Sistemas Operativos 2

Unidad 1: Administración de memoria

Paginación y segmentación

René Ornelis
Primer semestre de 2025

Contenido

Índice de figuras

Paginación y segmentación

La paginación y la segmentación son los mecanismos actuales de los sistemas operativos por medio de los cuales se cumple con las funciones de relocalización, protección y compartición.

1 Evolución de los mecanismos de memoria

En el desarrollo de los sistemas operativos, el hardware necesario para que este pudiera cumplir con las funciones de relocalización ha evolucionado en las siguientes fases:

- Carga estática
- Carga dinámica: Registros de relocalización/límite
- Paginación

1.1 Carga estática

Inicialmente, los sistemas operativos no contaban con mecanismos de protección de memoria por ser sistemas monousuarios y no existía ningún hardware especial para esto. En estos sistemas se utilizó la **carga estática** de programas, lo cual consiste en que el programador debía conocer con antelación la posición de memoria en la que el programa sería cargado a memoria para poder hacer *referencias absolutas* de memoria, tanto para sus datos (lecturas de memoria) como para su código (saltos y saltos condicionales). Un ejemplo tardío de esto es el sistema DOS, el cual tenía los ejecutables de tipo .COM que acompañaban al sistema como programas especiales para línea de comandos. Estos programas .COM solo funcionaban con la versión específica de DOS ya que estaban programados para iniciar en una determinada posición de memoria según la memoria del sistema operativo. Al cambiar de versión de sistema operativo, cambiaba la memoria utilizada por el sistema operativo, por lo que la posición inicial de memoria disponible para los programas era diferentes y los programas de otra versión ya no funcionaban.

1.2 Carga dinámica: registros de relocalización/límite

Esto cambió con la carga dinámica de programas, la cual ya no necesitaba que estos trabajaran en una dirección específica de memoria porque utilizaban referencias relativas de memoria, las cuales estaban basadas en el concepto general de registro de relocalización (o registro base) y registro límite, los cuales funcionaban en el Memory Management Unit (MMU) tal como se muestra en la Figura 1.

Con este mecanismo, al acceder una dirección lógica (P_i) el MMU verifica:

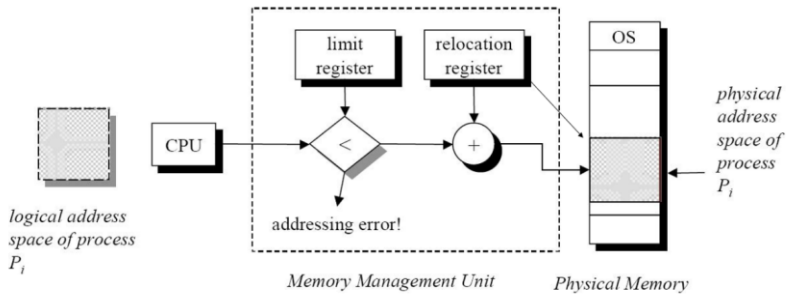

Figura 1: Traducción de memoria lógica a real con registros de relocalización

1. Que la dirección esté dentro del rango de memoria asignado al proceso, es decir que el proceso no esté accediendo a memoria más allá de su área de memoria asignada.
2. Si la dirección de memoria está dentro del rango de memoria asignada al proceso, entonces la dirección lógica se suma al contenido del registro relocalizador para formar la dirección física a acceder.

Tener en cuenta que el registro límite y del registro relocalizador son parte del MMU y su contenido será determinado por los valores del proceso en ejecución, es decir, al momento de que a un proceso se le otorga el uso del procesador, junto con todos sus registros y estado en general, debe ser restaurado el valor de estos registros.

Por ejemplo, una implementación en los primeros procesadores de Intel, la relocalización se realizaba a través de varios registros base, como el DX, CS, SS, etc. de forma que las instrucciones

mov ax , [FF]

jump [5]

equivalen a:

mov ax , DX:[FF]

jump CS:[5]

De esta forma el programador no se preocupa de las direcciones absolutas de memoria, sino solo de las direcciones relativas.

