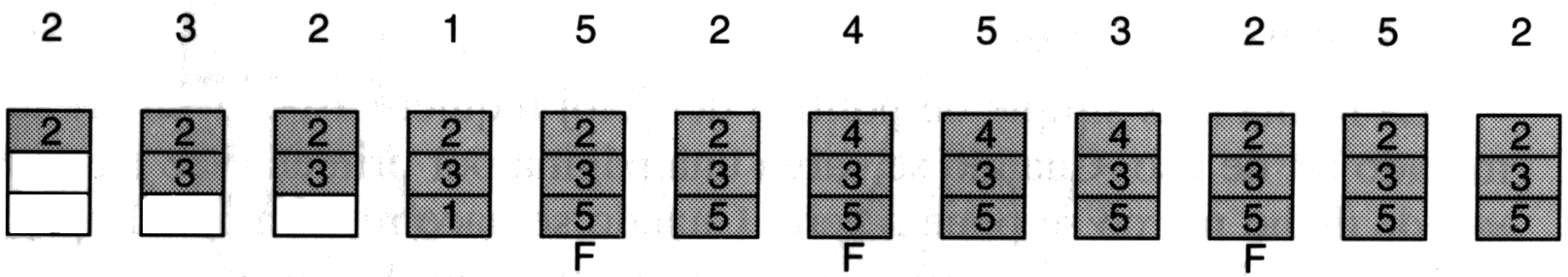


Algoritmo Óptimo

- Es el que produce menos fallos de página para cualquier cantidad de marcos
- Se reemplaza la página que va a tardar mas tiempo en ser referenciada
- No puede implementarse pues implicaría conocer de antemano las páginas que va a referenciar el proceso
- Aunque no puede implementarse se utiliza como referencia para los demás algoritmos

Algoritmo Óptimo: ejemplo

- La cadena de referencia *2 3 2 1 5 2 4 5 3 2 5 2* con tres marcos produce 3 fallos



Algoritmo LRU

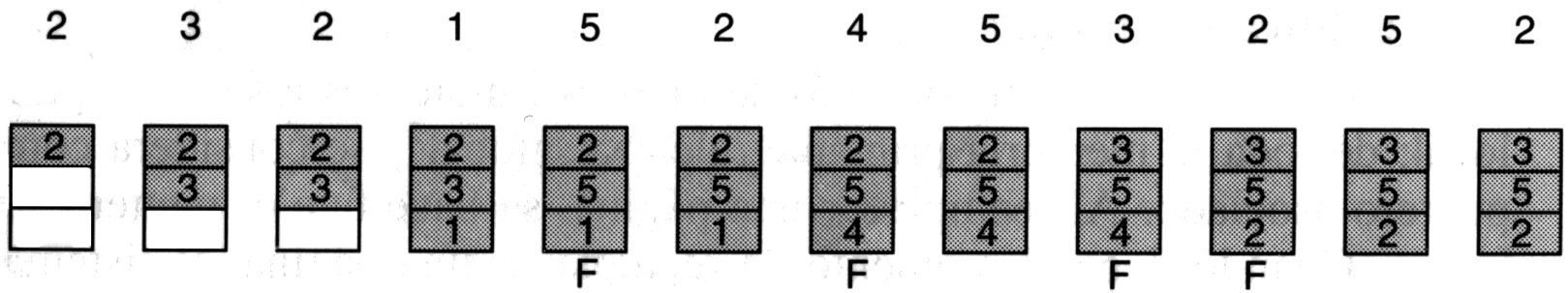
- Reemplaza la página menos recientemente usada (Least Recently Used)
- Es como el óptimo pero con la cadena de referencias invertida en el tiempo
- Produce buenos resultados puesto que por el principio de localidad temporal, páginas referenciadas recientemente es probable que sean referenciadas proximamente
- Se adapta muy bien a la localidad del programa

Algoritmo LRU

- Dos posibles implementaciones
  - **Contadores** Se asocia a cada página un contador que representa el instante en que fue referenciada, pudiendo determinar la que hace más tiempo que no se referencia por el valor del contador
  - **Lista** Cada página referenciada se coloca al final de una lista, la primera de la lista es la que hace más tiempo que no se referencia
- Ninguna de las implementaciones es factible por la carga que supondría manejar los contadores (o la lista) **con cada referencia a memoria**

