TEMA 17

SISTEMAS OPERATIVOS: GESTIÓN DE MEMORIA

ÍNDICE

1. INTRODUCCIÓN

Conceptos

En un sistema monoprogramado, la memoria principal se divide en dos partes: una para el sistema operativo (monitor residente, núcleo) y otra parte para el programa que se ejecuta en ese instante. En un sistema multiprogramado, la parte de "usuario" de la memoria debe subdividirse aún más para hacer sitio a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el sistema operativo y se conoce como **gestión de memoria**.

La gestión de memoria consiste en la asignación de una memoria física de capacidad limitada a los diversos procesos que la soliciten con objeto de activarse y entrar en ejecución.

Asimismo la gestión de memoria tiene que proceder a poner fuera de la memoria los procesos temporalmente inactivos para dejar espacio a otros.

La gestión de memoria debe cumplir las siguientes funciones:

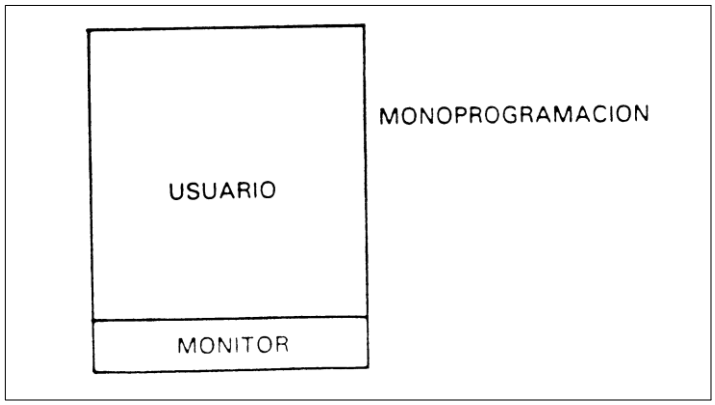
- Determinar la forma de asignación de memoria: cómo se debe dar, a quién y en qué orden.
- Técnicas de designación: qué procesos han de abandonar la memoria cuándo y en qué orden.
- Llevar la contabilidad de las posiciones asignadas y libres.

También debe facilitar el aislamiento y protección a los distintos espacios de memoria suponiendo al mismo tiempo una forma de sincronización entre procesos de variables comunes.

2. ASIGNACIÓN EN MONOPROGRAMACIÓN

Consiste en dividir la memoria en dos partes contiguas: una se le asigna a la parte del sistema operativo que ha de residir en memoria de forma permanente (monitor residente) y la otra se le asigna a los procesos de los usuarios.

La parte de sistema operativo que no constituye el monitor residente ha de ejecutarse en la zona de procesos.



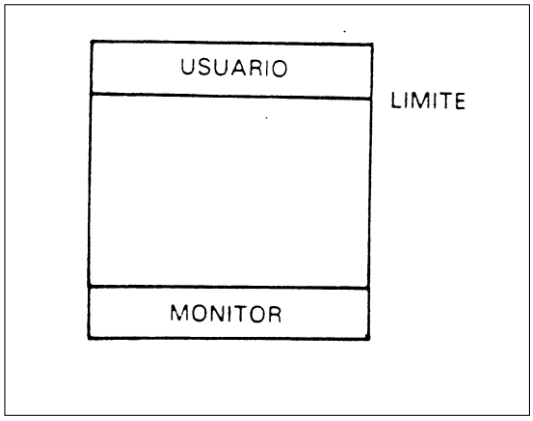
Para poder suministrar un espacio contiguo se localiza un extremo de la memoria. Dicho extremo se elige según donde se encuentre la tabla del vector de interrupciones.

Siempre que se carga un proceso se asegura que sus direcciones estén dentro de los límites que corresponden a la región de usuario. Si la dirección resulta por encima o debajo de esta el S.O. provoca una interrupción y el proceso no es cargado en memoria.

Esta dirección límite a veces se especifica por hardware y en otros casos se utiliza un registro límite o base donde se graba la dirección de comienzo de la zona del usuario. A veces el S.O. se graba en memoria de solo lectura y en otros se utilizan bits de protección.

Se observa que los límites de protección del S.O. son muy estáticos en todos los casos y por ello el tamaño asignado inicialmente es muy difícil de modificar.