2 Direcciones y tamaño de palabra

- Tamaño de palabra: número de bits que se transfieren de memoria a un registros (ancho del bus del sistema)
- Una dirección puede referirse a un byte o a un grupo de bytes o a una palabra

3 Paginación

La paginación es el mecanismo actual por medio del cual los sistemas operativos y los procesadores cumplen con la función de relocalización, de protección y compartición de las páginas de memoria. La paginación consiste en dividir la memoria en partes iguales, cada una de

Fragmentación interna: Es desperdicio de memoria que existe dentro de un bloque B de tamaño N, que se entrega a un proceso que solicitó un bloque de tamaño K y K < N. Existe un desperdicio de memoria N-K, *dentro* del bloque B.

ellas denominada página o marcos de página. Cada una de estas páginas es la unidad mínima de asignación de memoria para cada proceso y cualquier petición que realice un proceso para adquirir nueva memoria, implica que el sistema le asignará tantas páginas como sea necesario para cumplir con el requerimiento. Dado que los requerimientos no siempre van a coincidir con un múltiplo del tamaño de las páginas es posible que dentro de alguna página no se aproveche el espacio al máximo. Por ejemplo: si un sistema tiene una un tamaño de página de 10kb y un proceso pide un bloque de memoria de 45Kb, el sistema le asignará un total de 5 páginas lo que implica que en la última página habrá un desperdicio de 5Kb, el cual el proceso ni ningún otro proceso del sistema podrán utilizarlo. A este fenómeno se le conoce como **fragmentación interna.**

Tal como se puede apreciar en la Figura 2, el mecanismo de paginación consiste en:

1. Direcciones lógicas o virtuales, las cuales están divididas en dos partes: el número de página (*Page #*) y el desplazamiento (*Offset*) dentro de esa página
2. Tabla de páginas (*Page Table)* que contiene la información de cada página, principalmente la dirección base de esa página (*Frame #*) o la dirección en memoria real donde está ubicada la página solicitada.
3. Registro apuntador a la tabla de páginas (*Page Table Pointer*) que contiene la dirección de la tabla de páginas del proceso en ejecución. En el momento de un cambio de contexto (cuando a un proceso se le asigna el uso del procesador) parte de la información que se tiene que restaurar del proceso es su Page Table Pointer, junto con todos registros de estado del proceso, como el Instruction Pointer, Data Segment, Stack Segment, etc.

El proceso de **traducción de direcciones** para trasladar una dirección lógica (o virtual) a una dirección física, involucra los siguientes pasos:

1. En el momento que el proceso acceda a esta dirección lógica el MMU o la unidad de memoria administración de memoria, toma la parte de número de página (*Page #*) la busca en una tabla de páginas (*Page Table*).
2. Si la página existe, se toma la dirección base (*Frame #*) y se le agrega la parte del desplazamiento (*Offset)* de la dirección lógica para formar la dirección real.

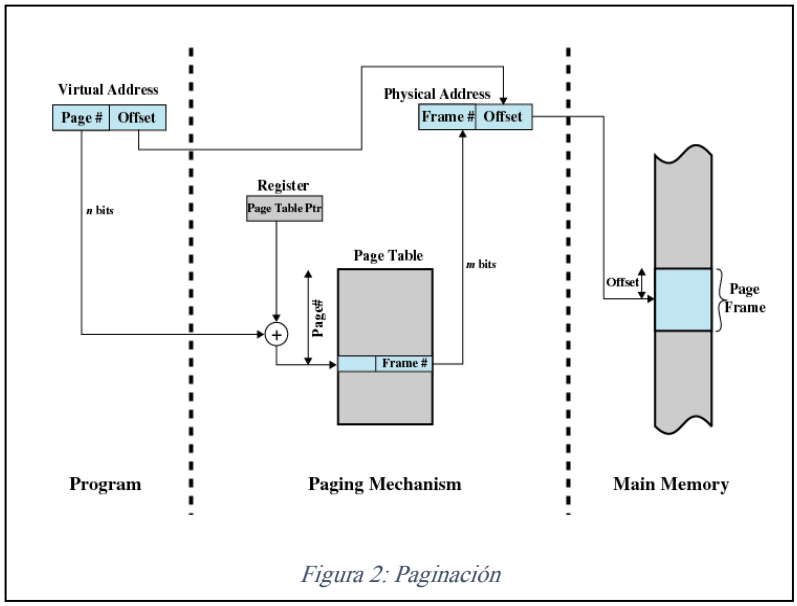

Figura 2: Paginación

Al iniciar el sistema operativo se puede configurar el MMU para determinar cuál va a ser nuestro tamaño de página, lo cual debe elegir cuidadosamente para mantener un balance adecuado entre el tamaño de la página y el tamaño de la tabla de páginas. Si tenemos páginas muy grandes la fragmentación