Algoritmo LRU: ejemplo

- La cadena de referencia *2 3 2 1 5 2 4 5 3 2 5 2* con tres marcos produce 4 fallos



Algoritmos de reemplazo de página

Algoritmo LRU: ejemplo

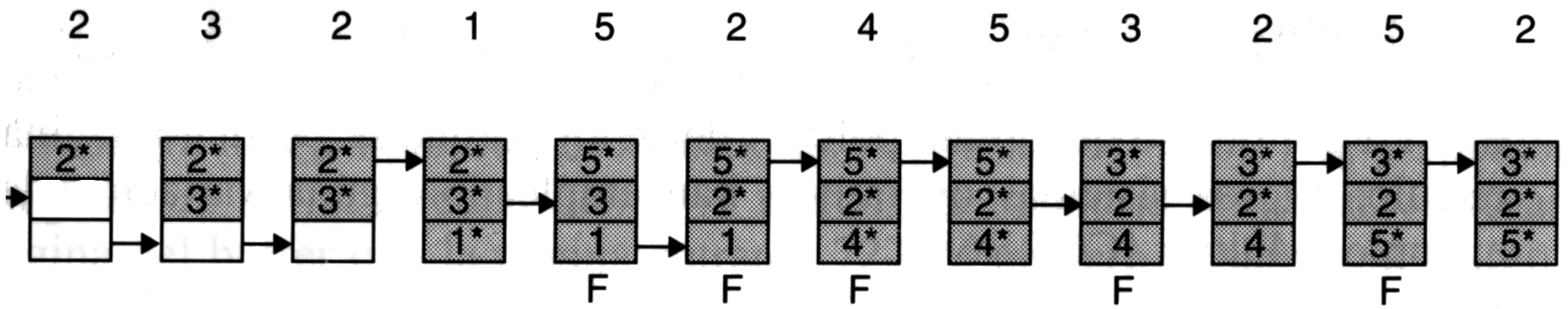
- El algoritmo LRU produce unos resultados razonablemente buenos pero no puede implementarse.
- En los sistemas con paginación bajo demanda el *hardware* tipicamente proporciona un *bit de referencia* y un *bit de modificación (dirty bit)*
- Con estas ayudas se suelen implementar algoritmos que se aproximan al LRU con la idea de adaptarse a la localidad del proceso
- Los mas usuales son los de *segunda oprtunidad*, reloj, empleo de bits de referencia adicionales ...

Algoritmo Segunda Oportunidad

- Es basicamente un FIFO en el que además se tiene en cuenta el bit de referencia
- Implementación muy sencilla: cola circular con las páginas en la que se almacena tambien el bit de referencia
- Cuando hay que reemplazar se mira el índice que indica la página siguiente a reemplazar
  - Si el bit de referencia está a 0 se reemplaza
  - Si el bit de referencia esta a 1, se pone a 0 y se avanza el índice a la siguiente...

Algoritmo segunda Oportunidad: ejemplo

- La cadena de referencia *2 3 2 1 5 2 4 5 3 2 5 2* con tres marcos produce 5 fallos



Variaciones del algoritmo de Segunda Oportunidad

- Utilizando la cola circular de páginas pero con el par de bits (referencia,modificacion)
  1. A partir del índice actual se busca la primera página con (0,0) y se reemplaza
  2. Si no se encuentra ninguna, se vuelve a recorrer la cola, buscando una página con (0,1). La primera que se encuentra es la que se reemplaza. Al mismo tiempo que se busca se van limpiando los bits de referencia de las páginas por las que se pasa
  3. Si no se encuentra ninguna se vuelve al paso 1

Variaciones del algoritmo de Segunda Oportunidad

- Utilizando dos índices para recorrer la lista.
  - A las páginas que se examinan con el primer índice se les limpia el bit de referencia.
  - Las páginas que al examinarlas con el segundo índice tiene el bit de referencia a 0 son marcadas como libres
  - El algoritmo se ejecuta a intervalos regulares de tiempo y se mantienen en memoria las páginas que fueron referenciadas desde que fueron examinadas con el primer índice hasta que se examinan con el segundo
  - El proceso de robo de páginas en UNIX utiliza este algoritmo

