FSO - ejercicios de gestión de memoria

Esta es una lista de escenarios de uso de memoria paginada, en la que te planteamos algunos ejercicios que manejan los espacios de direcciones lógico y físico y la arquitectura de traducción de direcciones. Los objetivos que pretendemos con estos ejercicios son ayudarte a:

1. Tomar consciencia de las diferencias entre las direcciones físicas y lógicas.
2. Entender la organización de un sistema de paginación jerárquica.
3. Conocer el impacto de la utilización de una TLB en el tiempo de acceso a memoria.

Caso 1: direcciones de memoria paginada

Para empezar, supongamos un sistema que utiliza memoria paginada. Las direcciones lógicas son de 24 bits. Si se dedican 10 bits para seleccionar la página y los restantes para el desplazamiento dentro de la página:

- ¿Cuántas páginas puede llegar a tener un proceso?
- ¿Qué tamaño de página utiliza el sistema?
- ¿Cuánta memoria puede direccionar un proceso?

Pongamos otro caso similar. Esta vez tenemos direcciones lógicas de 24 bits y sabemos que el tamaño de página es de 4KiB. Entonces:

- ¿Cuántas páginas puede llegar a tener un proceso?
- ¿Cuántos bits hay que dedicar a página y a desplazamiento?
- ¿Cuánta memoria puede direccionar un proceso?

Otra cuestión interesante es observar la influencia entre el tamaño de página y el espacio ocupado por la **tabla de páginas**. ¿Cuál de los dos sistemas anteriores genera tablas de páginas más grandes?

Caso 2: espacio físico y espacio lógico diferentes

NOTA: este tópico no está lo bastante resaltado en la bibliografía, así que lo describimos con un ejemplo a continuación, para que te sirva de ilustración.

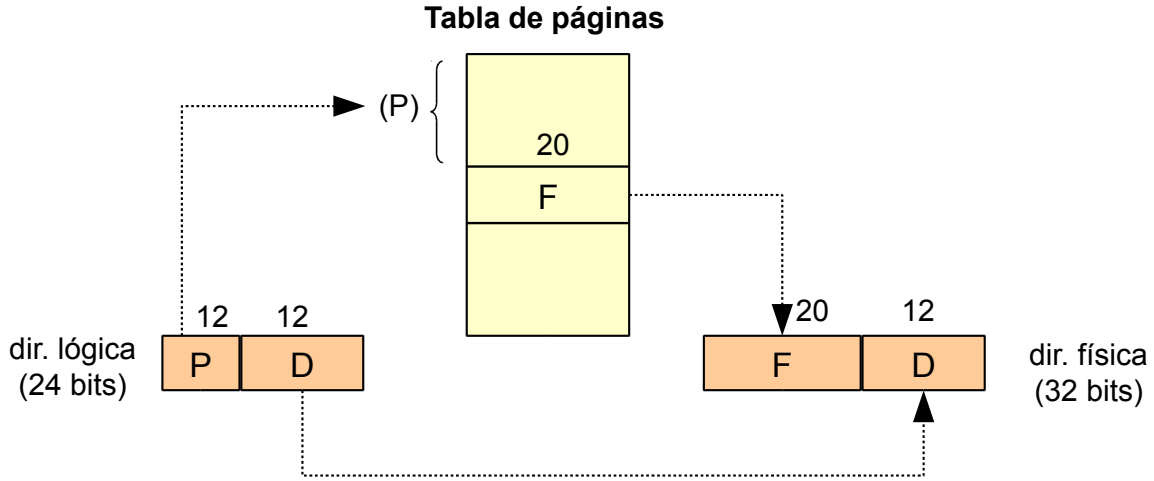
Una cuestión muy importante para entender el hardware de gestión de memoria es que el **espacio lógico** y el **espacio físico** de direcciones pueden tener tamaños diferentes.

Supongamos la máquina descrita en el segundo sistema del Caso 1, que maneja direcciones lógicas de 24 bits y páginas de 4KiB. En este sistema un proceso dispone de un espacio lógico de hasta 16 megabytes, pero esto no implica forzosamente que el espacio físico direccionable deba ser de 16MiB. Por ejemplo, supongamos que en esta máquina el bus que accede a la memoria física es de

32 bits, o lo que es lo mismo, las direcciones físicas son de 32 bits. Eso significa que el *hardware* puede manejar una memoria física de hasta 2^{32} = 4GiB, aunque un proceso sólo puede manejar 16MiB de espacio lógico.