interna va a ser mayor y por lo tanto va a haber mayor desperdicio memoria, por el otro lado si tenemos páginas muy pequeñas el número de páginas crecerá, lo cual implica que nuestras tablas de páginas también crecerán, lo cual tendrá un impacto en la búsqueda de páginas al momento de acceder.

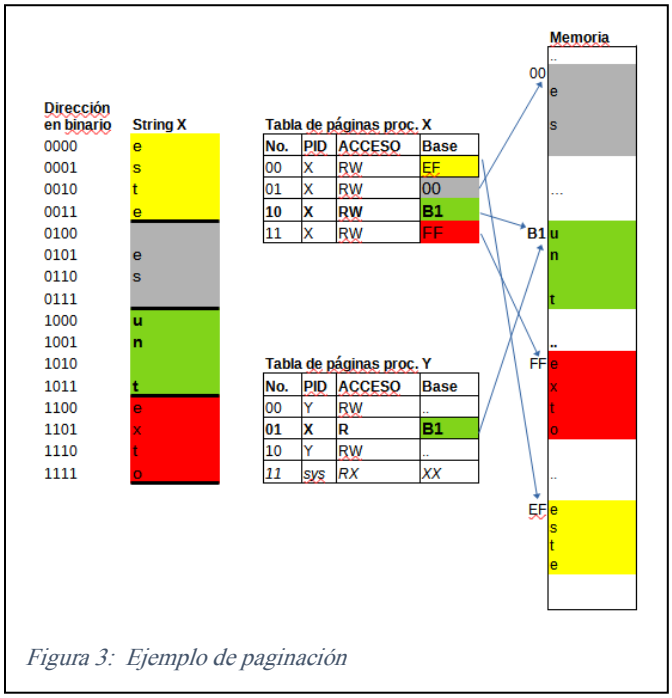

Figura 3: Ejemplo de paginación

Consideremos el ejemplo de la Figura 3. Supongamos un sistema que tiene capacidad de direccionamiento de cuatro bits y un tamaño de palabra de 8 bits. Usaremos en la dirección lógica 2 bits para el tamaño de página y 2 bits para el desplazamiento; esto significa que este sistema teórico tiene un máximo de 16 direcciones. Si el proceso necesita un string con contenido "*este es un texto*", la representación en memoria de la tabla de páginas va a tener un total de cuatro páginas, en la gráfica numeradas de 00 a la 11 en (0 a 3 en decimal). La información de la tabla de páginas incluye el *process id (PID)* , el cual es la identificación del proceso propietario de esa página y los accesos que el proceso en ejecución tienen sobre dicha página. En este caso vemos que todas las todas las entradas de la tabla de páginas tienen por propietario al proceso X, con acceso de lectura y escritura.

También en la tabla de páginas del proceso X está la dirección base de cada uno de los marcos de páginas que corresponden a la posición de memoria de cada dirección lógica.

De esta forma, lo que para el proceso es una secuencia de direcciones que va de las 00 a las 1111 (16 en decimal), en memoria física cada una de las páginas puede estar en cualquier orden de forma independiente. Para aplicar la relocalización, cuando el sistema mueva una de estas páginas, a cualquier otra posición de memoria, lo único que tiene que hacer es actualizar la entrada correspondiente en la tabla de páginas, sin que esto afecte el direccionamiento lógico del proceso. Es decir, el proceso sigue viendo las mismas direcciones lógicas sin ningún cambio, aunque la página física cambie de lugar.