Uso de bits de referencia adicionales

- El S.O. guarda un contador para cada página
- El S.O. periodicamente comprueba los bits de referencia de cada página, y lo guarda en su contador introduciéndolo por la izquierda y desplazando los otros bits a la derecha
- La página con el valor mas pequeño del contador es una aproximación a la *menos recientemente usada*
- No es la *menos recientemente usada* porque los bits se comprueban a determinados intervalos de tiempo, no cada acceso a memoria
- Por ejemplo si tenemos 3 páginas A, B y C cuyos contadores son 1110000000000000, 1110000000000000, 0000000001000000, respectivamente,
  - La página que hace más tiempo que no se referencia es la C (las últimas 8 veces que el S.O. comprobó los bits no había sido referenciada
  - Las páginas A Y B han sido ambas referenciadas los tres últimos intervalos de tiempo en que el S.O. comprobó los bits, sin embargo no sabemos cual de ellas ha sido referenciada más recientemente

Hiperpaginación (*Thrashing*)

- Se produce hiperpaginación cuando un sistema pasa más tiempo paginando que ejecutando
- Un proceso que tiene asignados menos marcos que los que está usando activamente fallará continuamente de página ya que cada fallo reemplazará una página tambien usada
- Si el reemplazo es global (un proceso puede reemplazar páginas de otros) puede *contagiar* el problema
- Para evitar la hiperpaginación se intenta asignar a cada proceso un número de marcos suficiente. Dos modelos principales
  - modelo del *working set*
  - modelo de la frecuencia del fallo de página