Existen algunas formas de modificar el tamaño de carga del S.O., como la de cargarlo en la parte baja de la memoria e ir incorporando procesos de usuario de arriba hacia abajo de forma que la zona intermedia quede libre y permita expansiones al monitor.

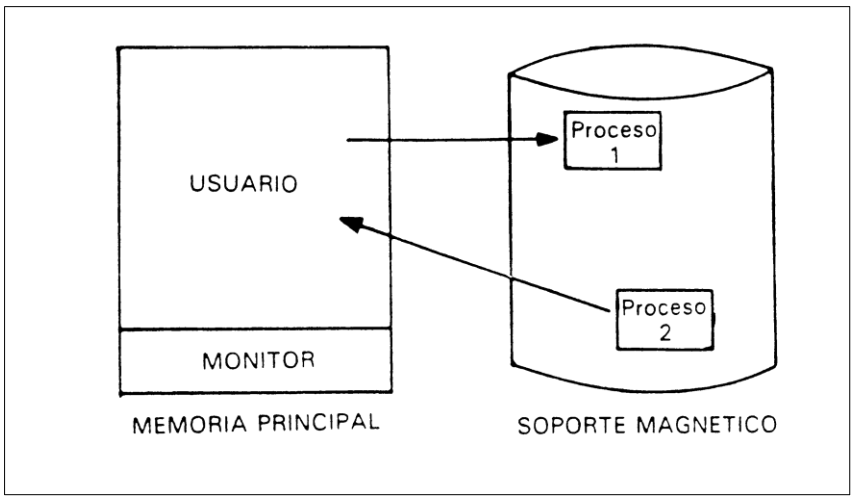


2.1. Swapping

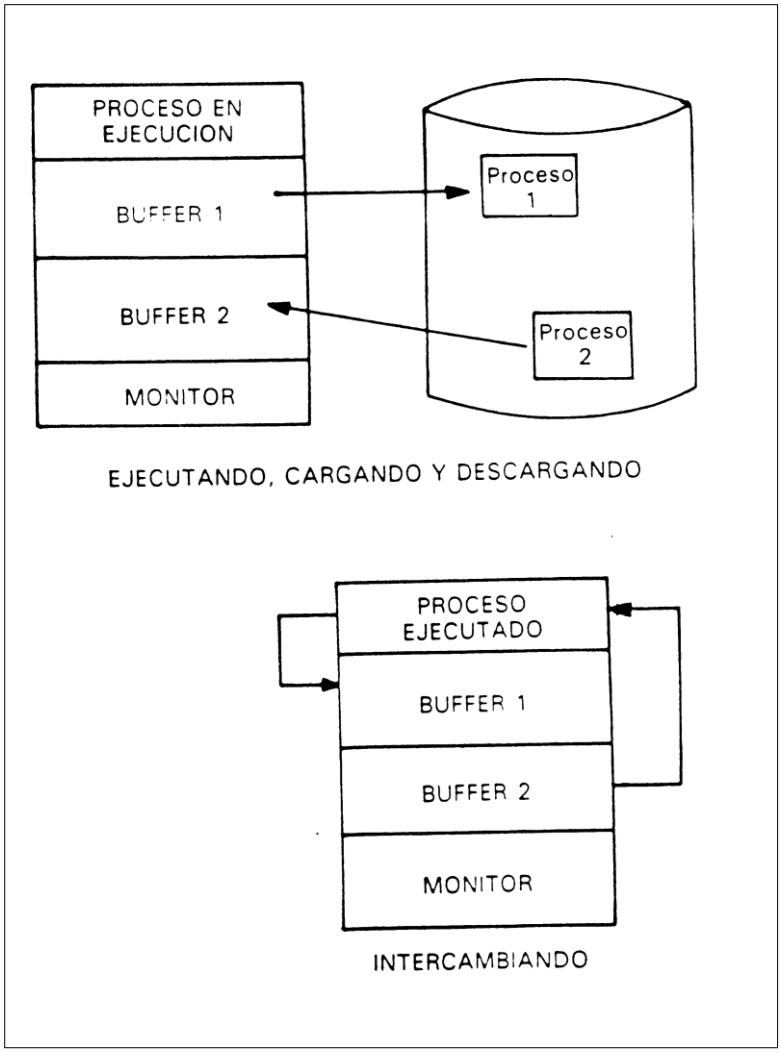
Como en la memoria sólo puede encontrarse un proceso en un instante determinado, cuando este termina es necesario sacarlo hacia un soporte externo y luego introducir el siguiente en ejecución. Por tanto vemos que sólo las operaciones de E/S consumen gran parte del tiempo y la CPU mientras permanece ociosa.