Supongamos también que existe un proceso Y, el cual debe utilizar parte de la memoria del proceso X. Para esto, el proceso X debe compartir explícitamente una página para que otro proceso pueda accederla. En este caso ejemplificamos que se va a compartir la página 10 del proceso X, por lo que en la tabla de páginas de proceso Y, se agregará una entrada en la tabla de páginas para acceder a la página del proceso X. Notemos dos cosas:

1. La dirección lógica con la que el proceso Y va a acceder a la página compartida del proceso X no es la misma. Son dos entradas independientes, porque la asignación en la tabla de páginas depende del estado en que esta estaba antes de dicha asignación.
2. La entrada de la tabla de páginas del proceso Y que corresponde a la página compartida, indica que pertenece al proceso X y que el proceso Y tiene solo acceso de lectura a dicha página.

4 Enlace estático y dinámico

Otro ejemplo en el cual podemos apreciar el uso de la paginación es el caso del enlace dinámico de las aplicaciones. Al momento de generar ejecutables, existen 2 formas de generar un ejecutable en cualquier compilador: uno es **enlace estático** y otro es **enlace dinámico**. El enlace estático se refiere a que un ejecutable contiene en sí mismo todas las librerías y todo el código necesario por el cual puedes ejecutarse por completo. La forma de distribución es simplemente entregar el archivo ejecutable (programa.exe) y el usuario lo puede correr directamente. En el caso del enlace dinámico hay un ejecutable y una serie de archivos o librerías las cuales se enlazan dinámicamente a los programas conforme los van necesitando.

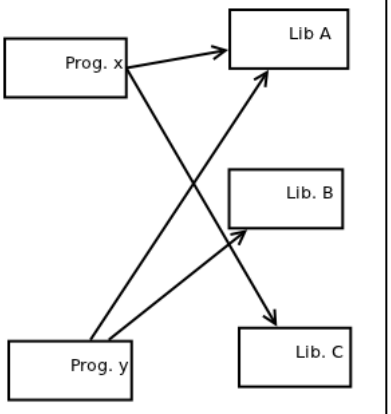

Figura 4: Programas con librerías

Supongamos el caso de dos programas que utilizan un conjunto común de librerías, tal como se muestra en la Figura 4, tenemos dos programas X y Y, los cuales utilizan un conjunto de librerías en común (librerías A, B y C), las cuales tienen diferentes funcionalidades y se pueden integrar a diferentes programas. En este ejemplo, vemos que el programa X utiliza la librería A y la librería C, por su lado el programa Y utiliza la librería A y la librería B.

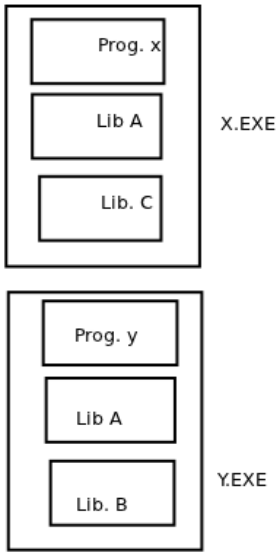

Figura 5: Compilación con enlace estático

En la Figura 5 podemos ver cómo estarían conformados los programas X y Y si estos fueran compilados de forma estática. Vemos que el programa X contiene dentro de su archivo (X.exe) el contenido específico del programa X, el contenido de la librería A y el contenido de la librería C. Por su parte el programa Y tiene dentro de su archivo (Y.exe) el contenido específico del programa Y, la librería A y la librería B.

Si el usuario corre ambos programas simultáneamente la librería A está cargada 2 veces en memoria: una vez dentro del programa X y otra dentro del programa Y, resultando en un desperdicio de memoria. Claro que en un sistema se utilizan muchos más que dos programas y tres librerías, por lo que el enlace estático implica un desperdicio de memoria y tiempo de carga de programas.

Para contrarrestar estas desventajas, se inventó el enlace dinámico que consiste en el uso compartido de librerías a través de varios programas como se ve en la Figura 6, en la cual podemos apreciar que se producen los siguientes archivos:

- Programa **X.exe**, el cual contiene el código específico y único del programa x
- Programa **Y.exe,** el cual contiene sólo el código específico del programa Y.

- **A.dll**, **B.dll** y **C.dll** los cuales contienen cada uno el código de las librerías correspondientes.