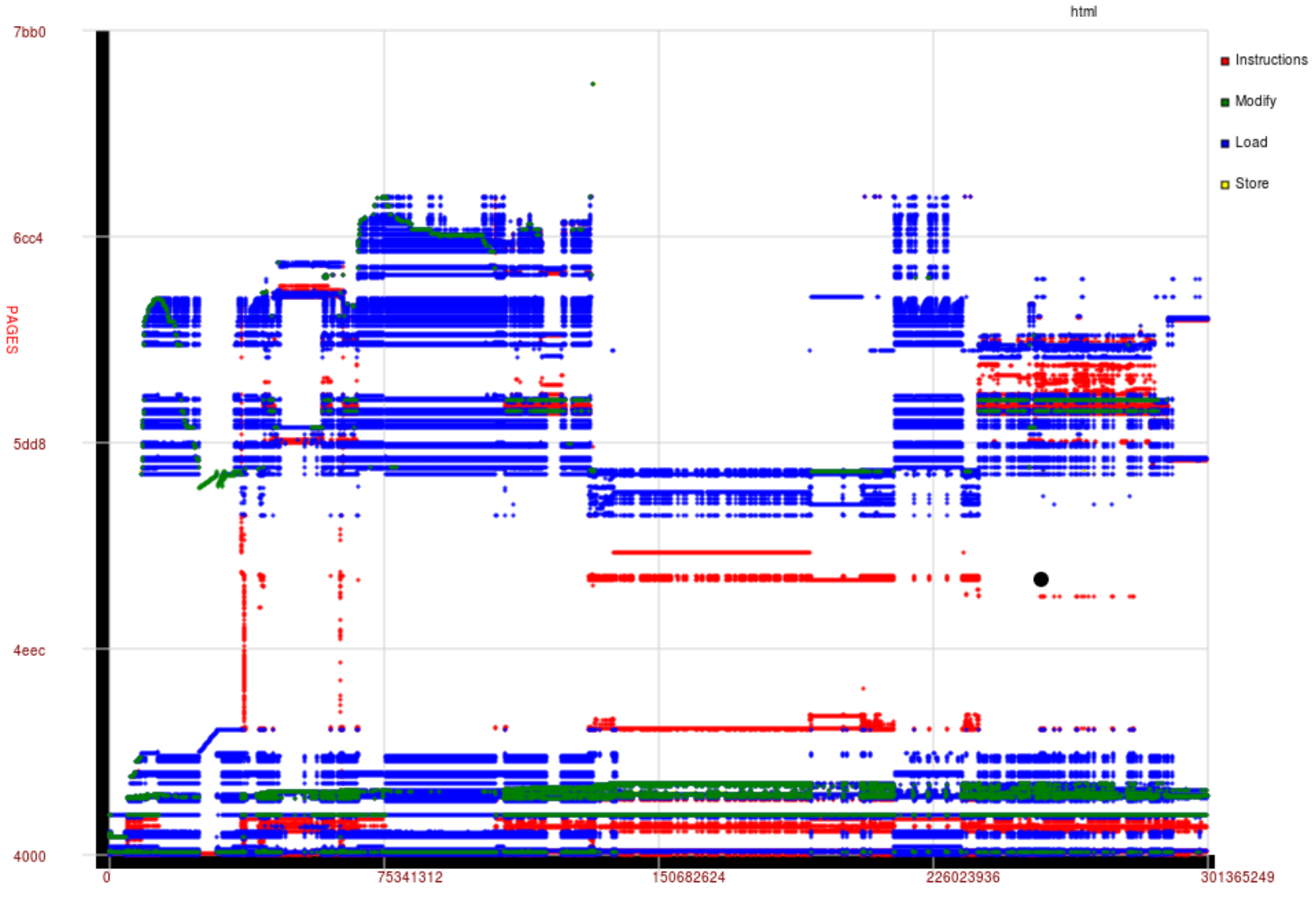
Localidad de un programa

- Este principio afirma que a medida que un programa se ejecuta pasa de una localidad a otra
- Una localidad es un conjunto de páginas que se referencian activa y conjuntamente
- Un programa en general se compone de varias localidades, que se pueden solapar
- Dicho de otra manera: los conjuntos de páginas referenciadas por un proceso durante su ejecución estan agrupados en el espacio (*localidad espacial*) y en el tiempo (*localidad temporal*)

Localidad espacial y temporal

- **Localidad temporal:** Direcciones de memoria referenciadas recientemente es probable que vuelvan a ser referenciadas proximamente
  - bucles
  - funciones, procedimientos, subrutinas ...
  - pilas
  - contadores, acumuladores...
- **Localidad espacial:** Si se referencia una dirección de memoria es probable que se referencie una localización cercana
  - arrays y estructuras
  - ejecución secuencial
  - variables relaccionadas suelen declararse juntas

