Gestión de memoria

El sistema operativo debe asignar la memoria a los procesos y controlar la memoria que está asignada.

Monotarea	Multitarea	
	Memoria real	Memoria virtual
Memoria real	Particiones fijas Particiones variables Paginación pura Segmentación pura	Memoria virtual paginada Memoria virtual segmentada Memoria virtual segmentada y paginada

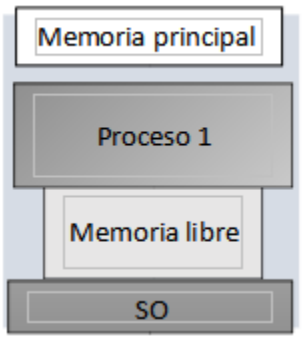
Monoprogramación

Es el esquema más sencillo. En un determinado momento solo hay un proceso cargado en memoria.

Dos grandes desventajas:

- La CPU se desaprovecha: si el proceso está a la espera de una operación de entrada/salida, la CPU esta ociosa.
- La memoria se desaprovecha: un proceso no ocupa toda la memoria.

Por este motivo, los sistemas operativos actuales utilizan la multiprogramación, cargando varios programas en memoria a la vez.



Multiprogramación con memoria real

En este caso el sistema operativo carga en memoria varios procesos. La carga de los procesos en memoria se puede realizar:

- de forma contigua: todo el proceso y sus datos en posiciones contiguas de memoria. Aquí podemos encontrar particiones fijas y particiones variables.
- de forma no contigua: los procesos se dividen en trozos, y cada uno puede cargarse en lugares diferentes y no contiguos de memoria

Asignación de memoria contigua

Hay dos opciones:

- **Particiones fijas** (o estáticas): en el momento del arranque del sistema operativo se divide la memoria en trozos de tamaño fijo (no tienen por qué ser del mismo tamaño). Según van llegando los procesos, el sistema operativo le asigna una partición en la que quepa. Cada partición contiene exactamente un proceso.
Se pueden cargar tantos procesos como particiones se hayan creado.
Desventajas:
  - Si un proceso es más grande que cualquiera de las particiones libres, debe ser redimensionado para poder cargarlo.
  - Normalmente un proceso es más pequeño que la partición en la que se carga. Se desaprovecha por tanto un porcentaje de la memoria → **fragmentación interna**
- **Particiones variables** (o dinámicas): no se crean las particiones en el momento de arranque del sistema, sino que se van creando conforme llegan los procesos.
Resuelve el problema de la fragmentación interna, ya que a cada proceso se le asigna exactamente la memoria que necesita. El número de particiones es variable, ya que depende de los procesos cargados en cada momento.
Principal desventaja: **fragmentación externa**. Conforme los procesos llegan y se les asigna memoria, van quedando huecos entre particiones. Estos huecos irán siendo usados para nuevos procesos. Pero llega un momento en que los huecos son tan pequeños, que no cabe ningún proceso. Se desaprovecha esa memoria.
Una solución a la fragmentación externa es la compactación, pero necesita que el sistema detenga temporalmente su actividad. Con este proceso, el sistema operativo mueve todos los procesos a posiciones contiguas, generando un hueco con toda la memoria no utilizada.

Asignación de memoria no contigua

En este caso se dividen los procesos en varias partes, y cada parte se carga en zonas no contiguas de memoria. Varias opciones posibles:

- **Paginación**

La memoria principal se divide en **marcos de página** de un tamaño fijo. Por ejemplo, si el sistema tiene 64KB y escogemos un tamaño de página de 1KB, la dividimos en 64 marcos de página.
Los procesos que se quieren cargar se dividen en **páginas** del mismo tamaño.
Si un proceso tiene un tamaño de 8,5KB, se dividirá en 9 páginas de 1KB. Cada página del proceso se almacena en un marco. En el ejemplo anterior consumimos 9 marcos de página.

Inconvenientes:

- Si un proceso necesita más memoria de la que hay disponible, no se puede cargar. Si hay procesos muy grandes en ejecución el número de procesos en paralelo es reducido.
- Hay fragmentación interna, el último marco de página no se ocupa en su totalidad. En el ejemplo anterior, se desaprovechan 0,5KB. El tamaño de página se configura en cada sistema operativo, cuanto más pequeño lo escojamos, menor será la fragmentación interna.

