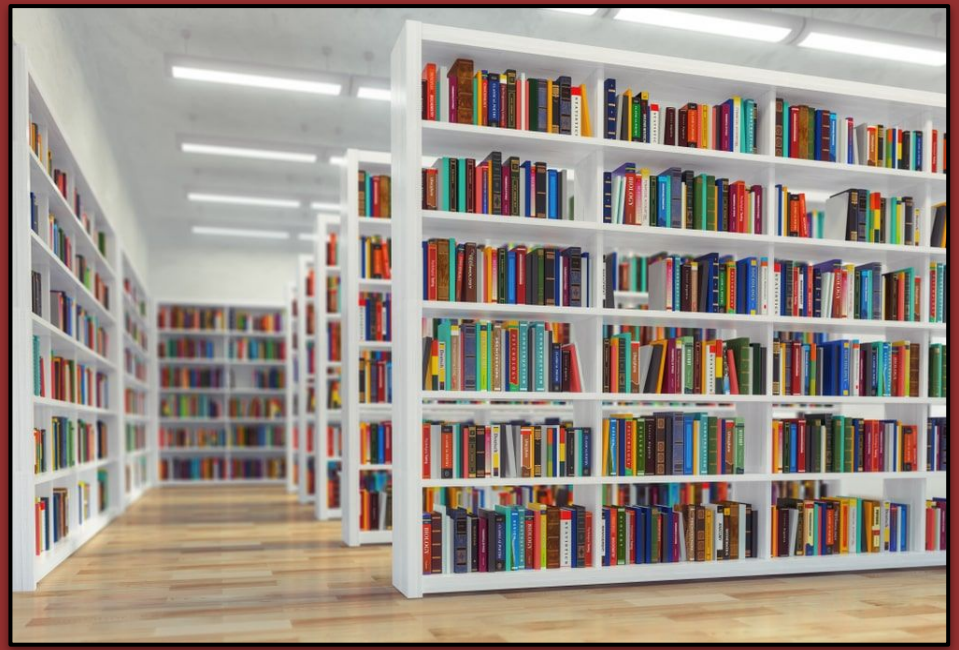
- Tema 3.1 -

GESTIÓN DE LA MEMORIA: PAGINACIÓN Y SEGMENTACIÓN



Índice de contenidos

1. Requisitos de la gestión de la memoria.
2. Particionamiento de la memoria.
3. Paginación.
4. Segmentación.

Requisitos de la gestión de la memoria

Requisitos de la gestión de la memoria

En un sistema monoprogramado, la memoria se divide en dos partes:

- Una parte para el sistema operativo (núcleo).
- Una parte para el programa actualmente en ejecución.

En un sistema multiprogramado, la parte de usuario de la memoria se **debe dividir posteriormente para acomodar múltiples procesos**.

- El S.O. es el encargado de la tarea de subdivisión y a esta tarea se le denomina **gestión de la memoria**.
- Una buena gestión de la memoria es vital en un sistema multiprogramado.

Requisitos de la gestión de la memoria

De forma general, una buena gestión de memoria debe satisfacer los siguientes cinco requisitos:

- Reubicación.
- Protección.
- Compartición.
- Organización lógica.
- Organización física.

1. Reubicación

1. Reubicación

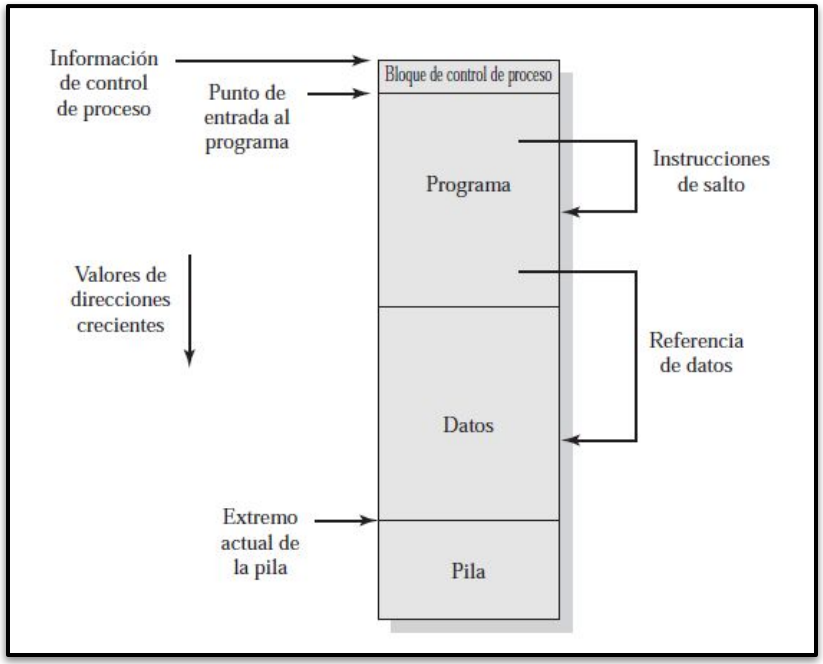
En un sistema multiprogramado…

- La memoria disponible se comparte entre varios procesos.
- Normalmente, no es posible que el programador sepa anticipadamente qué programas residirán en memoria principal.
- Adicionalmente, como vimos en el Bloque 2, resulta útil poder intercambiar procesos en la memoria principal para maximizar la utilización del procesador, proporcionando un mayor número de procesos para ejecución.
- Cuando traemos un proceso de disco a memoria principal, es muy limitante tener que colocarlo en la misma región de la memoria principal en la que se encontraba anteriormente. Por el contrario, podría ser necesario reubicar el proceso a un área de memoria diferente.

Estos aspectos ponen de manifiesto algunos aspectos técnicos relacionados con el direccionamiento.

1. Reubicación

Por razones de simplicidad, asumimos que la imagen de un proceso ocupa una región contigua de la memoria principal:



1. Reubicación

El S.O. necesita conocer la ubicación de la información de control del proceso y de la pila de ejecución, así como el punto de entrada que utilizará el proceso para iniciar la ejecución.

- Debido a que el S.O. se encarga de gestionar la memoria y es responsable de traer el proceso a la memoria principal, estas direcciones son fáciles de adquirir.

Sin embargo, de forma adicional, el procesador debe tratar con referencias de memoria dentro del propio programa.

- Las instrucciones de salto contienen una dirección para referenciar la instrucción que se va a ejecutar a continuación.
- Las instrucciones de referencia de los datos contienen la dirección del byte o palabra de datos referenciados.