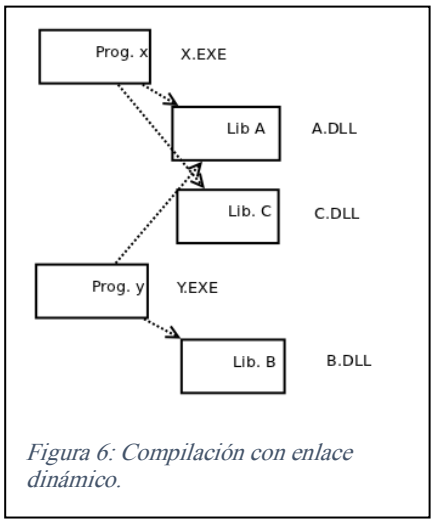

Figura 6: Compilación con enlace dinámico.

Al momento de ejecutarse el programa X, invoca al sistema operativo para indicar que necesita la librería A y, suponiendo que no hay ningún otro programa cargado con anterioridad, el sistema operativo determinará que la librería A no está en memoria y por lo tanto lo cargará y le compartirá a X la dirección de memoria de la librería A. Luego se repite el proceso para la librería C, después de los cual el programa X se puede ejecutar en su totalidad, ya que está en memoria todo lo que necesita.

Por su lado, el programa Y utiliza la librería A y la librería B. Al momento de cargar el programa Y, indicará al sistema operativo que necesita la librería A (A.dll) y el sistema determinará que este archivo ya está cargada en memoria, por lo que ya no la cargará de nuevo sino que sólo compartirá con el proceso Y la dirección de memoria para que pueda ejecutarla. Luego el programa Y indicará que necesita la librería B, se cargará en memoria y se le compartirá la dirección al programa Y.

De esta forma optimizamos la memoria, porque no habrá doble (o múltiple) carga del código de una librería utilizada por varios programas.

Una de las desventajas de utilizar múltiples librerías es de que se complica el mantenimiento de estas, dado que se tiene que mantener una compatibilidad "hacia atrás" de forma que las nuevas versiones deben ser compatibles, en interfaz y en funcionamiento, con las versiones anteriores para qué programas que utilizan dicha librería no se vean afectados por el cambio de librería.

Los compiladores proporcionan diferentes facilidades para los programadores que permiten realizar el enlace dinámico. Esto varía según el compilador, pero en general se puede resumir en el siguiente código:

```
/* se define la función y se indica en qué librería dinámica se encuentra */
int a1 (int a, X c) ; extern "a.dll" ;
….
/* se utiliza la función definida como si fuera una función definida localmente */
int z = a1 (0, null) ;
….
```

Esta facilidad, común en los compiladores, nos permiten ahorrarnos la interfaz directa con el sistema operativo, para lo cual transforman el código anterior en el siguiente:

```
/* variable tipo función para definición de parámetros */
typedef int tf(int a, X c) ;
….
/* se solicita cargar la librería */
int handle = loadLibrary ("a.dll") ;
```

```
/* se obtiene la dirección de la función dentro de la librería */
tf  a1 = getProc (handle, "a1") ;
…
/* se utiliza la función */
int z = a1(0, null) ;
….
/* se solicita descargar la función cuando ya no se utilizará */
unloadLibrary (handle) ;
..
```

Aquí, el programa X tiene que cargar manualmente la librería **a.dll** y obtener la dirección de memoria de la función que nos interesa dentro de esa librería ("a1.dll"), y después de asignar esa dirección podemos iniciar el uso de dicha función. Posteriormente hay que indicar al sistema operativo que descargue esa librería porque no la volveremos a utilizar.

Descargar una librería (*unloadLibrary*) no significa necesariamente que el sistema operativo la va a eliminar de memoria, sino que tiene en cuenta el conteo de procesos que la están usando. En nuestro ejemplo: cuando el proceso X solicita a la librería A, el sistema la carga a memoria y pone el contador de esta librería en 1; cuando el programa Y solicita la misma librería, el sistema se da cuenta de que ya está cargada y sólo incrementa su contador a 2. De forma inversa, si el programa X decide que ya no va a utilizar la librería A, puede dar la instrucción de descargarla, pero el sistema no lo va a eliminar de memoria, sino solo disminuye su contador a 1, lo que indica que aún hay un proceso que la está utilizando. Cuando el programa Y finalice e indique que se descargue la librería A, entonces el contador llegará a cero, por lo cual el sistema ya sabe que puede eliminarla de memoria sin afectar ningún proceso.