- **Segmentación**

En este caso se dividen los procesos en segmentos de tamaño variable. Cada segmento se coloca en una partición de la memoria de tamaño variable.
Evitamos la fragmentación interna (cada proceso ocupa exactamente la memoria que necesita), pero como inconvenientes:

- Como en la paginación, si un proceso necesita más memoria que la que hay disponible, no se puede cargar.
- Aparece la fragmentación externa, al generarse huecos en los que potencialmente no quepa ningún segmento.

- **Segmentación paginada**

Combinación de las dos anteriores para aprovechar sus ventajas. Los procesos se dividen en segmentos, y cada segmento emplea la técnica de paginación.

8.3 Multiprogramación en memoria virtual

Como se ha visto en la multiprogramación en memoria real, el principal inconveniente es que los procesos deben estar cargados de forma completa en memoria, aunque haya partes que no se estén utilizando. El número de procesos en ejecución queda muy limitado.
El uso de memoria virtual consiste en utilizar almacenamiento secundario como una extensión de la memoria. Emplea las mismas técnicas de asignación no contigua vistas antes (paginación/segmentación), pero permite cargar en memoria real solo algunas partes. El resto de cada proceso está almacenado en disco.
Debido a que no todas las partes de un proceso pueden estar cargadas en memoria en un instante determinado, cuando un proceso haga referencia a una parte que no se encuentra asignada en memoria principal, provocara un fallo de página o de segmento, y el gestor de memoria traerá dicha parte del proceso desde el disco a la memoria en ese momento.
Distinguimos:

- Memoria virtual paginada

- Memoria virtual segmentada
- Memoria virtual segmentada paginada

Es el sistema de gestión de memoria que utilizan la mayor parte de los sistemas operativos. En concreto, Linux y Windows (en sus versiones actuales) utilizan memoria virtual segmentada y paginada.

El proceso de intercambio de las partes de los procesos entre memoria principal y memoria secundaria se denomina **swapping**:
- Windows: utiliza ficheros de paginación (pagefile.sys)
- Linux: utiliza una partición dedicada montada en /swap. Típicamente se crea con un tamaño del doble de la memoria real. En algunas versiones modernas de Linux, por defecto se emplea un fichero en lugar de una partición (ejemplo, Ubuntu).

	DIRECCIÓN LÓGICA	DIRECCIÓN FÍSICA
Qué es?	Es la dirección virtual generada por la CPU	La dirección física es una ubicación en una unidad de memoria.
Espacio de dirección	El conjunto de todas las direcciones lógicas generadas por la CPU en referencia a un programa se denomina Espacio de direcciones lógicas.	El conjunto de todas las direcciones físicas asignadas a las direcciones lógicas correspondientes se denomina Dirección física.
Visibilidad	El usuario puede ver la dirección lógica de un programa.	El usuario nunca puede ver la dirección física del programa
Acceso	El usuario usa la dirección lógica para acceder a la dirección física.	El usuario no puede acceder directamente a la dirección física.
Generacion	La dirección lógica es generada por la CPU	La dirección física es calculada por MMU

Paginación

Las direcciones que genera un procesador cuando está ejecutando un proceso son lógicas, y son relativas a una dirección inicial 0. Cuando el procesador está ejecutando un proceso y quiere acceder a sus datos o instrucciones lo hace refiriéndose a estas direcciones relativas o lógicas.

Pero como los datos se almacenan en posiciones físicas de la memoria hace falta traducir de direcciones lógicas a físicas. Esta traducción se hace mediante un hardware especial dedicado a tal fin que se llama MMU: Memory Management Unit.