De alguna forma, el hardware del procesador y el S.O. deben poder traducir las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas, que se reflejan en la ubicación actual del programa en la memoria principal.

2. Protección

2. Protección

Cada proceso debe protegerse contra interferencias no deseadas por parte de otros procesos, sean estas accidentales o malintencionadas.

- Por tanto, los programas de otros procesos no deben ser capaces de referenciar sin permiso posiciones de memoria de un proceso, tanto en modo lectura como en modo escritura.

Lograr los requisitos de la reubicación comentados anteriormente incrementa la dificultad de satisfacer los requisitos de protección.

- Más aún, la mayoría de los lenguajes de programación permite el cálculo dinámico de direcciones en tiempo de ejecución (por ejemplo, un puntero a una estructura de datos).
- Por tanto, todas las referencias de memoria generadas por un proceso deben comprobarse en tiempo de ejecución para poder asegurar que se refieren sólo al espacio de memoria asignado a dicho proceso.

Veremos que los mecanismos que dan soporte a la reasignación también dan soporte al requisito de protección.

2. Protección

Normalmente, un proceso de usuario no puede acceder a cualquier porción del S.O. ni al código ni a los datos.

Un programa de un proceso no puede saltar a una instrucción de otro proceso.

Sin un trato especial, un programa de un proceso no puede acceder al área de datos de otro proceso.

El procesador debe ser capaz de abordar tales instrucciones en el punto de ejecución.

Los requisitos de protección de memoria deben ser satisfechos por el procesador (hardware) en lugar de por el S.O. (software), debido a que el S.O. no puede anticipar todas las referencias de memoria que un programa hará.

Por tanto, sólo es posible evaluar la permisibilidad de una referencia (acceso a datos o salto) en tiempo de ejecución de la instrucción que realiza dicha referencia. Para poder hacer esto, **el hardware del procesador debe tener dicha capacidad**.

3. Compartición

3. Compartición

Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal.

- Por ejemplo: si varios programas están ejecutando el mismo código, es ventajoso permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada.
- Otro ejemplo: procesos que estén cooperando en la misma tarea podrían necesitar coordinar compartir el acceso a la misma estructura de datos.

Conclusión: el sistema de gestión de la memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial. Veremos que los mecanismos usados para dar soporte a la reubicación también soportan capacidades para la compartición.

4. Organización lógica

4. Organización lógica

La memoria principal de un computador se organiza como un espacio de almacenamiento lineal compuesto por una secuencia de bytes o palabras.

- A nivel físico, la memoria secundaria está organizada de forma similar.

Esta organización es similar al hardware real del computador, pero **no se corresponde a la forma en la que los programas se construyen normalmente**.

- La mayoría de los programas se organizan en módulos, algunos de los cuáles no se pueden modificar (sólo lectura, sólo ejecución) y algunos de los cuales contienen datos que sí se pueden modificar.

4. Organización lógica

Si el S.O. y el hardware del computador pueden tratar de forma efectiva los programas de usuarios y los datos en la forma de módulos de algún tipo, entonces se pueden lograr algunas ventajas:

- Los módulos se pueden escribir y compilar independientemente, con todas las referencias de un módulo desde otro resultas por el sistema en tiempo de ejecución.
- Con una pequeña sobrecarga adicional, se pueden proporcionar diferentes grados de protección a los módulos (sólo lectura, sólo ejecución).
- Es posible introducir mecanismos por los cuáles los módulos se pueden compartir entre los procesos. La ventaja de proporcionar compartición a nivel de módulo es que se corresponde con la forma en la que el usuario ve el problema y, por tanto, es fácil para este especificar la compartición deseada.

La herramienta que más adecuadamente satisface estos requisitos es la **segmentación**, que es una de las técnicas de gestión de la memoria que vamos a comentar en este tema.

5. Organización física

5. Organización física

Como comentamos en el Bloque 1 de la asignatura, la memoria de un computador se almacena en al menos dos niveles: memoria principal y memoria secundaria.

- La memoria principal proporciona acceso rápido a un coste relativamente alto. Adicionalmente, esta memoria es volátil. Por tanto, la memoria principal contiene programas y datos actualmente en uso.
- La memoria secundaria es más lenta y más barata que la memoria principal y no es volátil. Por tanto, la memoria secundaria proporciona almacenamiento para programas y datos a largo plazo.

En este esquema de dos niveles, **la organización del flujo de información** entre la memoria principal y la secundaria supone una de las preocupaciones principales del sistema.

5. Organización física

La responsabilidad para la programación de este flujo podría asignarse a cada programador en particular, pero esto no es deseable ni practicable por dos motivos:

- La memoria principal disponible para un programa más sus datos podría ser insuficiente. En estos casos, el programador tendría que usar una técnica conocida como **superposición** (***overlaying***), en la cual los programas y los datos se organizan de tal forma que se puede asignar la misma región de memoria a varios módulos, con un programa principal responsable para intercambiar los módulos entre disco y memoria según las necesidades.
- En un entorno multiprogramado, el programador no conoce en tiempo de codificación cuánto espacio estará disponible o dónde se localizará dicho espacio.

Conclusión: la tarea de mover la información entre los dos niveles de la memoria debería ser una responsabilidad del sistema (esta es la tarea esencia de la gestión de la memoria).

Particionamiento de la memoria

Particionamiento de la memoria

La operación principal de la gestión de la memoria es traer los procesos a la memoria principal para que el procesador los pueda ejecutar.

En prácticamente todos los sistemas multiprogramados modernos, esto implica el uso de un sofisticado esquema conocido como **memoria virtual**.

- La memoria virtual está basada en una o ambas de las siguientes técnicas básicas: **segmentación** y **paginación**.

Antes de llegar a eso, tenemos que hablar de algunas técnicas más sencillas que no utilizan memoria virtual.

Particionamiento fijo

Particionamiento fijo

En la mayoría de los esquemas para gestión de la memoria, asumimos que el S.O. ocupa alguna porción fija de la memoria principal y que el resto de la memoria está disponible para otros (múltiples) procesos.