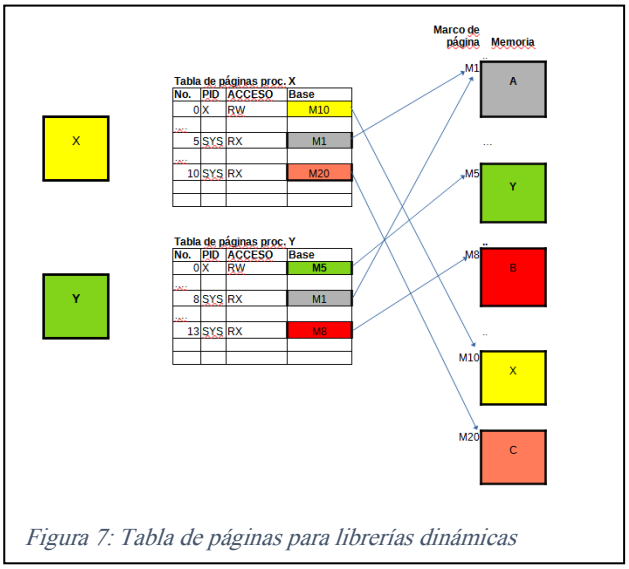

Figura 7: Tabla de páginas para librerías dinámicas

En la Figura 7, se muestra cómo podrían configurarse las tablas de páginas de cada uno de los procesos compartiendo las páginas que contienen las librerías. Notemos que en la tabla de páginas de cada proceso existen entradas, tanto para el código propio, como entradas para las páginas donde se encuentran las librerías. En el caso del proceso X, vemos que el marco de página M10 contiene la página específica de este, que no la está compartiendo con ningún proceso. También contiene el marco de página M1 que contiene el código de la librería A y el marco de página M20 que contiene el código de la librería C. Vemos en estas dos últimas entradas que el propietario de la M1 y M20 es el sistema, y el proceso X tiene acceso de lectura y ejecución en estas páginas.

De la misma forma, el proceso Y tiene la página M5 la cual es propia de este proceso y también tiene entradas para las librerías A y B que corresponden a las direcciones M1 y M8, respectivamente. Notar las mismas características: son propiedad del sistema y el proceso Y solo tiene acceso para leer y ejecutar.

5 Segmentación

Vemos que la paginación cumple con funciones de relocalización, compartición y protección, pero adolece del problema de la **fragmentación interna**. Adicionalmente, los programadores ven la memoria como un conjunto de bloques semánticamente relacionados, es decir se ve el bloque de datos, el bloque de código, el bloque para la pila, etcétera.

La segmentación surge como otro esquema de organización de la memoria, para superar los problemas de la paginación. Consiste básicamente del mismo proceso que la paginación, pero con una diferencia importante: el tamaño. Vimos que la paginación trabaja con páginas del mismo tamaño, lo cual nos lleva a la fragmentación interna, por lo cual se definió en la segmentación como bloques de memoria semánticamente relacionados de tamaño variable. Es decir, cada segmento es de diferente tamaño, lo cual nos elimina el problema de la fragmentación interna, ya que se entrega a cada proceso la cantidad exacta de memoria que está requiriendo.

El mecanismo porque por medio del cual funcionan la segmentación se muestra en la Figura 8. Aquí podemos ver que se tiene:

1. Una dirección lógica o dirección virtual, compuesta de un número de segmento (*Seg #*) y desplazamiento (*Offset*)
2. La tabla de segmentos, similar a la tabla de páginas que básicamente almacena la misma información, como la dirección base del segmento. También incluye el tamaño del segmento.
3. El registro apuntador a la tabla de segmentos (*Segment Table Pointer),* , el cual contiene la dirección de la tabla de segmentos del proceso, similar al Page Table Pointer de la paginación.