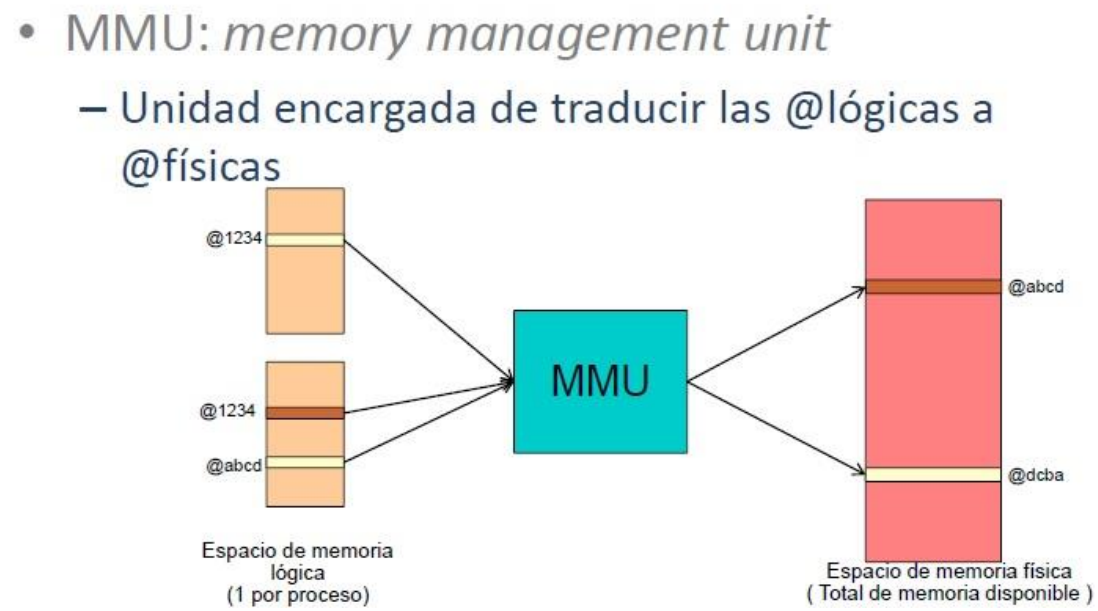
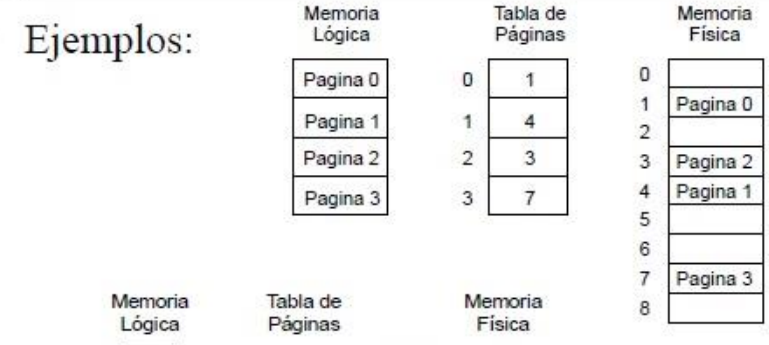


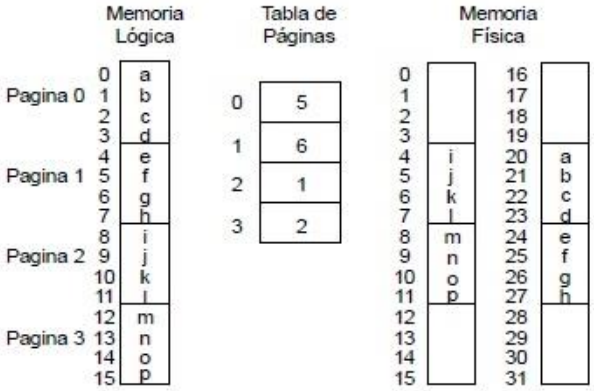
Tabla de páginas:

- Existe una tabla por cada proceso
- Hay una entrada por cada marco de página
- Suele guardarse en memoria y el sistema operativo debe conocer la dirección base de la tabla de cada proceso.

Paginación.