¿Tiene sentido que un proceso sólo pueda acceder a una fracción del espacio físico? Pues sí: históricamente ha ocurrido bastantes veces que las tecnologías de RAM se abaratan y permiten más capacidad (ej. pasar de decenas de megas a centenares o miles de megas), pero la arquitectura de la CPU no se puede ampliar tan fácilmente, porque hay mucho código ya escrito y no se pueden estar reescribiendo/recompilando/probando todas las aplicaciones para un formato de instrucciones distinto. Por ello es planteable que, durante unos años, la CPU maneje un espacio lógico antiguo y más pequeño que el físico.

Si la CPU utiliza direcciones lógicas de 24 bits, ¿cómo podemos aprovechar que hay más memoria física? Antes de responder a esta cuestión, veamos cómo quedaría el circuito de traducción de direcciones en esta máquina de 24/32 bits:



P = página lógica; D = desplazamiento; F = marco físico.
Los números indican el tamaño en bits de cada elemento.

Mediante el mecanismo descrito en la figura, cada página lógica de un proceso puede residir en *cualquier* marco de la memoria física. La tabla de páginas guarda la correspondencia entre las páginas físicas y los marcos lógicos. Es bueno también recordar que el tamaño de página es necesariamente el mismo en las direcciones lógicas y físicas, porque la página es precisamente la unidad de asignación de espacio físico.

Si nuestro sistema operativo es multiprogramado, cada proceso puede tener su espacio en zonas diferentes de la memoria, resuelto gracias a su propia tabla de páginas (recordemos que cada proceso tiene una tabla de páginas diferente). Así pues, si tenemos N procesos en ejecución, en

conjunto pueden estar manejando N×16 megas de memoria física, aunque ningún proceso individual puede direccionar simultáneamente más de 16 megas. Además, el núcleo del sistema operativo podría tener su propia tabla de páginas, disponiendo así de hasta 16 megas de memoria física que no restarían capacidad al espacio lógico de los procesos de usuario.

Esta diferencia de tamaños entre el espacio lógico y el espacio físico puede ocurrir en el otro sentido: el espacio lógico podría ser más grande que el físico. Sin ir más lejos, los actuales procesadores x86 manejan un espacio lógico de 64 bits, pero utilizan direcciones físicas de 48 bits. Esto se debe a que actualmente todavía no es rentable fabricar procesadores con buses físicos de 64 bits.

Ejercicio para el caso 2

Supongamos una máquina con direcciones lógicas de 16 bits, direcciones físicas de 32 bits y un tamaño de página de 16KiB. En este sistema:

- ¿Cuánta memoria puede direccionar un proceso?
- ¿Cuánto espacio físico puede direccionarse?
- ¿Cuántos marcos de página pueden llegar a existir en la memoria?
- ¿Cuántos bits dedicados a página y desplazamiento habría en una dirección física?
- ¿Cuántas entradas puede llegar a tener la tabla de páginas de un proceso?
- ¿Cuántos bits son necesarios para almacenar una entrada en la tabla de páginas?

Caso 3: tamaño de la tabla de páginas

En los sistemas comerciales se han manejado tamaños de páginas entre 256 bytes y hasta 1 gigabyte. Un tamaño típico y clásico son los 4KiB de los procesadores x86 de Intel, que vamos a tomar como caso para este ejercicio. En los x86 el tamaño de una entrada en la tabla de páginas es de 32 bits (4 bytes).

Bajo estas condiciones, supón que tenemos un proceso que tiene reservados 10 megabytes de memoria y calcula estos datos:

- ¿Cuántas páginas necesita este proceso?
- ¿Cuánto ocupa la tabla de páginas de este proceso?

Caso 4: paginación jerárquica en los Intel x86

En el supuesto del Caso 3, el tamaño de la tabla de páginas del proceso es superior al tamaño de una página. Lo diremos con otras palabras: *una tabla de páginas no cabe en una sola página.* Y para que funcione correctamente el circuito de traducción de la MMU, la tabla de páginas debe estar almacenada de forma contigua en la memoria física. Esto significa que para poder ejecutar un proceso de cierto tamaño, debemos buscar varios marcos libres consecutivos en la memoria física. Y esto es justamente lo que queríamos evitar con la paginación. Si necesitamos bloques contiguos de memoria, vamos a tener el problema de la *fragmentación*. Este problema se acentúa cuanto más grande es el proceso y mayor es su tabla de páginas.