Localidad de un programa



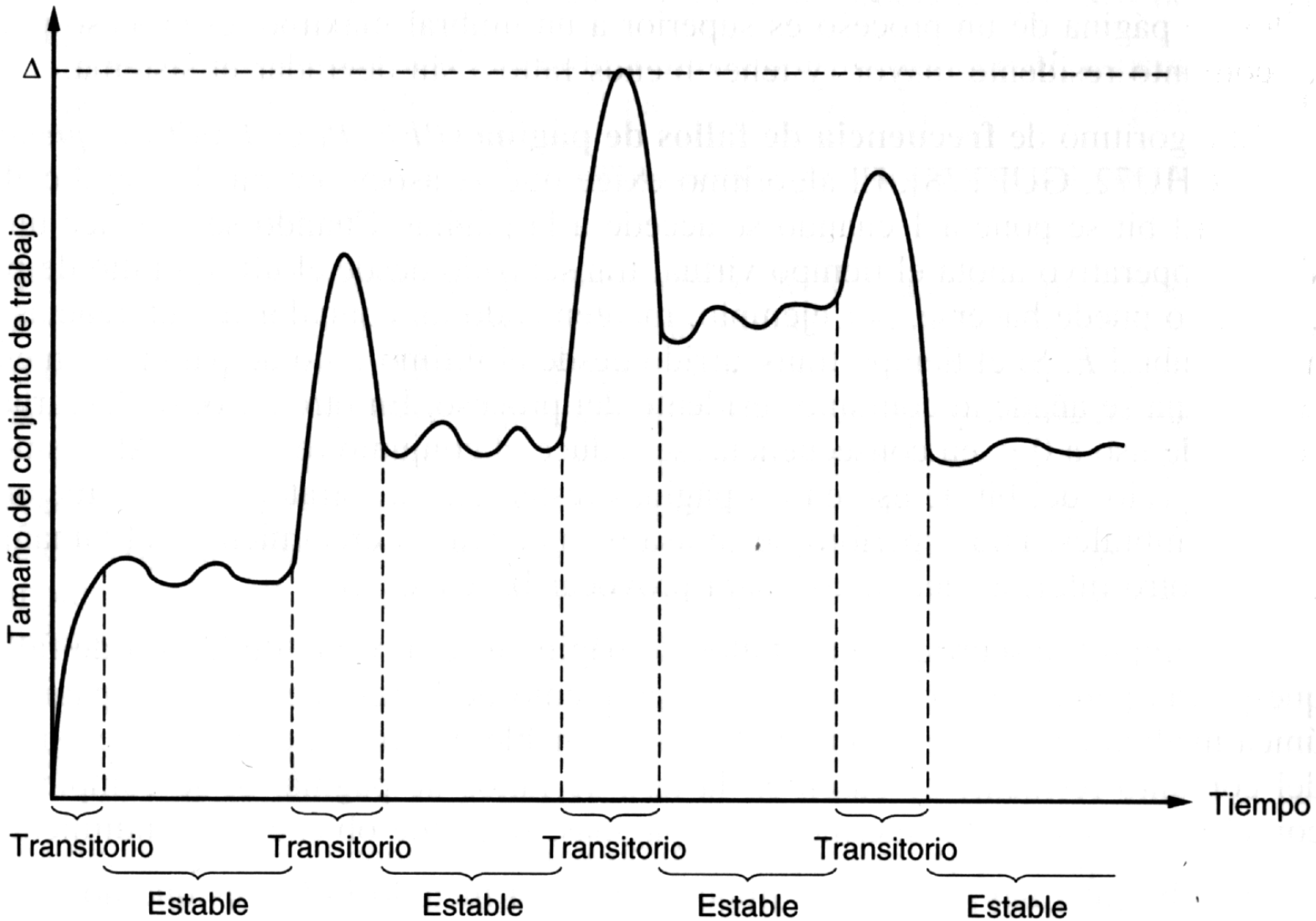
Working Set

- Se basa el el principio de *localidad*
- Para un proceso se define el *working set* de ventana Δ en un instante t, $W(t, \Delta)$ como el conjunto de páginas referenciadas entre los instantes $t - \Delta$ y t
- Se trata de las páginas referenciadas en los últimos Δ instantes
- Si Δ se escoge adecuadamente puede adaptarse a la localidad del programa. Si Δ es demasiado pequeño, no abarcará todo el conjunto de trabajo; si Δ es demasiado grande, puede solapar varias localidades
- La idea es que asignarle a cada proceso el número de marcos necesario para que pueda contener su conjunto de trabajo
- Se implementan aproximaciones puesto que el modelo ideal implicaría trabajar en cada referencia a memoria.