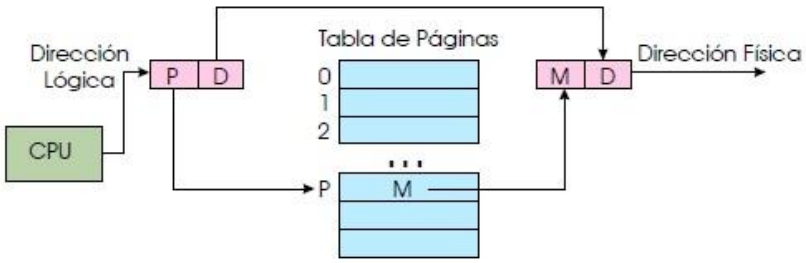
- Ejemplos:





Paginación.

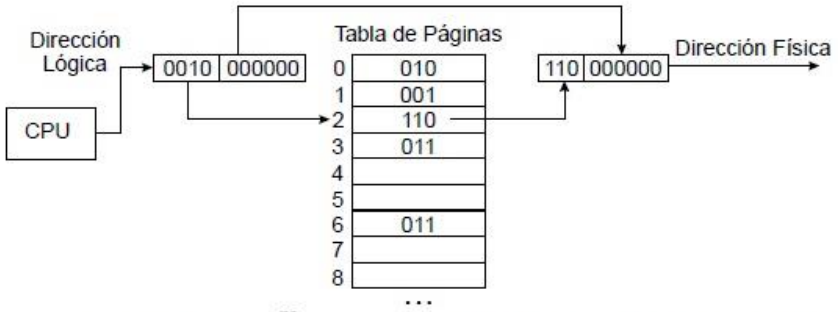
- Hardware de paginación: para traducción de direcciones



- La dirección lógica generada consta de dos partes:
  - Número de Pagina (P).
  - Desplazamiento dentro de la página (D).
- La tabla de páginas: (contiene la dirección base en memoria física)
  - Permite establecer una correspondencia entre el número de página y un número de marco de memoria física.
- La dirección física es el número de marco y el desplazamiento.

Paginación.

- Tamaño de páginas y marcos definidos por Hardware.
- Normalmente se escoge un tamaño de página potencia de 2:
  - Ya que es más fácil la traducción de direcciones lógicas a físicas.



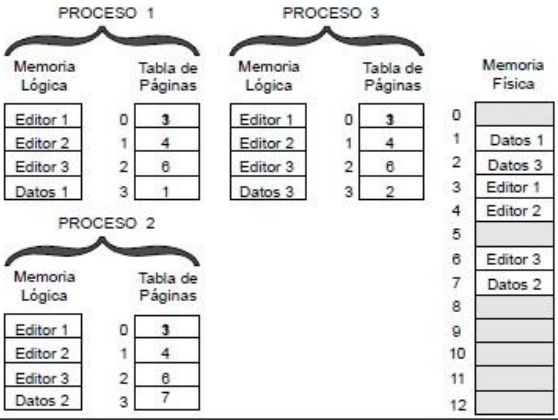
Tamaño memoria lógica 2^m
tamaño página 2^n (bytes o palabras)
P índice en tabla de páginas
D desplazamiento

M-n bits altos de la dirección lógica= P
n bits bajos de la dirección lógica = D

Paginación.

- **Ventaja:** Páginas Compartidas:
  - La paginación permite compartir código común entre varios procesos:
    - Sólo si el código es reentrante (no se modifica durante ejecución).
    - El área de datos de los procesos sería diferente.
    - Ejemplo: varios procesos ejecutan el mismo editor de textos

Una única copia
Del editor en
Memoria física



Segmentación

Un aspecto importante de la gestión de la memoria que la paginación convierte en inevitable es la separación de la visión que el usuario tiene de la memoria y la memoria física real. La visión del usuario no coincide con la memoria física real. La visión del usuario se transforma en la memoria física. La traducción de direcciones permite esta diferencia entre la memoria lógica y la física.