El mecanismo de traducción de direcciones en la segmentación involucra los siguientes pasos:

1. El MMU toma el número del segmento (*Seg #*), de la dirección lógica, y lo busca en la tabla de segmentos.
2. Con la información del tamaño en la tabla de segmentos el MMU compara que el desplazamiento de la dirección lógica (*Offset)* no sobrepase el tamaño de dicho segmento.
3. Si todo es correcto, a la dirección base de la tabla de segmentos (Base) se le agrega el desplazamiento, lo cual forma la dirección física de la memoria a acceder.

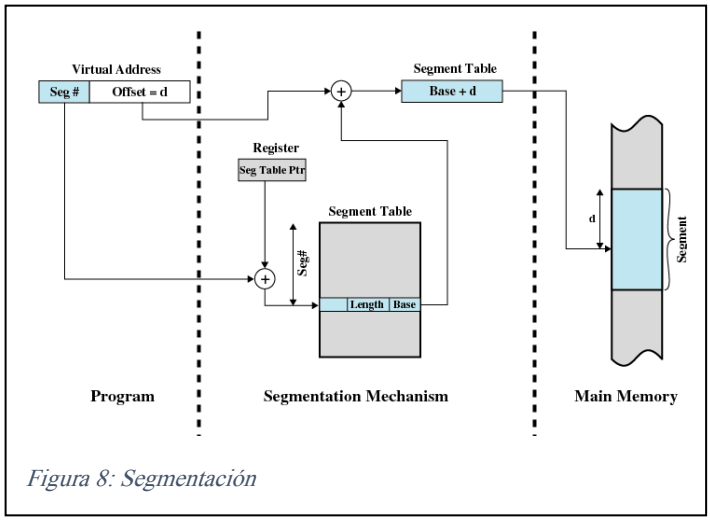

Figura 8: Segmentación

Como vemos, el mecanismo de paginación y segmentación funcionan con los mismos principios de la paginación, con la única diferencia del control del tamaño de los bloques de memoria. Sin embargo, aunque la segmentación resuelve el problema de la fragmentación interna, introduce el problema de la **fragmentación externa.**

Fragmentación externa: Es cuando en la memoria existen pequeños bloques de memoria libre, intercalados entre los bloques utilizados por procesos, de modo que, aunque la memoria disponible total sea de tamaño N (la suma de todos los bloques libres), el sistema no puede atender requerimientos de dicho tamaño, y aún tamaños menores, dado que está distribuido a lo largo de toda la memoria.

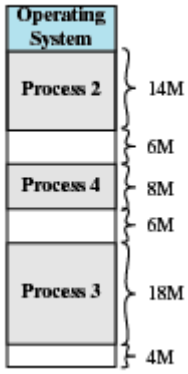

Figura 9: Memoria con fragmentación externa

Tal como se ilustra en la Figura 9, tenemos la memoria de los procesos 2, 3 y 4 los cuales ocupan bloques de tamaño 14Mb, 18Mb y 18Mb respectivamente. Asimismo, vemos que hay tres bloques libres, dos de 6Mb y uno de 4Mb, que hace un total de 16 Mb libres. Si un proceso requiere un bloque de 7Mb, no podrá ser atendido por el sistema dado que no hay ningún bloque libre que pueda satisfacer este requerimiento.

Para solucionarlo se debe recurrir a la a compactación de memoria la cual consiste en mover todos los bloques ocupados hacia un extremo de la memoria y dejar del otro extremo, un solo bloque libre con el total de memoria disponible tal como se muestra en la Figura 10.

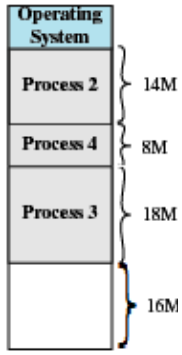

Figura 10: Memoria compactada

Aunque la compactación soluciona el problema de la fragmentación externa, hay que tomar en cuenta que mientras se realiza este procedimiento los procesos del usuario y del sistema deben detenerse hasta finalizar relocalización de todas las páginas. Esto implica un impacto directo al rendimiento general del sistema.