La técnica del swapping o intercambio consiste en disponer de una memoria auxiliar (buffer) donde se descarga momentáneamente el proceso saliente y donde se carga el siguiente proceso a ejecutar.

Esta memoria auxiliar puede ser un disco de acceso rápido, ya que hay que conseguir tiempos de intercambio menores que los de ejecución. Este tiempo de intercambio es proporcional a la memoria intercambiada por tanto es necesario saber que parte del programa realiza tal proceso.



Otra forma de reducir el tiempo de intercambio es realizarlo mientras se está ejecutando, es decir, de forma solapada con la ejecución utilizando para ello dos buffers, uno para descargar y otro para ejecutar.



3. ASIGNACIÓN PARTICIONADA

Una forma de conseguir la multiprogramación consiste en dividir la memoria disponible en partes, a cada una de las cuales se le asignará un proceso o partes del mismo. Si todas las partes del programa se cargan juntas se dice que la asignación es contigua y por el contrario si el programa está grabado en diferentes espacios, se dice que su asignación es no contigua.

De **asignación contigua** veremos dos modelos:

- Particiones fijas.

- Particiones variables.

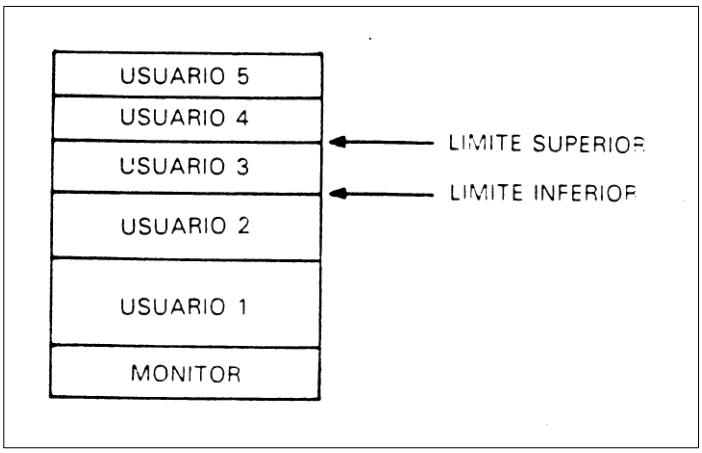
En cuanto a la **asignación no contigua**:

- Paginación.

- Segmentación.

Uno de los problemas a los que hay que enfrentarse cuando en la memoria existen varios programas es la posibilidad de proteger el espacio de memoria que ocupa cada uno de ellos.

Una posibilidad es la de utilizar dos registros que contienen los límites superior e inferior de las direcciones.

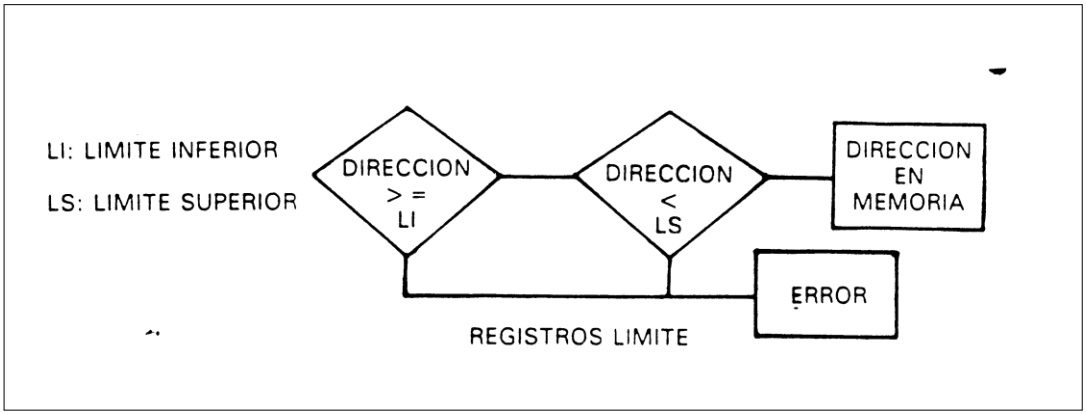


Éstos a su vez se pueden definir de dos maneras:

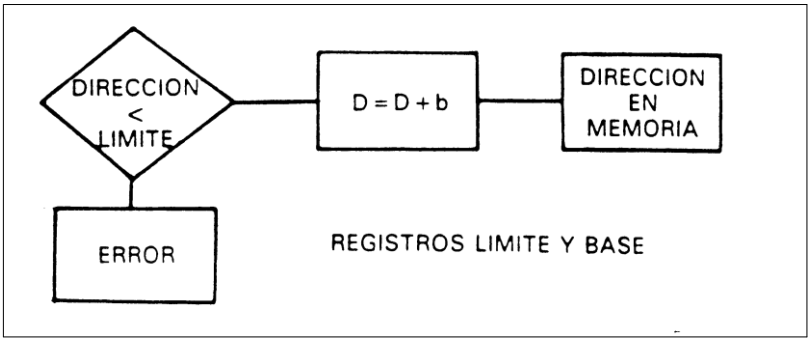
Registros límites: Son los valores de las direcciones físicas mayores y menores ocupadas por un programa.

Registros base y límite: Son los valores de la dirección física más baja y del número de direcciones que contiene el programa.

Los registros límites exigen la reubicación estática en el momento de la carga como muestra la siguiente figura:



En los registros base y límite cada dirección válida ha de ser menor que el límite y es reubicada dinámicamente sumándole el valor del registro base tal y conforme se indica a continuación.



Otro método de dirección consiste en grabar los derechos de acceso de cada programa en la propia memoria, Este método se basa en la misma idea de división en dos zonas de la memoria para un solo usuario, añadiendo unos bits para la identidad del propietario.

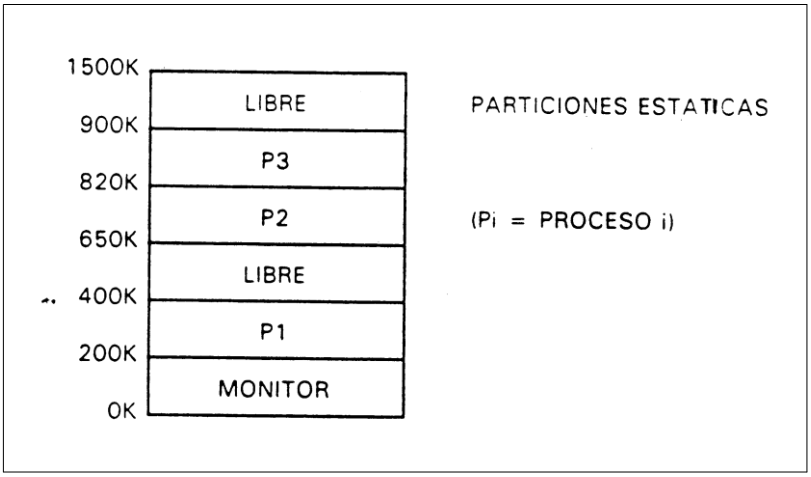
4. PARTICIÓN ESTÁTICA

La memoria principal se divide en un conjunto de **particiones fijas** durante la generación del sistema. Un proceso se puede cargar en una partición de mayor o igual tamaño.

El tamaño se ajusta al tamaño de los procesos que más frecuentemente utilice el sistema.

El número de particiones se hará en función de la capacidad de la memoria y el número de procesos que vayan a ejecutarse a la vez, denominándose dicho número *grado de multiprogramación.*

Una vez en marcha el sistema se puede obtener información sobre el número y tamaño de procesos que con mayor frecuencia se utilizan, estadística que debe mejorar la eficacia en la división de la memoria.



Puede darse el caso que existan regiones sin utilizar, por ejemplo, demasiado pequeñas para los procesos existentes, denominándose a éstas *fragmentación externa*.