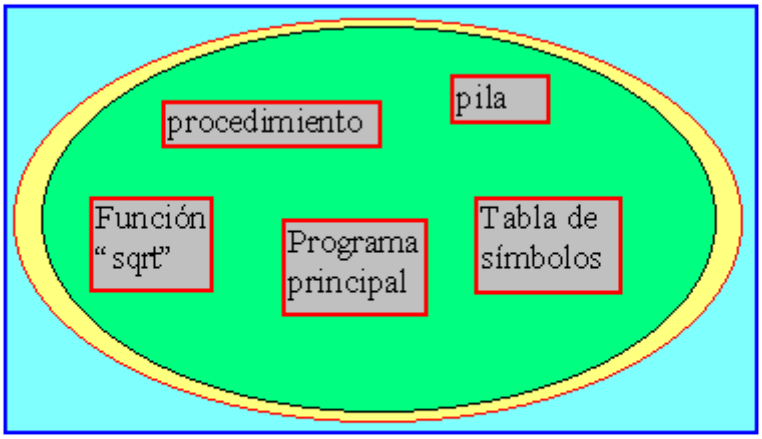


Figura 6.14. Espacio de direcciones lógicas

¿Cuál es la visión de la memoria que tiene el usuario? Concibe el usuario la memoria como una tabla lineal de palabras, algunas de las cuales contienen instrucciones mientras que otras contienen datos, o bien se prefiere alguna otra visión de la memoria? Hay un acuerdo general en que el usuario o programador de un sistema no piensa en la memoria como una tabla lineal de palabras. Más bien prefieren concebirla como una colección de segmentos de longitud variable, no necesariamente ordenados (fig. 6.14).

Consideremos cómo ve usted un programa cuando lo está escribiendo. Piensa en él como un programa principal, con un conjunto de **subrutinas, procedimientos, funciones o módulos**. También puede haber diversas estructuras de datos: tablas, matrices, pilas, variables, etc. Cada uno de estos módulos o elementos de datos se referencian por un nombre. Usted habla de la "tabla de símbolos", A "la función *Sqrt*", "el programa principal", sin tener en cuenta qué direcciones de memoria ocupan estos elementos. Usted no se preocupa de si la tabla de símbolos se almacena

antes o después de la función *Sqrt*. Cada uno de estos elementos es de longitud variable; la longitud está definida intrínsecamente por el propósito del segmento en el programa. Los elementos dentro de un segmento están identificados por su desplazamiento desde el principio del segmento: la primera instrucción del programa, la decimoséptima entrada de la tabla de símbolos la quinta función *Sqrt*, etc.

La **segmentación** es un esquema de administración de la memoria que soporta la visión que el usuario tiene de la misma. Un espacio de direcciones lógicas es una colección de segmentos. Cada segmento tiene un nombre y una longitud. Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro del segmento. Por lo tanto, el usuario especifica cada dirección mediante dos cantidades: un nombre de segmento y un desplazamiento. (Compárese este esquema con la paginación, donde el usuario especificaba solamente una única dirección, que el *hardware* **particionaba** en número de página y desplazamiento, siendo todo ello invisible al programador).

Por simplicidad de implementación, los segmentos están numerados y se referencian por un número de segmento en lugar de por un nombre. Normalmente el programa de usuario se ensambla (o compila), y el **ensamblador** (o el **compilador**) construye automáticamente segmentos que reflejan el programa de entrada. Un compilador de Pascal podría crear segmentos separados para (1) las variables globales, (2) la pila de llamada de procedimientos, para almacenar parámetros y devolver direcciones, (3) el código de cada procedimiento o función, y (4) las variables locales de cada procedimiento y función. El cargador tomaría todos esos segmentos y les asignaría números de segmento.

Hardware en segmentación

Aunque el usuario ahora puede referenciar los objetos del programa por medio de una dirección de dos dimensiones, la memoria física real es todavía, por supuesto, una secuencia unidimensional de palabras. La transformación se efectúa por medio de una **tabla de segmentos**.

El empleo de una tabla de segmentos se muestra en la figura 6.15. Una dirección lógica consta de dos partes: un número de segmento *s* y un desplazamiento dentro de ese segmento, *d*. El número de segmento se

utiliza como un índice en la tabla de segmentos. Cada entrada de la tabla de segmentos tiene una *base* de segmento y un *límite*. El desplazamiento *d* de la dirección lógica tiene que estar comprendido entre 0 y el límite de segmento. En caso contrario se produce una **excepción** al sistema operativo (tentativa de **direccionamiento lógico** más allá del fin de segmento). Si este desplazamiento es legal, se añade a la base para producir la dirección de la tabla deseada en la memoria física. La tabla de segmentos es así esencialmente una matriz de pares **registros** base/límite.