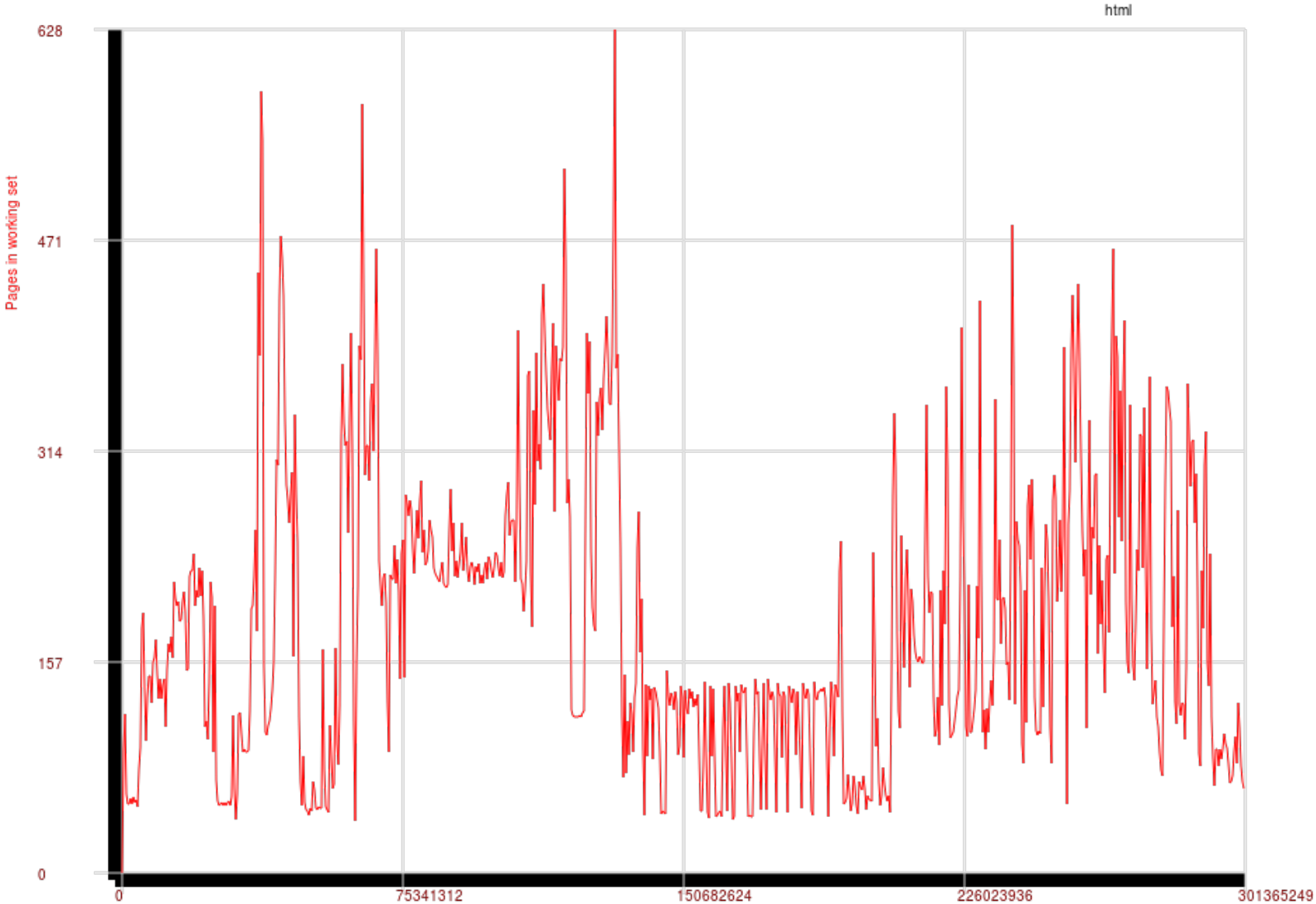
Working Set

- El Working Set de un proceso va cambiando a lo largo de su ejecución
- Durante algunas etapas de la ejecución de un proceso permanece estable
- Entre etapas estables hay periodos transitorios
- El S:O: debe llevar contabilidad de los WS de todos los procesos de manera que la memoria física utilizada se corresponda con la suma de los WS de los procesos en ejecución
  - Si a un proceso no puede asignarle su Workin Set suspenderá su ejecución hasta que quede más memoria disponible

Working Set Real



Working Set Real



Frecuencia del fallo de página

- Es un método efectivo para controlar la hiperpaginación
- Se establece un límite de tiempo,
- Cuando se produce un fallo de página se compara el tiempo trascurrido desde el último fallo con el límite
  - Si es menor que el límite, la nueva página se AÑADE al conjunto de páginas residentes del proceso (se aumenta el número de marcos asignados al proceso)
  - En caso contrario, todas las páginas que no han sido referenciadas desde el último fallo de página se descartan, disminuyendo así el número de marcos asignados al proceso
- Es suficiente con que el *hardware* proporcione un bit de referencia

Frecuencia del fallo de página

- Puede implementarse tambien con dos límites, en este caso el tiempo trascurrido desde el último fallo con el límite
  - Si es menor que el límite inferior la nueva página se AÑADE al conjunto de páginas residentes del proceso (se aumenta el número de marcos asignados al proceso)
  - Si es mayor que el límite superior, todas las páginas que no han sido referenciadas desde el último fallo de página se descartan, disminuyendo así el número de marcos asignados al proceso
  - Si está comprendido entre los dos límites se reemplaza una página del proceso

