Arquitectura del Computador
y
Sistemas Operativos

Vigésimotercera Clase

Manejo de Memoria (1/2)

El SO debe llevar un registro de la memoria total y qué bloques entregó a cada programa.

Hay dos formas en que los Sistemas Operativos hacen ésto:

Bitmaps	Ventajas	• Ocupa poco espacio (poco overhead)
	Desventajas	• Lento para buscar un bloque de n unidades
Listas	Ventajas	• Múltiples algoritmos de búsqueda con buena performance
	Desventajas	• Algunos formatos pueden ser más lentos al liberar.

Dentro de las listas hay varias opciones. Se puede mantener una lista única ordenada por dirección donde cada elemento sea un bloque libre (Hole) o uno ocupado (Process).

Otras implementaciones utilizan dos listas, uno de bloques libres y otro de procesos. Esta alternativa es más rápida para alocar pero más lenta al liberar.

Manejo de Memoria (2/2)

Como se dijo antes, dentro de la alternativa de listas, hay múltiples formas de buscar un lugar libre. Entre ellas:

First Fit	Se barre una lista simplemente encadenada buscando el primer bloque ≥ al necesario
Next Fit	Se barre una lista circular simplemente encadenada buscando el primer bloque ≥ al necesario
Best Fit	Se barre la lista completa buscando el menor bloque ≥ al necesario
Worst Fit	Se barre la lista completa buscando el mayor bloque ≥ al necesario
Quick Fit	Se generan varias listas de bloques libres de distintos tamaños que a su vez están ordenadas por tamaño. Si bien es rápida para alocar es muy lenta para liberar.

Varias de ellas involucran el mantenimiento de varias listas en vez de una sola.

Virtual Memory

Como vimos el SO puede tener en ejecución procesos cuya suma de la memoria de los segmentos es mayor a la memoria del sistema (manteniendo algunos segmentos en disco) en lo que llamamos *Memory Swapping*.

Sin embargo, si un segmento quisiera seguir creciendo y llegara a no entrar en memoria el SO ya no tendría otra opción más que terminar la tarea con error de falta de memoria.

El concepto de memoria virtual (o Virtual Memory) es un enfoque distinto:

- Todos los procesos ven un *enorme* bloque de memoria que ocupa toda la RAM disponible (o incluso más)
- Este enorme bloque está dividido en páginas relativamente pequeñas. Cada una de estas páginas puede estar:
  - No alocada: El proceso nunca la solicitó
  - Alocada presente: Contiene datos del proceso y está físicamente en RAM.
  - Alocada ausente: Contiene datos del proceso, pero no está en RAM dado que por falta de memoria se la bajó a disco.

Paginado (1/8)

Definición

Este nuevo enfoque se denomina paginado (o *Paging*). El problema es que cuando hablábamos de *segmentos*, como eran relativamente grandes, su información se podía mantener en registros del procesador.

Ahora las páginas son pequeñas y por ende hay muchas. El SO debe mantener grandes tablas con los datos de cada página en las llamadas tablas de páginas, o "*Page Tables*".

Por la cantidad de entradas en las tablas, la información de estas tablas no puede mantenerse en registros, sino que deben estar en memoria.

Acceder a memoria para consultar la tabla con cada acceso a memoria sería inaceptablemente lento. La tabla, o al menos la parte más usada de ella, requiere ser mantenida en un cache dedicado y muy rápido para que este enfoque sea viable.

Este cache se denomina TLB (*Translation Lookaside Buffer*).

Paginado (2/8)

TLB (1/2)

El primer sistema que utilizó paginado (y por ende TLB) fue Multics, de ahí en adelante se incluyó en múltiples procesadores.

Por ejemplo, los procesadores Intel i3, i5 e i7 Sandy Bridge tienen, cuando trabajan con páginas de 4Kb:

TLB nivel 1:

- Segmentos de código: 128 entradas, 4-way
- Segmentos de datos: 64 entradas, 4-way

TLB nivel 2:

- Compartido por código y datos: 512 entradas, 4-way

La implementación del paginado sobre TLB se basa en un hecho empírico, que indica que los programas hacen muchos accesos a pocas páginas, de ahí que sea viable.