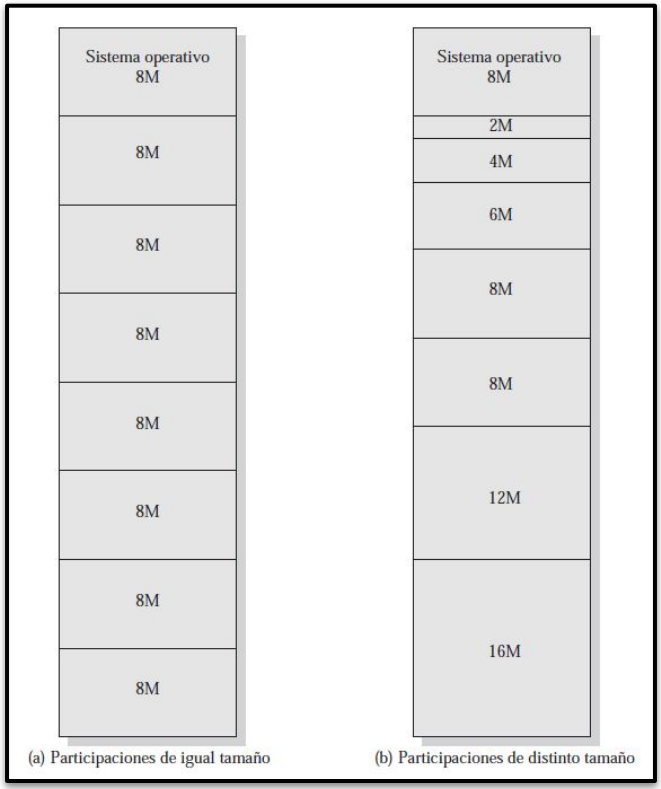
- El esquema **más simple** para gestionar la memoria disponible es **repartirla en regiones con límites fijos**.

¿De qué tamaño hacemos cada partición?

Opción 1: Particiones del mismo tamaño.

- En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de partición puede cargarse en cualquier partición disponible. Si todas las particiones están llenas y no hay ningún proceso en estado Listo o Ejecutando, el S.O puede mandar swap a un proceso de cualquiera de las particiones y cargar otro proceso, de forma que el procesador tenga trabajo que realizar.

Particionamiento fijo



Particionamiento fijo

Existen dos dificultades con el uso de las particiones fijas del mismo tamaño:

- Un programa podría ser demasiado grande para caber en una partición. En este caso, el programador debe diseñar el programa con el uso de *overlays*, de forma que sólo se necesite una porción del programa en memoria principal en un determinado momento. Cuando se necesite un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndolo (*overlaying*) a cualquier progama o datos que haya allí.
- La utilización de la memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupa una participación entera. Por ejemplo: podría haber un programa cuya longitud es menor que 2MB, pero ocuparía una partición de 8 MB cuando se lleva a la memoria. Este fenómeno, en el cual hay espacio interno malgastado debido al hecho de que el bloque de datos cargado es menor que la participación, se conoce con el nombre de **fragmentación interna**.

Ambos problemas se pueden mejorar, aunque no resolver, utilizando particiones de tamaño diferente. En este ejemplo, los programas de 16 MB se pueden acomodar sin *overlays*. Las particiones más pequeñas de 8 MB permiten que los programas más pequeños se puedan acomodar sin menor fragmentación interna.

Particionamiento fijo

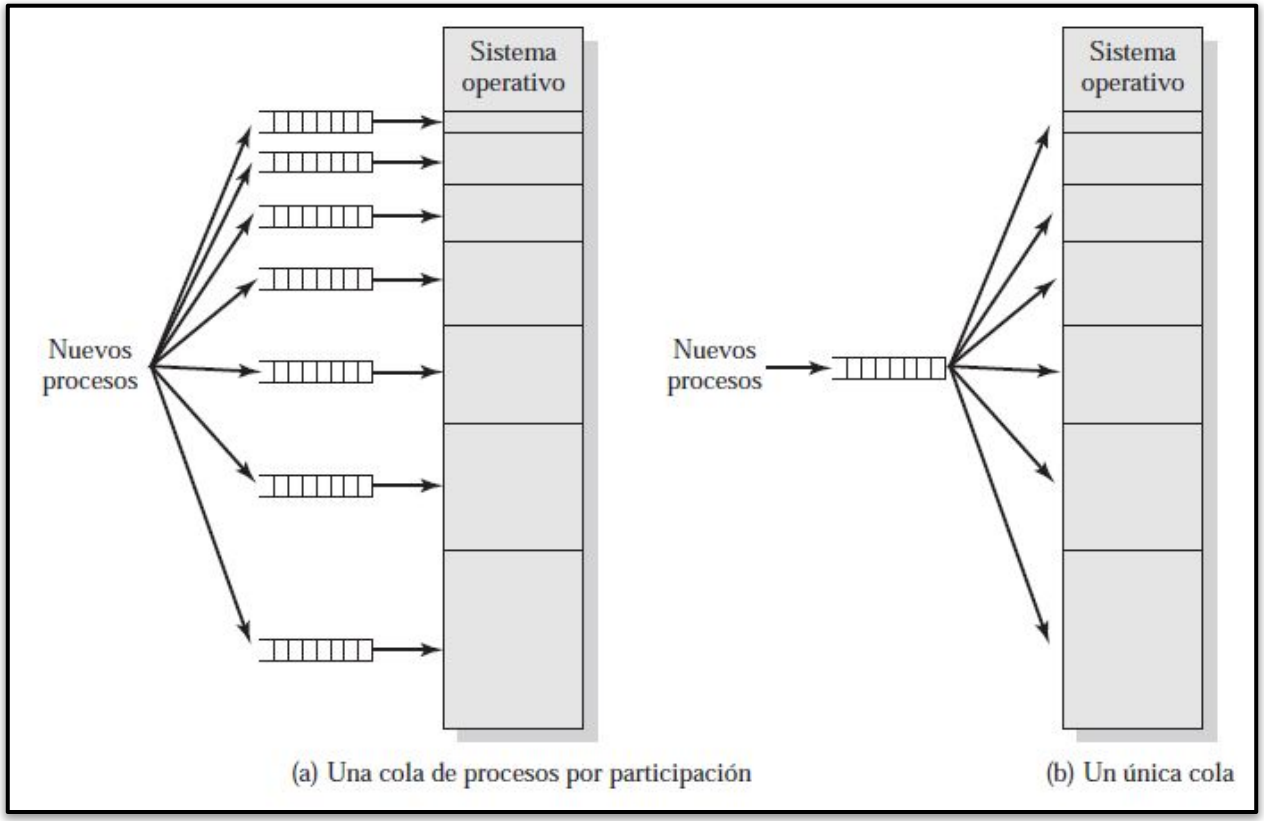
Algoritmo de ubicación

Con particiones **del mismo tamaño**, la ubicación de los procesos en memoria es trivial. En cuanto haya una partición disponible, un proceso se carga en dicha partición. Debido a que todas las particiones son del mismo tamaño, no importa qué partición se utiliza. Si todas las particiones se encuentran ocupadas por procesos que no están listos para ejecutar, entonces uno de dichos procesos debe llevarse a disco para dejar espacio para un nuevo proceso.