Una solución a este problema de la tabla de páginas es la **paginación jerárquica**, que consiste en *paginar la tabla de páginas*. El Intel 80386 y sus sucesores de 32 bits utilizan un esquema de dos niveles de paginación, con un primer nivel llamado *directorio de páginas*, que apunta a los diferentes fragmentos de la tabla de páginas. Este es el modelo:

10 bits	10 bits	12 bits
1er nivel de paginación	2º nivel de paginación	desplazamiento

Las entradas de las tablas de páginas son siempre de 4 bytes (la de primer nivel y las del segundo nivel).

Sobre este modelo de paginación, contesta las siguientes preguntas:

- ¿Cuántas páginas puede llegar a tener un proceso en total?
- ¿Cuántas entradas puede llegar a tener el directorio de páginas de un proceso? *(la tabla de primer nivel)*
- ¿Qué tamaño en bytes puede alcanzar el directorio de primer nivel?
- ¿Cuántas entradas puede llegar a tener una tabla de páginas de segundo nivel?
- ¿Qué tamaño en bytes puede llegar a tener una tabla de segundo nivel?

Imagina un proceso que ocupa 100 megabytes. ¿Cuánto espacio consumirán sus tablas de páginas?

Una vez que has obtenido los resultados, podrás ver que los ingenieros de Intel seleccionaron esta organización del espacio lógico [10+10+12] de forma muy intencionada. Una forma de verlo es contestar las mismas preguntas anteriores, pero con una organización [12+8+12] o con una organización [8+12+12]. ¿Ves alguna ventaja del [10+10+12] de Intel sobre esas dos alternativas?

Caso 5: TLB y tiempo de acceso a la memoria

Un problema que tiene la memoria paginada es que penaliza fuertemente el tiempo de acceso a memoria. Para traducir una dirección lógica hay que acceder a una tabla que está almacenada en la RAM. Por tanto, en el mejor de los casos **se duplica** el tiempo de acceso efectivo a la memoria. Y si se utiliza un sistema de paginación jerárquica con varios niveles, el tiempo se multiplica aún más: un sistema con dos niveles de tablas requiere tres accesos a memoria física para resolver un acceso a una dirección lógica.

La solución que se ha dado a este problema es disponer de una **caché** dentro de la CPU con aquellas entradas de la tabla de páginas que se estén utilizando con más frecuencia. Esta caché se llama **TLB** (*translation lookaside buffer*). Si se consiguen mantener en la TLB las entradas más probables, se pueden obtener *tasas de aciertos* superiores al 96-98%. Gracias a ello disminuye notablemente el tiempo efectivo de acceso a la memoria y la paginación puede seguir siendo una técnica viable.

*NOTA: entendemos por **tiempo de acceso efectivo** a la memoria como el tiempo total que se invierte en completar un acceso, desde que la CPU genera una dirección lógica hasta que se realiza la operación en la RAM. O sea, se incluye el tiempo de traducción de la dirección más todos los accesos reales que hay que hacer a la RAM para completar la operación.*

Para analizar el impacto del uso de una TLB, vamos a plantear este ejercicio: tenemos un sistema de gestión de memoria paginada, de un solo nivel, que utiliza una TLB. Se tienen los siguientes tiempos medios:

- Tiempo de acceso a la TLB: 4 nanosegundos.
- Tiempo de acceso a la RAM: 30 nanosegundos.
- Tasa de aciertos observada en la TLB: 98%.

¿Cuál sería el tiempo medio de acceso efectivo a memoria? ¿En cuánto se diferencia este tiempo medio del tiempo de acceso si no usáramos paginación?

Ahora cambiemos el escenario y supongamos que tenemos un sistema de paginación jerárquica como el de los x86 (dos niveles de paginación). Si se mantiene el resto de los valores iguales que en el ejercicio anterior, ¿cuál sería ahora el tiempo medio de acceso efectivo a memoria?