Independientemente de ello, como con todo cache, siempre existe la posibilidad que al buscar un descriptor el mismo no esté. En ése caso se debe levantar de RAM.

Paginado (3/8)

TLB (2/2)

Los procesadores, a lo largo de la historia, hay usado dos técnicas para para subir al TLB un descriptor inexistente:

- Hardware: El procesador tiene la información para levantar el descriptor de RAM en forma autónoma. Este autonomía debe ser tal que le permita identificar dónde está y luego cargarlo.
- Software: Al detectarse que falta se ejecuta una interrupción para que el SO lo cargue.

Los procesadores de la familia del x86 tienen hardware para levantar un descriptor que no está en el TLB.

Paginado (4/8)

Soporte de Hardware

Al igual que los segmentos, el hardware apoya al SO en su tarea. El sistema de paginado en el x86 tiene los siguientes agregados:

- (A) Accessed bit: El procesador lo enciende para indicar que se utilizó la página. Ayuda al SO a determinar qué páginas están en uso.
- (P) Present: Le indica al SO que la tarea está tratando de acceder a una página que no está en RAM. Le permite al SO detener la tarea, levantar la página y luego volver a transferirle el control. La tarea “nunca se enteró” que la página faltaba y se la detuvo.
- (M) Modified: Este bit lo enciende el SO cuando se graba en la página. Si la página se levantó de disco, se leyó de la misma pero no se la modificó, se la puede sacar de RAM sin tener que volver a grabarla en disco.

Cuando se trata de acceder a una página que no está en RAM (P=0), se genera una interrupción al SO que llamamos *Page Fault*.

Paginado (5/8)

Paginado en i386 (1/2)

Ante todo, en la familia x86, el segmentado está siempre encendido y el paginado puede activarse o no.

La dirección lógica pasa por el sistema de segmentado y genera lo que se llama dirección lineal (*linear address*). Si el paginado está desactivado la dirección lineal coincide con la física. Al activar el paginado, la dirección lineal es transformada nuevamente por el sistema de paginado para generar la dirección física.

Activar el paginado bajo el segmentado permite que el SO "*haga lugar*" sin tener que copiar la memoria, simplemente utiliza el sistema de paginado para hacer el desplazamiento.

Paginado (6/8)

Paginado en i386 (2/2)

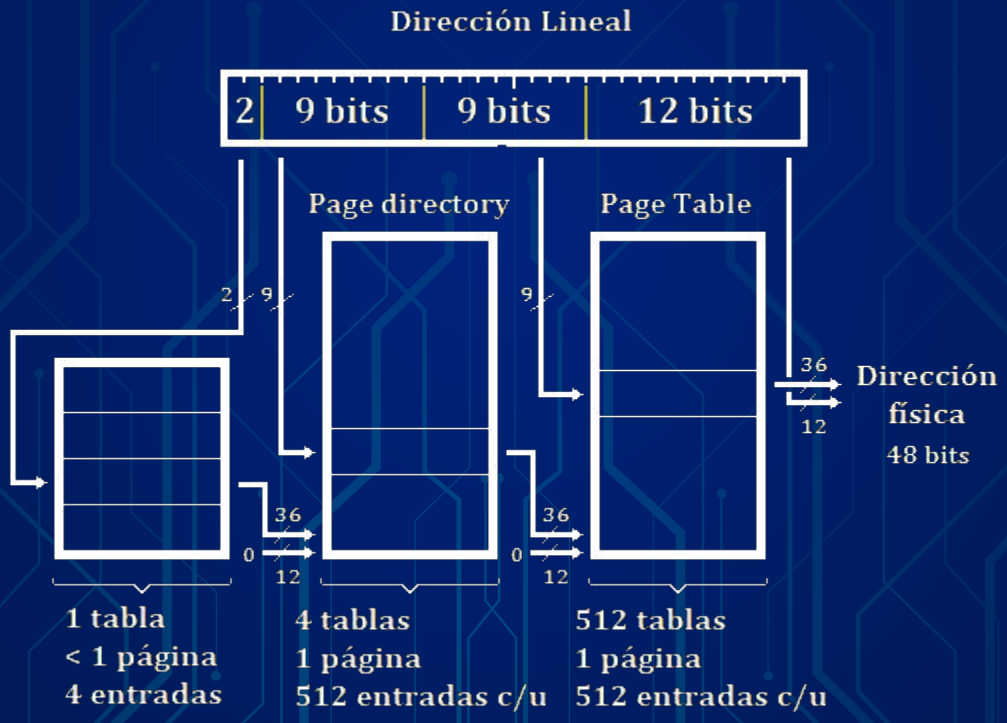
Paginado de 32 bits en dos niveles:

10 + 10 + 12:

Paginado (7/8)

Paginado en Pentium Pro

Pentium Pro: Paginado de dos niveles: 2 + 9 + 9 + 12:



Paginado (8/8)

Paginado en x86-64

Paginado de cuatro niveles: 9 + 9 + 9 + 12:

48 bits logical → 48 bits physical (256 Tb)

Paginado de cinco niveles: 9 + 9 + 9 + 9 + 12:

57 bits logical → 57 bits physical (128 Pb)

Ejercicios (1/2)

Considere el siguiente contexto:

Registros:

```
CS      0003
LDTR    0001 0000
```

Memory Dump:

```
0001 0000    00 02 00 00 F0 70 00 00
0001 0008    00 01 12 34 56 F0 00 00
```

¿Qué sucede al ejecutar las siguientes instrucciones asumiendo que el procesador i386 ejecuta en modo protegido con paginado deshabilitado?

a)
```
MOV     DS,0x000F
MOV     [0x10],EAX
```

b)
```
MOV  DS,0x0007
MOV  [0xF000],EAX
```

c)
```
MOV     DS,0x0007
MOV     [0x10],EAX
```

d)
```
MOV  DS,0x0000
MOV  [0x10],EAX
```

Ejercicios (2/2)

Considere los siguientes agregados al contexto anterior:

Registros: `CR3 0002 0000`

Memory Dump:

```
0002 0000    00 00 00 00
0002 0004    00 03 00 05

0003 018C    22 33 40 05
```

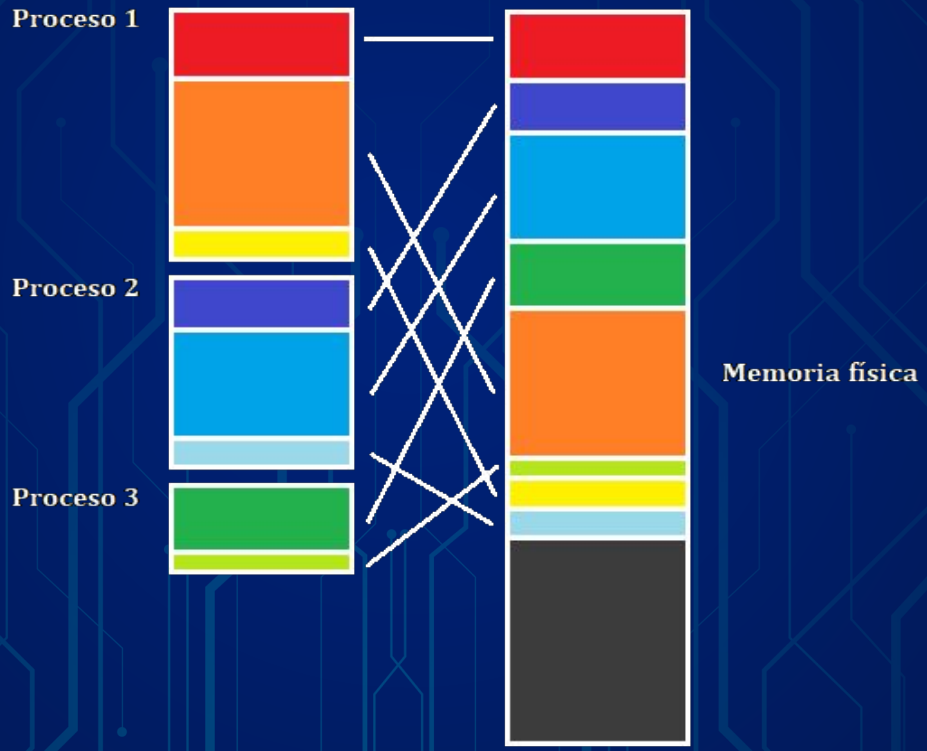
Si ahora el paginado está habilitado, ¿Qué sucede al ejecutar las siguientes instrucciones?

```
MOV     DS,0x000F
MOV     [0x10],EAX
```