Con particiones **de diferente tamaño**, hay dos formas posibles de asignar los procesos a las particiones. La forma más sencilla consiste en asignar cada proceso a la partición más pequeña dentro de la cual cabe: en este caso, necesitamos una cola de planificación para cada partición, que mantenga ordenados los procesos en disco destinados a dicha partición.

Esta técnica no es óptima, ya que podría haber colas de particiones vacías, mientras que en otras podría haber varios procesos esperando por una partición, cuando podrían utilizar una superior: la solución es utilizar una única cola.

Particionamiento fijo



Particionamiento fijo

Algoritmo de ubicación

Si usamos una única cola, en el momento de cargar un proceso en la memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso.

- Si todas las particiones están ocupadas, se debe llevar a cabo una decisión para enviar a *swap* algún proceso.
- Tiene preferencia a la hora de ser expulsado a disco el proceso que ocupe la partición más pequeña que puede albergar el proceso entrante.

El uso del particionamiento fijo está en desuso a día de hoy.

Un ejemplo de S.O. exitoso que sí utilizó esta técnica fue el S.O. de los primeros mainframes de IBM, el sistema operativo conocido como OS/MFT.

Particionamiento dinámico

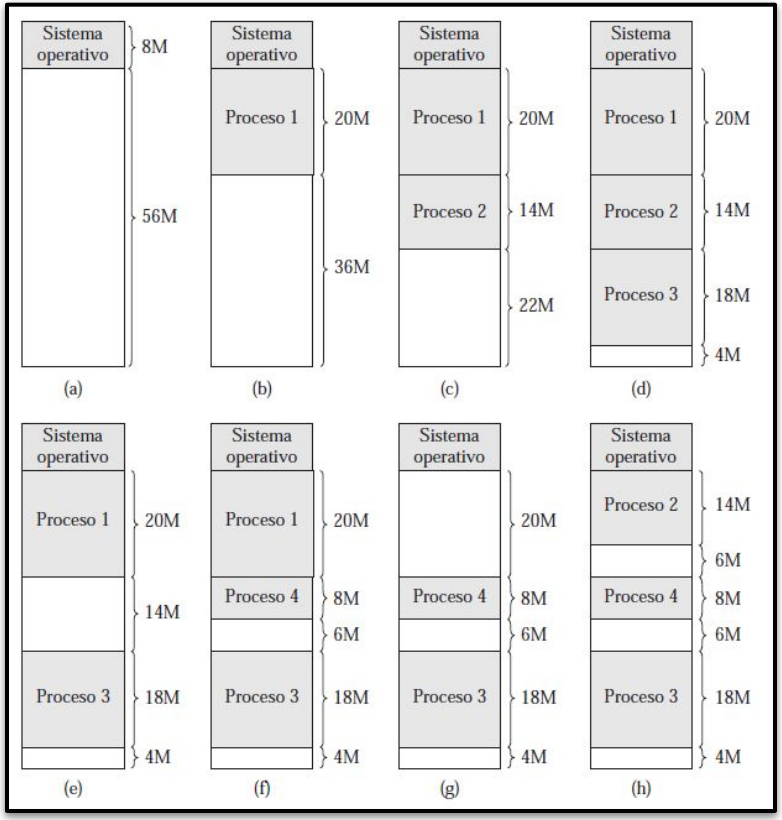
Particionamiento de la memoria

Para mejorar algunas de las dificultades del particionamiento fijo, se desarrolló una técnica conocida como **particionamiento dinámico**.

En el particionamiento dinámico, las particiones son de longitud y número variable.

- Cuando se lleva a memoria principal, se le asigna exactamente tanta memoria como requiera y nada más,

Particionamiento de la memoria



Particionamiento de la memoria

Como muestra el ejemplo anterior, el método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos pequeños en memoria.

- Con el paso del tiempo, la memoria se fragmenta cada vez más y la utilización de la memoria se decrementa. A este fenómeno se le conoce como **fragmentación externa**.

Una técnica para eliminar la fragmentación externa es la **compactación**: de vez en cuando el S.O. desplaza los procesos en memoria, de forma que se encuentren contiguos y, de este modo, toda la memoria libre se encontrará unida en un único bloque.

- La desventaja de la compactación es el hecho de que se trata de un procedimiento que consume tiempo y malgasta tiempo del procesador. → La compactación requiere la capacidad de reubicación dinámica (es decir, debe ser posible mover un programa desde una región a otra en la memoria principal sin invalidar las referencias de la memoria de cada programa.

Particionamiento de la memoria

Algoritmos de ubicación

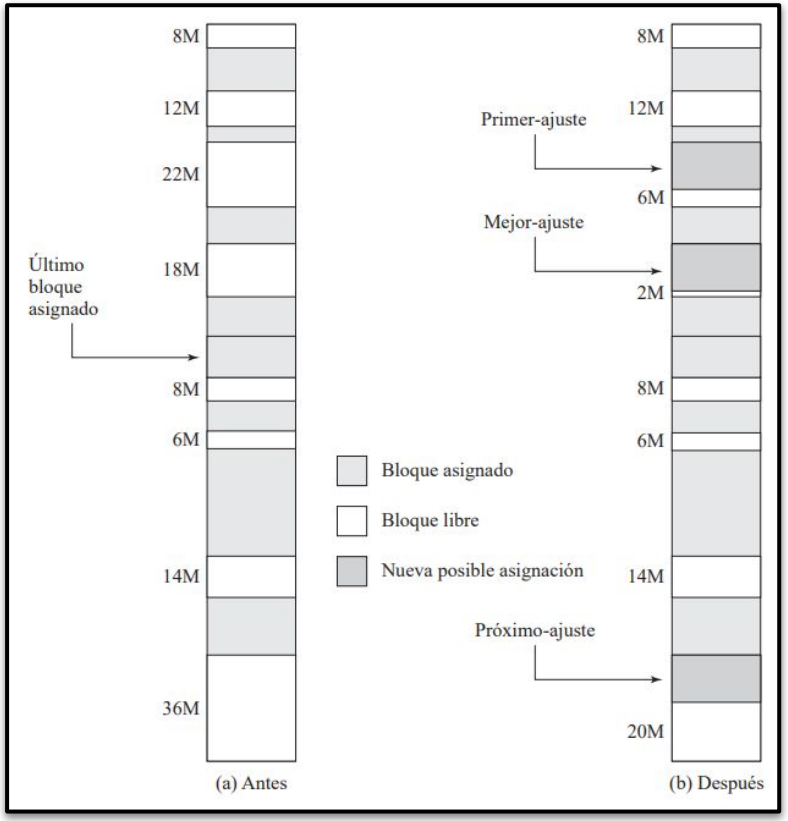
Debido a que la compactación de memoria consume una gran cantidad de tiempo, el diseñador del S.O. debe ser inteligente a la hora de decidir cómo asignar la memoria de los procesos (cómo eliminar los huecos).