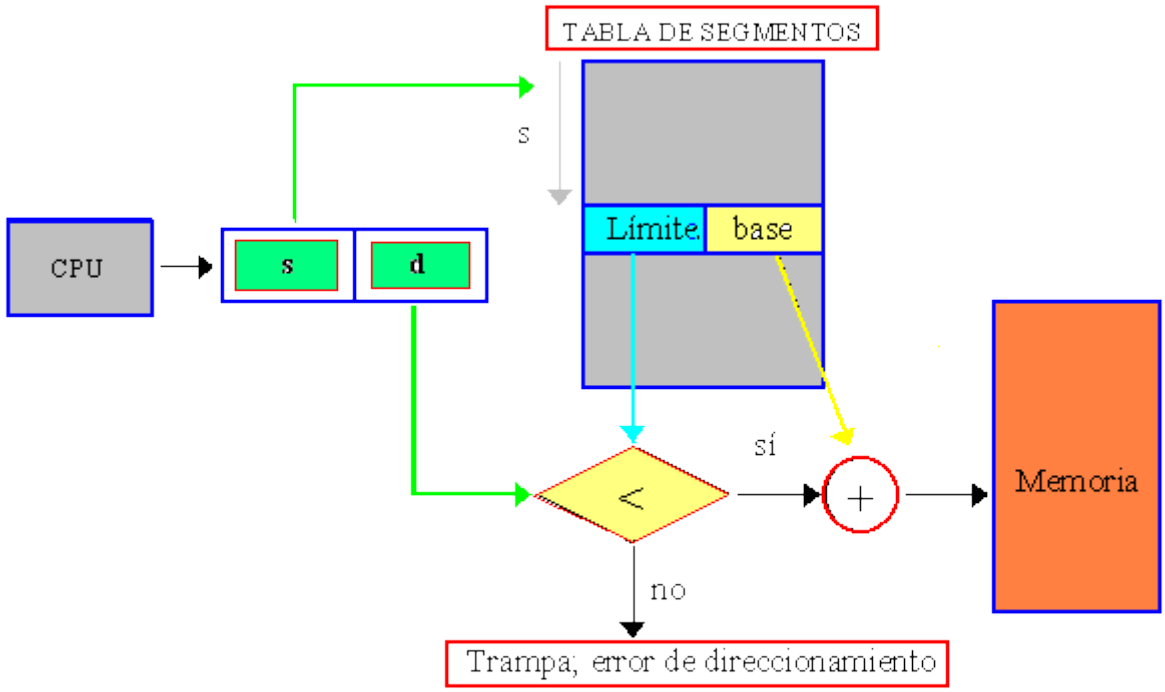


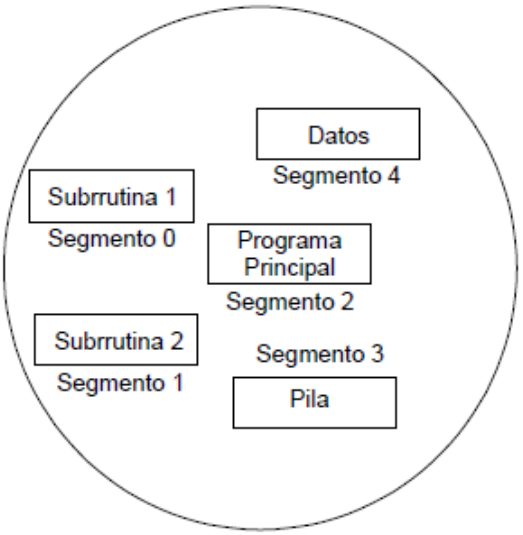
Figura 6.15 Hardware de segmentación

Ventajas de la segmentación:

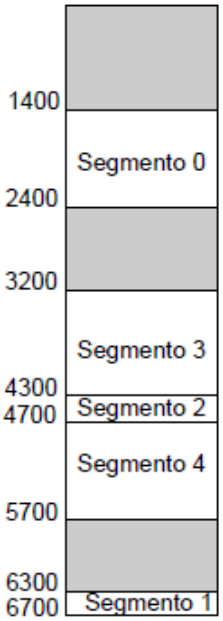
- Seguridad: Se puede proteger el contenido de un segmento contra escritura.
- Compartición de código y datos: Los segmentos se comparten cuando las entradas en las tablas de segmentos de dos procesos diferentes apuntan a las mismas posiciones físicas.