El estado actual y el estado de las particiones normalmente se recopilan en una tabla llamada PDT o tabla de descripción de particiones.

NUMERO PARTICION	BASE	TAMAÑO	ESTADO
0	0	200K	ASIGNADA
1	200	200K	ASIGNADA
2	400	250K	LIBRE
3	650	170K	ASIGNADA
4	820	80K	ASIGNADA
5	900	600K	LIBRE

Tabla de descripción de particiones (PDT)

Cuando se utiliza una partición, sólo el campo correspondiente a su estado varía, siendo los demás campos estáticos.

Cuando un proceso es activado, el S.O. intenta asignar una partición de la memoria que esté libre y sea de tamaño suficiente, para ello consulta a la PDT y si encuentra una partición que corresponde con los requerimientos de programa, cambia el estado de libre a asignada cargando el programa en la partición encontrada.

Para que el S.O. sepa qué proceso se encuentra en cada partición en el momento de la carga, se graba en el bloque de control de proceso PCB del proceso determinado la partición asignada.

Cuando un proceso es descargado de la memoria se toma del PCB el número de la partición donde se encontraba y se cambia el estado de la misma en la PDT colocándolo en libre para futuras asignaciones.

Ventajas de las particiones fijas: su sencillez de implementación y la poca sobrecarga del sistema operativo.

Desventajas: empleo ineficiente de la memoria debido a la *fragmentación interna* (cualquier programa que se carga ocupa una partición completa y se malgasta el espacio interno sobrante) y el número de procesos activos es fijo.

4.1. Estrategias de asignación

Cuando se trabaja en multiprogramación normalmente suelen existir gran cantidad de procesos que hay que grabar en pocas particiones; por ello es muy conveniente seguir alguna estrategia para que la asignación de una partición a un programa sea lo más beneficiosa posible en vistas a conseguir un óptimo rendimiento del sistema.

Con *particiones del mismo tamaño*, la ubicación de un proceso en memoria es trivial. Mientras haya alguna partición libre, puede cargarse un proceso en esa partición. Puesto que todas son iguales, no importa la partición que se use.

Con *particiones de distintos tamaños*, hay dos maneras de asignar los procesos a las particiones:

- A través de una sola cola de trabajos peticionarios.
- Mediante el concurso de varias colas.

En el primer caso se concede la primera partición libre que sea de igual o mayor tamaño que el requerido por dicho proceso.

En la otra forma se busca la partición que mejor se adapte al tamaño del proceso.

Si tomamos sólo las particiones que están sin asignar, el algoritmo se llama *el disponible que mejor se adapta* (*best available fit*). Si analizamos todas las particiones estamos ante el algoritmo llamado sólo *el que mejor se adapta* (*best fit*).

Cuando existen varias colas se suele tener una cola por partición y cada trabajo entra en una cola según su tamaño. Una vez allí sólo tiene que esperar su turno para su asignación a la partición correspondiente. Cada cola se gestiona por separado.

4.2. Carga y descarga de procesos

Se llama **carga dinámica** al procedimiento por el cual se retiran procesos que están suspendidos para dar entrada a otros procesos.

Para elegir la “víctima”, se tienen en cuenta la prioridad, el tiempo que lleva en memoria, etc.

En el proceso de descarga hay que tener en cuenta si la relación entre la partición que se asigna y el proceso asignado es fija o variable. Si es fija se ahorra tiempo de buscar una nueva partición, si por el contrario se hace de forma dinámica, el proceso se puede ampliar para añadir nuevos procesos a la ejecución.

4.3. Reubicación

La capacidad de cargar y ejecutar programas en un lugar arbitrario se conoce con el término de programa reubicable y la idea consiste en traducir las direcciones que conoce el programador por las verdaderas direcciones que el programa ocupa en la memoria.