A la hora de cargar e intercambiar un proceso a la memoria principal (siempre que haya memoria), el S.O. debe decidir qué bloque libre asignar. Existen tres algoritmos de colocación:

- Mejor ajuste (*best fit*): escoge el bloque de memoria más cercano en tamaño a la petición.
- Primer ajuste (*first fit*): comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea lo suficientemente grande.
- Siguiente ajuste (*next fit*); comienza a analizar la memoria desde la última colocación y escoge el siguiente bloque disponible que sea lo suficientemente grande.

Particionamiento de la memoria

Queremos satisfacer ahora una petición de asignación de 16 MB.



Particionamiento de la memoria

Algoritmos de ubicación

¿Qué algoritmo es mejor?

La respuesta depende de la secuencia exacta de intercambio de procesos y del tamaño de dichos procesos. Sin embargo, podemos hacer algunos comentarios:

- El algoritmo *first fit* no es sólo el más sencillo, sino que habitualmente es también el mejor y el más rápido.
- El algoritmo *next fit* tiende a producir resultados ligeramente peores que el algoritmo *first fit*.
- Generalmente, el algoritmo *best fit* es el que peores resultados ofrece, ya que hace que la memoria principal se quede llena de pequeños huecos inútiles de memoria, lo que hace que la compactación sea más frecuente que el resto de algoritmos.

Particionamiento de la memoria

Algoritmos de reemplazamiento

En un sistema multiprogramado que utiliza particionamiento dinámico, puede haber un momento en el que todos los procesos de la memoria principal estén en estado bloqueado y no haya suficiente memoria para un proceso adicional, incluso después de hacer una compactación.

- Para evitar malgastar tiempo de procesador esperando a que un proceso se desbloquee, el S.O. intercambiará alguno de los procesos entre la memoria principal y el disco, para hacer sitio a un nuevo proceso o para un proceso que esté Listo-Suspendido.

Hay diferentes algoritmos de reemplazo, y generalmente se ven influenciados por el esquema de memoria virtual que esté utilizando la máquina.

Reubicación

Reubicación

Con el particionamiento fijo, se espera que un proceso siempre sea asignado a la misma partición: es decir, sea cual sea la partición seleccionada cuando se carga un nuevo proceso, ésta será la utilizada para el intercambio del proceso entre memoria principal y disco (*swapping*).

- Cuando cargamos un proceso por primera vez en memoria principal, cambiamos todas las referencias de la memoria relativas del código por direcciones de memoria principal absoluta, determinadas por la dirección base de la partición en la que el proceso ha sido cargado.

La realidad es que tanto en muchos de los esquemas de particionamiento fijo (como por ejemplo el de particiones de distinto tamaño) y en el particionamiento dinámico, **un proceso puede ocupar diferentes posiciones durante el transcurso de su ciclo de vida**.

- Adicionalmente, cuando utilizamos la compactación, los procesos son desplazados mientras están en memoria principal.

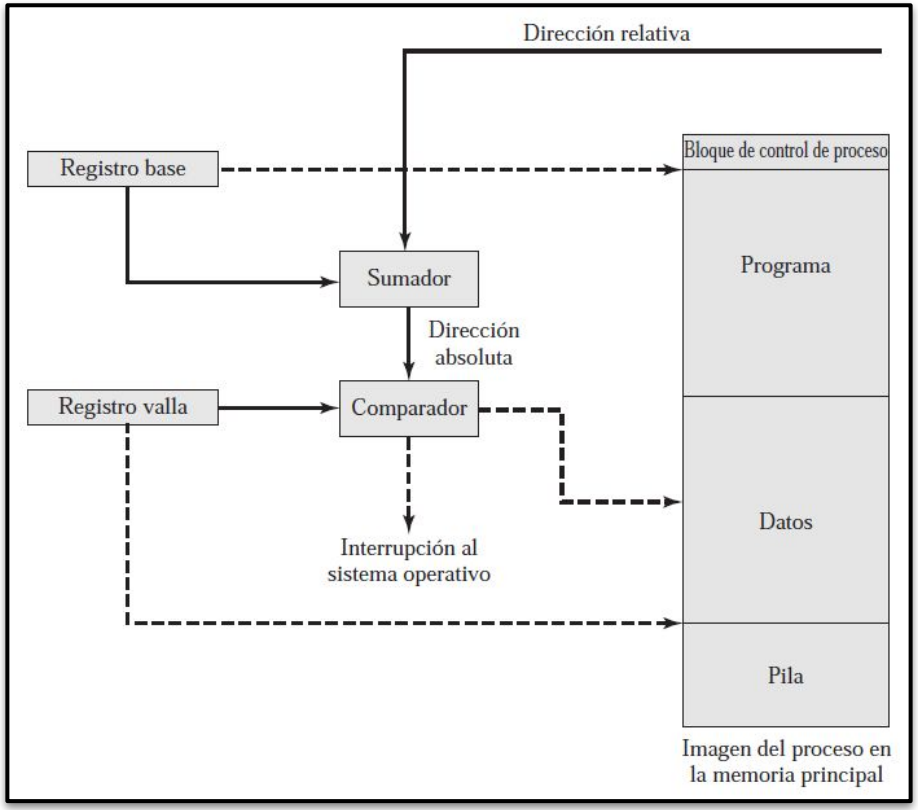
Por tanto, las ubicaciones (de instrucciones y datos) referenciadas por un proceso no son fijas → **¡problemón!**

Reubicación

Para poder resolver este problema, debemos empezar a distinguir entre varios tipos de direcciones:

- Una **dirección lógica** de memoria es una referencia a una ubicación de memoria independiente de la asignación actual de datos de la memoria. Es preciso una traducción a una dirección física antes de que se realice el acceso a memoria.
- Una **dirección relativa** de memoria es un ejemplo particular de dirección lógica, en el que la dirección se expresa como una ubicación relativa a algún punto conocido, habitualmente, un valor de un registro del procesador.
- Una **dirección física**, también conocida como **dirección absoluta**, es una ubicación real de la memoria principal.