6 Sistemas híbridos

Como veremos en el estudio de la memoria virtual, adicional a la fragmentación externa, los sistemas de segmentación introducen la complejidad de la variabilidad de los tamaños de los segmentos que complican el control y la eficiencia de estos. Por los tanto, surgen los sistemas híbridos, los cuales intentan tomar lo mejor de dos mundos y proveen un primer nivel de segmentos los cuales y en un segundo nivel paginación. De esta forma el programador solicita segmentos de memoria y transparentemente estos se dividen en páginas lo cual facilitará el manejo posterior de la memoria virtual. Aunque posiblemente haya fragmentación interna, esta se limita a la última página de cada segmento.

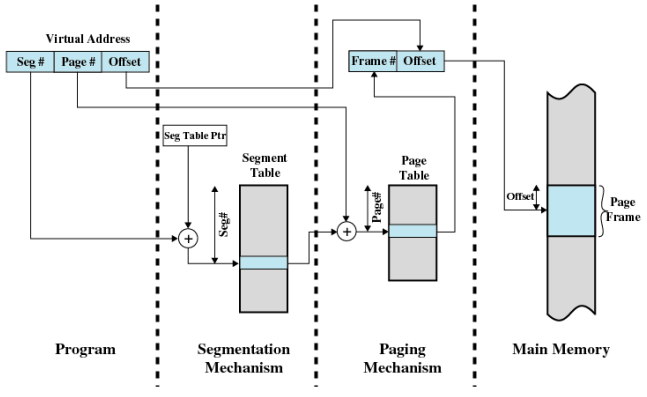

Figura 11: Sistema híbrido de segmentación/paginaciòn

Notemos en la Figura 11 que la dirección lógica (*Virtual A*ddress) ahora se conforma de tres partes:

1. El número de segmento (*Seg #*)
2. El número de página (*Page #*)
3. El desplazamiento (Offset).

El proceso de traducción de direcciones es similar a los esquemas de un nivel:

1. Se toma el número de segmento de la dirección lógica se busca en la tabla de segmentos. En esta configuración, la tabla de segmentos ya no tiene la dirección de memoria del segmento, sino la dirección a la tabla de páginas de ese segmento.
2. En la tabla de página del segmento, se busca la segunda parte de la dirección lógica, el número de página. Esto nos dará la dirección base de esta página
3. Se suma el desplazamiento, que es la tercera parte de la dirección lógica, a la dirección base que se obtuvo de la tabla de páginas, lo cual nos da la dirección física.

7 Paginación multinivel

Como se puede en la Figura 12, el concepto de 2 niveles que se utiliza en la segmentación/ paginación, se puede expandir al concepto de paginación multinivel, la cual consiste de una jerarquía de tablas de páginas y al final un desplazamiento dentro de la página, lo cual puede ser más eficiente si los procesos no utilizan todo el rango disponibles de páginas y solo utilizan una fracción de las mismas, ya que la creación de las tablas de páginas se hace dinámicamente, según se van necesitando las páginas.

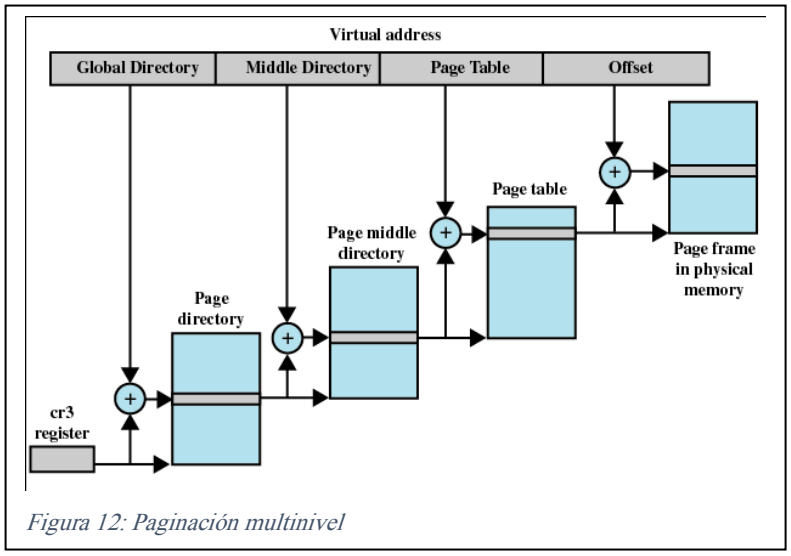

Figura 12: Paginación multinivel