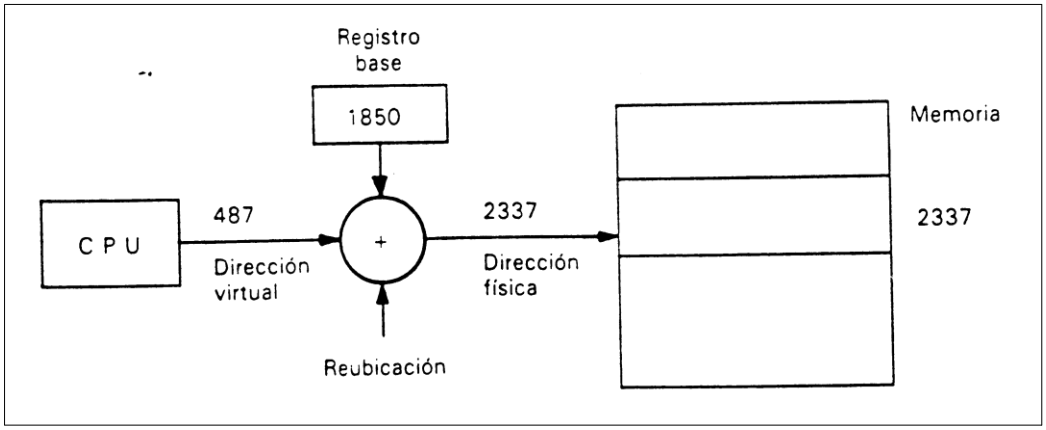
Esta asignación puede hacerse en el momento de la carga, en el momento de la compilación o en el momento de la ejecución.

Aunque las direcciones de memoria empiecen en 0, los programas se graban a partir de la dirección límite que marca la división entre el S.O. y la zona del usuario. Si por alguna ampliación del S.O. cambia el límite tendríamos que volver a compilar el programa reasignado nuevas direcciones.

Para evitar esta recompilación se suele retrasar la asignación de direcciones físicas hasta el momento de carga, conociéndose este procedimiento como **reubicación estática**, es decir no poder cambiar el límite de direcciones durante la ejecución.

Una solución para este problema consiste en la **reubicación dinámica** que permite traducir las direcciones virtuales en físicas en tiempo de ejecución.

Para llevar a cabo dicha traducción es necesario alguna asistencia del hardware. El registro límite se denomina ahora *registro base* y su valor se añade a toda la dirección generada por el proceso de usuario cuando se lleva a memoria.



Este tipo de reubicación permite mover un proceso parcialmente ejecutado desde una área de memoria a otra, lo cual resulta provechoso para una utilización más racional de la memoria ya que permite un particionamiento dinámico de la misma.

4.4. Ventajas y desventajas

Esta forma de particionar la memoria es la forma más sencilla de gestión en multiprogramación; es la forma idónea para procesos de tamaño y características conocidas. Sin embargo es inflexible para estructuras que impliquen un crecimiento dinámico.

La fragmentación interna puede ser muy alta con el consiguiente desaprovechamiento de buena parte de la memoria.

El uso de este tipo de partición es casi nulo hoy en día.

5. PARTICIONES DINAMICAS

En un sistema de particiones variables o dinámicas la descomposición en zonas no se define de forma permanente, sino que se va redefiniendo cada vez que un programa termina. Las particiones son variables en número y longitud. Las particiones se van creando dinámicamente, de forma que cada proceso se carga en una partición de exactamente el mismo tamaño que el proceso.

Al principio, se van creando particiones del tamaño del programa solicitante hasta que se llena la memoria o hasta que el programa que tiene que entrar es mayor que el fragmento que queda libre.

Después, a medida que salen los procesos de la memoria dejando espacio libre se crean zonas en dicho espacio, del tamaño de los procesos entrantes. En cualquier caso ningún programa ocupa más espacio del que necesita, por tanto sólo existe fragmentación externa (con el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae).

En principio, con esta gestión no existe límite para el número de particiones.

Para controlar las partes libres y asignadas, el S.O. mantiene una tabla de descripción de particiones. Cada vez que una partición es creada se introduce en la tabla base, el tamaño y el estado asignado.

Este tipo de gestión de memoria se denomina MVT (Multiprogramación con un número variable de tareas). El soporte hardware que se necesita es al igual que el de asignación estática, dos registros para contener el límite superior e inferior de la memoria asignada a cada proceso.

Ventajas: uso más eficiente de la memoria y no hay fragmentación interna.

Desventajas: uso del procesador para la compactación de memoria.