Reubicación



Paginación

Paginación

Tanto las particiones de tamaño fijo como las de tamaño variable son **ineficientes en el uso de la memoria**. Los primeros provocan fragmentación interna, los últimos fragmentación externa.

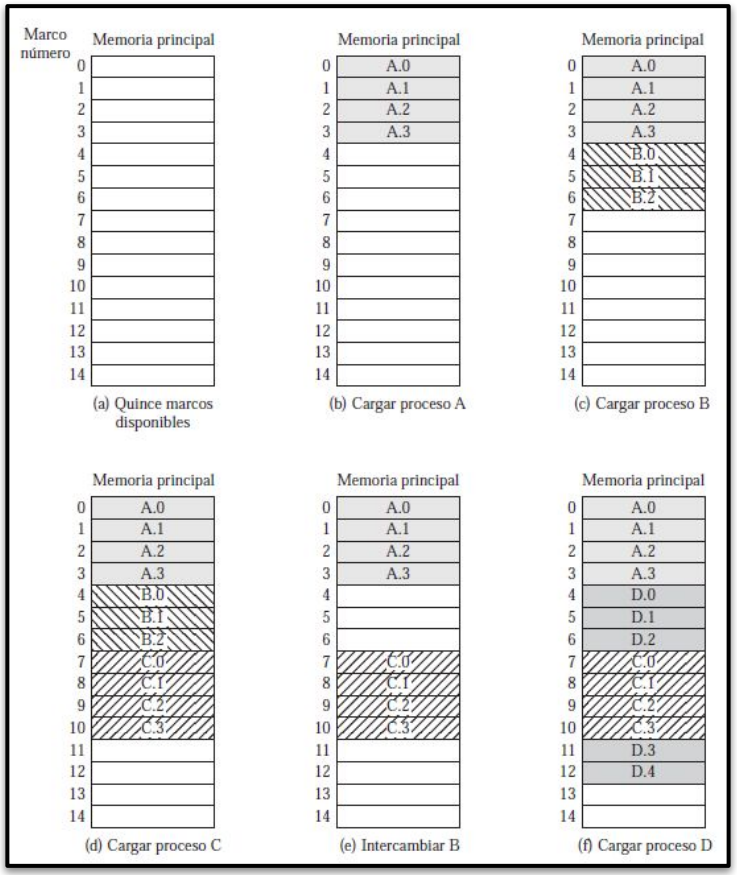
¿Cómo solucionamos esto?

- Podemos dividir la memoria principal en porciones de tamaño fijo relativamente pequeñas (a las que llamaremos **marcos**).
- De forma análoga, dividimos los procesos en porciones pequeñas del mismo tamaño (a las que llamaremos **páginas**).

A los marcos de la memoria se les asignan páginas.

- La fragmentación interna se ve reducida a la pequeña parte de la última página del proceso (la única que puede tener un tamaño inferior a las demás).
- No existe fragmentación externa.

Paginación


Marco número
Memoria principal
(a) Quince marcos disponibles
(b) Cargar proceso A
(c) Cargar proceso B
(d) Cargar proceso C
(e) Intercambiar B
(f) Cargar proceso D

Paginación

¿Y si no hay suficientes marcos contiguos sin utilizar para un proceso?

- Nos da igual → utilizaremos el concepto de **dirección lógica**.

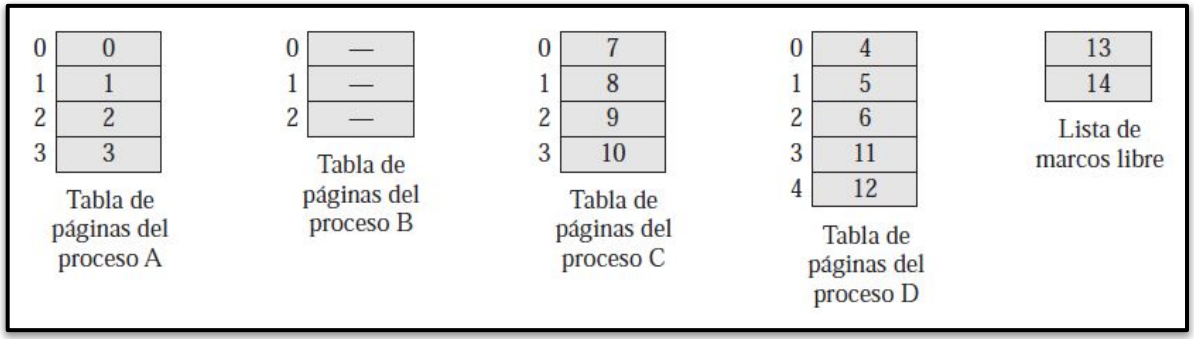
El S.O. mantiene **una tabla de páginas** por cada proceso, que muestra la ubicación del marco de cada página del proceso.

- Dentro de nuestro programa, **cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página**.

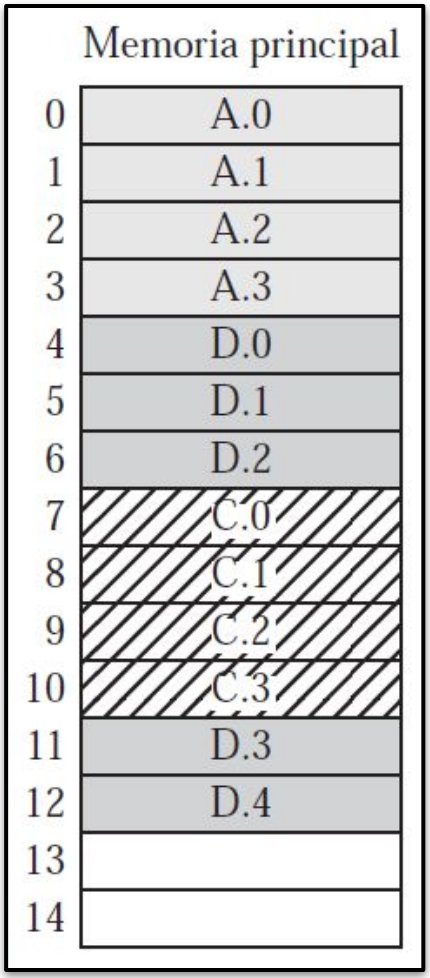
El **hardware del procesador** es el encargado de **traducir las direcciones lógicas a direcciones físicas**.