Segmentación.

–Ejemplo: sean 5 segmentos en memoria física



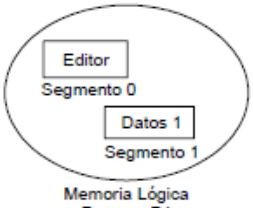
	Limite	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



- Acceso a byte 1200 del segmento 0 da error direccionamiento

Segmentación.

– Ejemplo compartición editor:



Memoria Lógica Proceso P1

Tabla de Segmentos Proceso P1

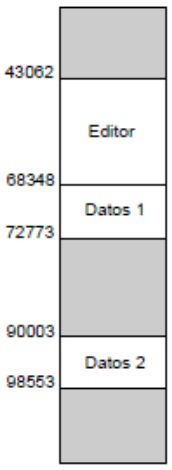
	Limite	Base
0	25286	43062
1	4425	68348

Memoria Lógica Proceso P2

Tabla de Segmentos Proceso P2

	Limite	Base
0	25286	43062
1	8550	90003

Memoria Física



– Si compartimos un segmento todos los procesos que lo comparten deben definir dicho segmento con el mismo código.

Dirección (S , desplazamiento)

Fragmentación

El sistema operativo tiene que encontrar y asignar memoria para todos los segmentos de un programa de usuario. Esta situación es similar a la paginación, excepto en el hecho de que los segmentos son de longitud *variable*; las páginas son todas del mismo tamaño.

La segmentación puede ocasionar entonces fragmentación externa, cuando todos los bloques libres de memoria son demasiado pequeños para acomodar a un segmento. En este caso, el proceso puede simplemente verse obligado a esperar hasta que haya disponible más memoria (o al menos huecos más grandes), o puede utilizarse la compactación para crear huecos mayores.

¿En qué medida es mala la fragmentación externa en un esquema de segmentación? La respuesta a estas preguntas depende principalmente del *tamaño medio de segmento*. En un extremo, se podría definir cada proceso como un segmento. En el otro extremo, cada palabra podría situarse en su propio segmento y reubicarse por separado. Esta disposición elimina la fragmentación externa. Si el tamaño medio de segmento es pequeño, la fragmentación externa también será pequeña. (Por analogía, consideremos la colocación de las maletas en el maletero de un coche; parece que nunca encajan bien. Sin embargo, si se abren las maletas y se colocan en el maletero los objetos sueltos, todo encaja). Puesto que los segmentos individuales son más pequeños que el proceso en conjunto, es más probable que encajen en los bloques de memoria disponibles.

Características

- Cada proceso en ejecución (esté activo, bloqueado o preparado) tiene su tabla de segmentos.
- Solapamiento: Se puede hacer que 2 segmentos se superpongan de manera que compartan direcciones de memoria física con direcciones lógicas diferentes. De esta manera, procesos diferentes pueden compartir información y código usando la memoria común.
- Protección de memoria: se puede añadir 3 bits a la tabla de segmentos para los permisos (r w x).

- Dos procesos pueden compartir segmentos de memoria de instrucciones/código, pero no para datos ya que esto complicaría la gestión.
- Es posible la redimensión de segmentos siempre que haya posiciones libres contiguas, o crear un nuevo segmento y copiar el contenido del anterior.
- Gestión compleja, sobre todo por su tamaño variable
- Permite la carga de segmentos a petición, de manera que no se disponga de todos los segmentos en memoria principal, que se puedan descargar a disco (en la zona de intercambio o swap).
- No hay fragmentación interna pero si que puede haber fragmentación externa.