Segmentación bajo demanda

- Si no se dispone de hardware de paginación, es posible implementar memoria virtual con segmentación bajo demanda
- Para ello es necesario
  - intercambio
  - hardware de segmentación que indique si el segmento está presente en memoria o no de manaera que:
    - Se referencia un segmento que está en memoria: se accede normalmente
    - Se referencia un segmento que no está en memoria: se produce una excepción
- EL Intel 286 tenia hardware de segmentación pero no de paginación: IBM OS/2, hasta la versión 1.3, porporcionaba memoria virtual con segmentación bajo demanda

Segmentación bajo demanda: algoritmo de reemplazo

- Los algoritmos de reemplazo son similares a los de lá paginación bajo demanda. Veamos como lo hacía OS/2
- El sistema mantenia una lista de los segmentos en memoria. Periodicamente el sistema
  - Colocaba los segmentos accedidos la final de la lista
  - Limpiaba los bits de acceso
- Cuando habia que reemplazar un segmento se reemplazaba, en caso de ser necesario, el primero (o los primeros) de la lista: se trata de una aproximanción a LRU pues la lista está *"ordenada"* por tiempo de acceso

Segmentación bajo demanda: reemplazo de segmentos

- El mecanismo de reemplazo es un poco distinto del de la paginación bajo demanda, pues los distintos segmentos tienen distinto tamaño. Veamos como lo hacía OS/2
- Al producirse un fallo de segmento
  1. Se mira si ha espacio suficiente en memoria, y si lo hay se hace una compactación
  2. Si no hay espacio en memoria se coge el primer segmento de la lista y si es necesario lo escribe en el intercambio
     - Si ahora hay espacio suficiente, se carga el segmento en memoria, se actualiza la tabla de segmentos y se pone al final de la lista
     - Si no hay espacio se vuelve al paso 1

Otras consideraciones

- Hasta ahora hemos visto los siguientes elementos que influyen en el rendimiento de la páginación bajo demanda
  - El algoritmo de reemplazo de páginas
  - El tipo de reemplazo: local o global
  - La asignación de marcos a un proceso: fija o variable
- Tambien influye el tamaño de página y el acceso a los datos en memoria
  - páginas pequeñas: se adaptan mejor a la localidad del programa
  - páginas grandes: simplifican la contabilidad y optimizan las transferencias

Otras consideraciones: colocación de los datos en memoria

- La paginación bajo demanda es transparente para el usuario y el programador, de manera que, en principio, no tienen por que ser conscientes de su existencia
- Sin embargo, si tenemos en cuenta que en C las matrices se almacenan por filas, los dos programas que se muestran a continuación muestran resultados muy distintos en la misma máquina debido a a la distinta manera de recorrer la memoria y por tanto al distinto número de fallos de página que provocan.

```
$ time ./p1
real 0m1.897s
user 0m1.236s
sys 0m0.644s
$time ./p2
real 0m17.966s
user 0m16.905s
sys 0m0.912s
```

```
#include <stdlib.h>

#define NCOLS 1024*16
#define NFILAS 1024*16

main()
{
  int **a;
  unsigned  i, j;

 a=(int**) malloc (NFILAS * sizeof (int*));
 for (i=0; i<NFILAS;i++)
   a[i]=(int *) malloc (NCOLS*sizeof(int));


   for (i=0; i<NFILAS; i++)
     for (j=0; j<NCOLS; j++)
        a[i][j]=23;

}
```

```
#include <stdlib.h>

#define NCOLS 1024*16
#define NFILAS 1024*16

main()
{
  int **a;
  unsigned  i, j;

 a=(int**) malloc (NFILAS * sizeof (int*));
 for (i=0; i<NFILAS;i++)
   a[i]=(int *) malloc (NCOLS*sizeof(int));

   for (j=0; j<NCOLS; j++)
     for (i=0; i<NFILAS; i++)
        a[i][j]=23;

}
```