- El procesador necesita poder acceder a la tabla de páginas del proceso actual, para **poder traducir las direcciones lógicas que se encuentre (número de página + desplazamiento) a direcciones físicas (número de marco + desplazamiento)**.

Paginación



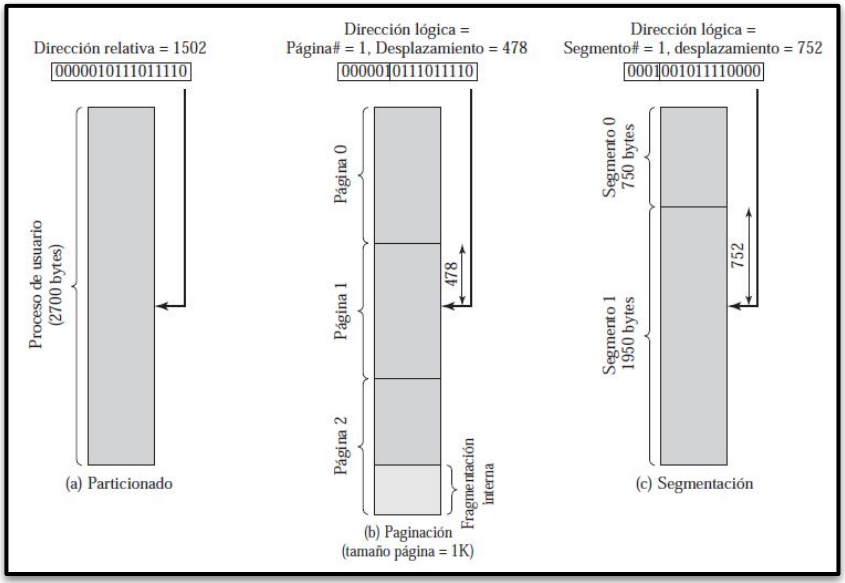
Las tablas de página se indexan fácilmente por el número de página.



Paginación

Para hacer que este esquema de paginación sea conveniente, el tamaño de página y marco **debe ser una potencia de 2**.

- Usando potencias de 2 es fácil demostrar que la dirección relativa, que se define con referencia al origen del programa, y la dirección lógica, expresada como un número de página y un desplazamiento, es lo mismo.



- Estamos utilizando direcciones de 16 bits.
- El tamaño de página es de 1KB (1024 bytes).
- La dirección relativa 1502, en binario es “0000010111011110”.
- Con una página de tamaño 1K, necesitamos un desplazamiento de 10 bits, lo que nos deja 6 bits para indicar el número de página.
- Es decir, como máximo, un programa puede estar compuesto por 64 (2^6) páginas de 1KB.
- La dirección relativa 1502 corresponde al desplazamiento de 478 (0111011110) de la página 1 (000001), que forman 16 bits: “0000010111011110”.

Paginación

Las ventajas de usar un tamaño de página potencia de 2 no acaban aquí:

- Es relativamente sencillo implementar una función que ejecute el hardware para llevar a cabo dinámicamente la traducción de direcciones en tiempo de ejecución.

Consideramos una dirección *n* + *m* bits, donde los *n* bits de la izquierda corresponden al número de página y los *m* bits de la derecha corresponden al desplazamiento.

En el ejemplo anterior *n* = *6* y *m* = *10*. Para la traducción de direcciones necesitaríamos hacer:

- Extraer el número de página como los *n* bits de la izquierda de la dirección lógica.
- Utilizar el número de página como un índice de la tabla de páginas del proceso para encontrar el número de marco, al que llamaremos *k*.
- La dirección física inicial del marco es $k \times 2^m$, y la dirección física del byte referenciado es dicho número más el desplazamiento. Esta dirección física no necesita calcularse: es fácilmente construida concatenando el número de marco al desplazamiento.

Paginación

Siguiendo el ejemplo anterior, partimos de la siguiente dirección lógica (página 1, desplazamiento 478):

000001 0111011110

Supongamos ahora que la página reside en el marco de memoria principal 6 (000110). Por tanto, la dirección física corresponde al marco número 6, desplazamiento 478:

000110 0111011110

Segmentación

Segmentación

Un programa de usuario se puede subdividir usando segmentación, una técnica por la cuál el programa y los datos asociados se dividen en un número de **segmentos**.

- No se requiere que todos los programas sean de la misma longitud, aunque sí hay una longitud máxima de segmento.

De forma similar al caso anterior, una **dirección lógica usando segmentación está compuesta por dos partes (número de segmento y un desplazamiento)**.

Debido al uso de segmentos de diferente tamaño, la segmentación es similar al particionamiento dinámico.

- La diferencia está en que la segmentación de un programa podría ocupar más de una partición, y estas no necesitan ser contiguas.
- Al igual que en el particionamiento dinámico, la segmentación elimina la fragmentación interna, pero sufre de fragmentación externa, aunque en menor medida que el particionamiento dinámico.

Segmentación

Mientras que la **paginación es invisible** al programador, la **segmentación es normalmente visible**.

La segmentación se ofrece como una utilidad para organizar programas y datos. Normalmente, el programador o el propio compilador asignarán programas y datos a diferentes segmentos.

El uso de la segmentación trae algunos inconvenientes:

- El programador debe ser consciente de la limitación de tamaño de segmento máximo.
- Al haber segmentos de diferente tamaño, no hay una relación simple entre direcciones lógicas y direcciones físicas.