SO

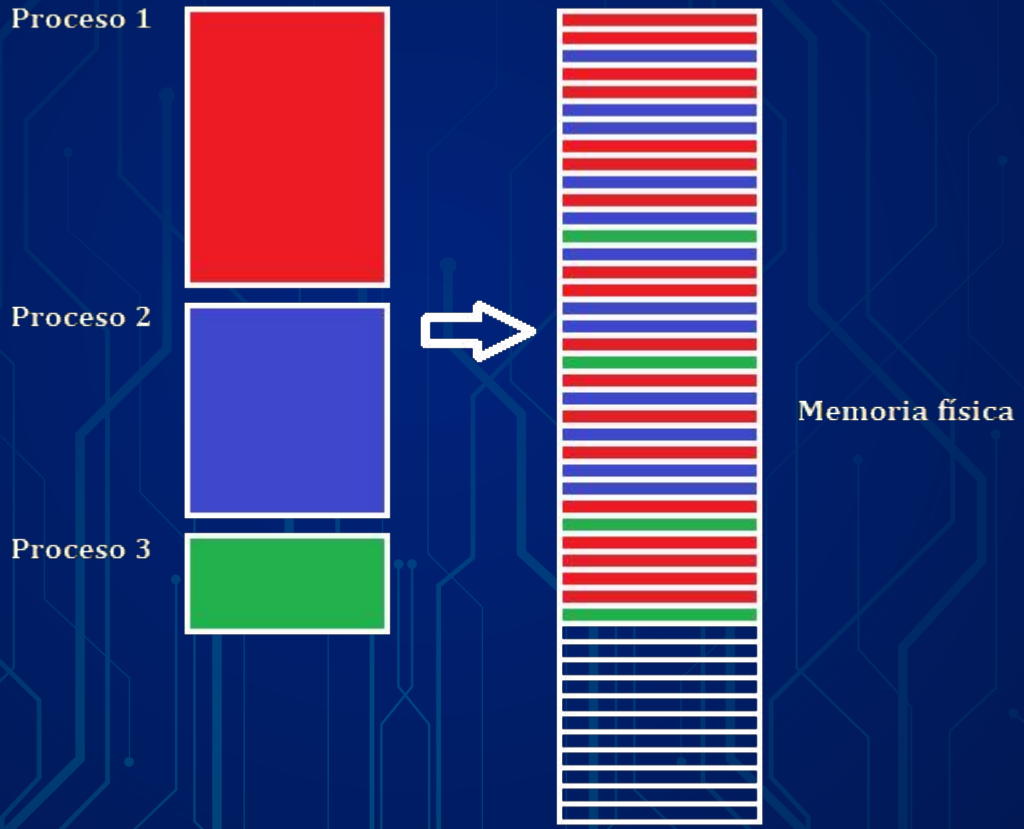
Resúmen de Segmentado

Así ven los procesos cuando se les asigna memoria segmentada:



Resúmen de Paginado (Memoria Virtual)

Así ven los procesos cuando el SO les asigna memoria virtual:



Fin

¿Preguntas?