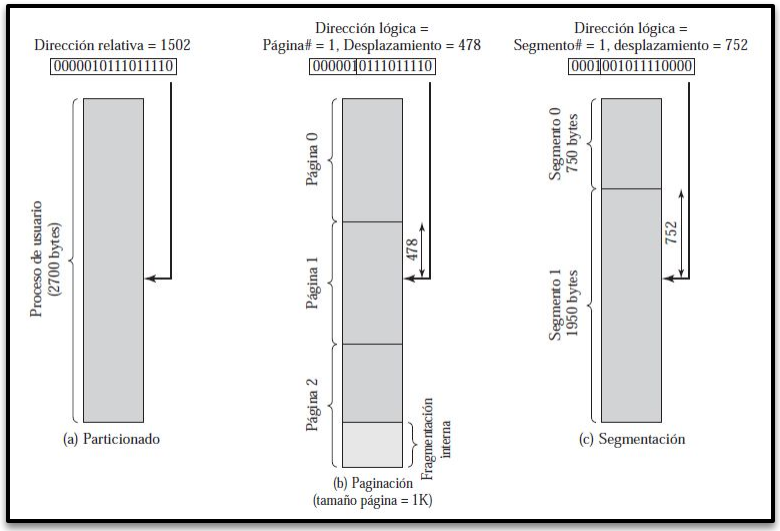
De forma análoga a la paginación, un esquema de segmentación sencillo haría uso de una tabla de segmentos por cada proceso y una lista de bloques libre de memoria principal.

- Cada entrada de la tabla de segmentos tendría que proporcionar la dirección inicial de la memoria principal del correspondiente segmento, junto con la longitud del mismo, para asegurar que no se utilizan direcciones no válidas.

Segmentación

Cuando un proceso entra al estado “Ejecutando”, la dirección de su tabla de segmentos se carga a un registro especial utilizado por el hardware de gestión de la memoria.

Consideramos ahora una dirección de *n* + *m* bits, donde los *n* bits de la izquierda corresponden al número de segmento, y los *m* bits de la derecha, corresponden al desplazamiento.

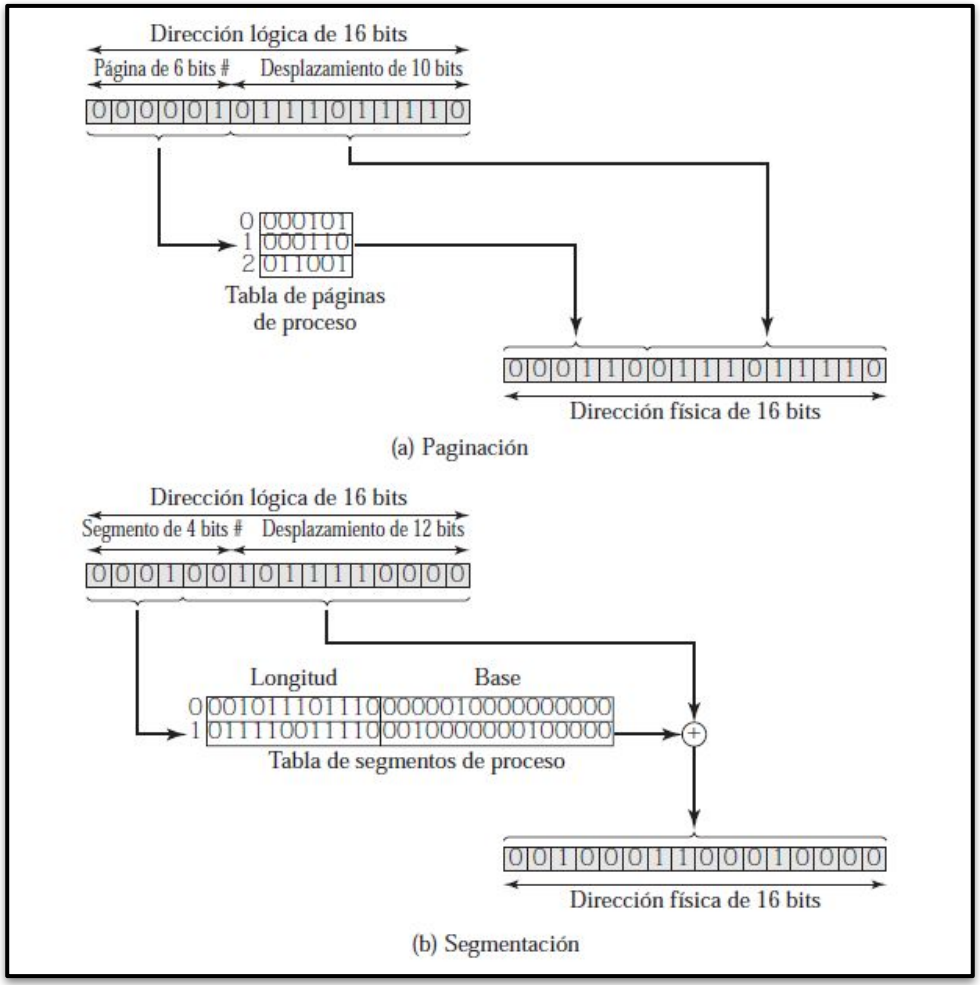


En el ejemplo, *n* = *4* y *m* = *12*, por tanto, el tamaño máximo de segmento es 2^{12} = 4096 bytes.

Segmentación

Para la traducción de las direcciones hay que hacer los siguientes pasos:

- Extraer el número de segmento como los *n* bits de la izquierda de la dirección lógica.
- Utilizar el número de segmento como un índice a la tabla de segmentos del proceso para encontrar la dirección física inicial del segmento.
- Comparar el desplazamiento, expresado como los *m* bits de la derecha, y la longitud del segmento. Si el desplazamiento es mayor o igual que la longitud, la dirección no es válida.
- La dirección física deseada es la suma de la dirección física inicial y el desplazamiento.


Dirección lógica de 16 bits
Página de 6 bits #
Desplazamiento de 10 bits
0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1 0
0 000101
1 000110
2 011001
Tabla de páginas de proceso
0 0 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0
Dirección física de 16 bits
(a) Paginación
Dirección lógica de 16 bits
Segmento de 4 bits #
Desplazamiento de 12 bits
0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0
Longitud
Base
0 001011101110 000001000000000
1 011110011110 001000000001000000
Tabla de segmentos de proceso
+
0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0
Dirección física de 16 bits
(b) Segmentación

Ejercicio

En un sistema en el que se administra la memoria por paginacion se dispone de 256 KB, siendo el tamaño de página empleado de 4 KB, y siendo el tamaño de palabra de 32 bits.

A. Razone si las direcciones de memoria 0xABC10008 y 0xABC100AA pertenecen al mismo marco. ¿Y las direcciones 0xABC1FA00 y 0xABC2FA08?
B. ¿De cuántas páginas físicas dispone el sistema?
C. Si las páginas fueran de 1 KB, ¿pertenecen las direcciones 0xABC10008 y 0xABC800AA al mismo marco? Razone la respuesta.
D. Indique el tamaño máximo de la tabla de páginas de un proceso suponiendo que cada entrada ocupa 8 bytes.