5.1. Algoritmos de planificación

Para lograr una asignación eficaz, es necesario seguir algún tipo de algoritmo de selección de particiones.

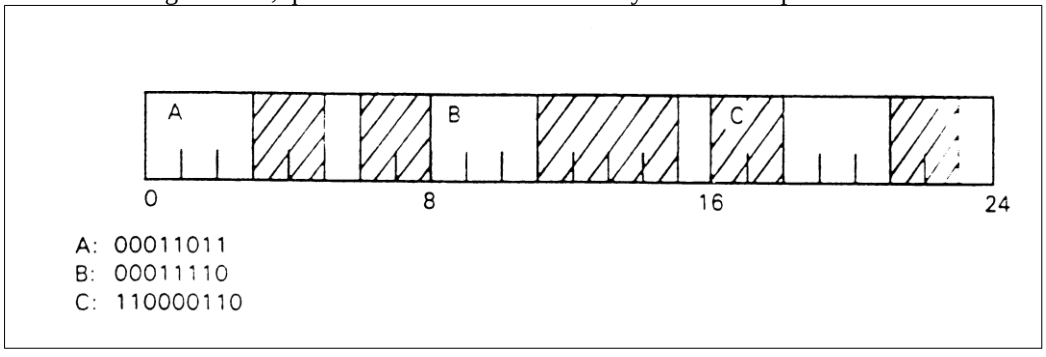
Un primer proceso podría consistir en elegir la primera partición libre que sea de igual o mayor tamaño.

Uno mejor seria rastrear toda la memoria, mediante la PDT, y buscar aquella partición de entre las libres, que al cargar el proceso deje menor hueco.

El peor método sería al igual que el caso anterior pero con el mayor hueco posible.

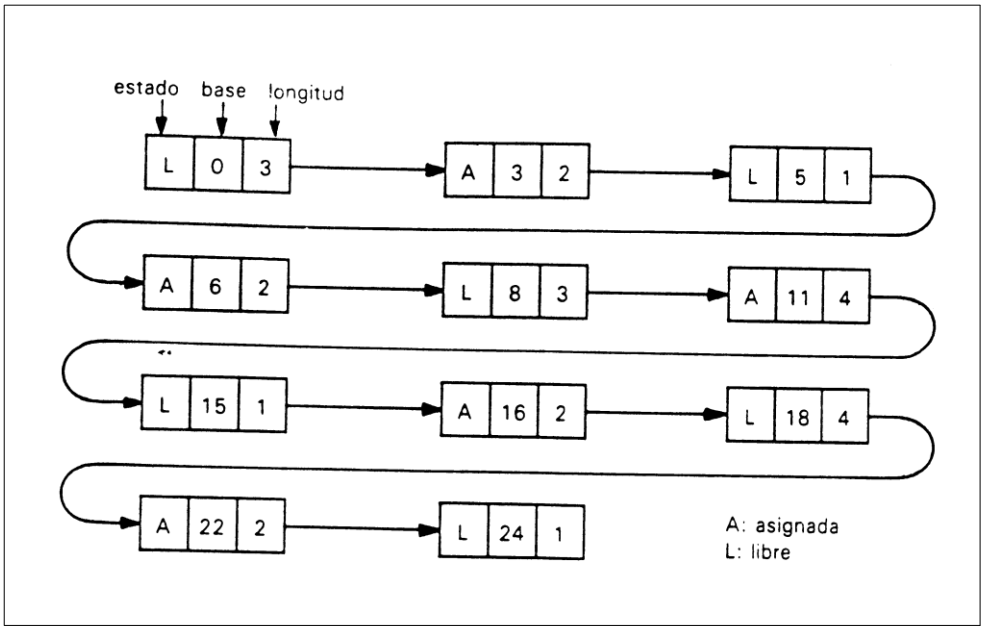
En términos generales, existen tres maneras de llevar a cabo el *control del uso de la memoria*: mapa de bits, listas y sistemas compañeros.

Con un **mapa de bits**, la memoria se divide en unidades de asignación de un tamaño determinado y a cada unidad se le asigna un bit, que será 0 si la unidad está libre y 1 si está ocupada.



En general este método es de búsqueda lenta y no muy utilizado.

El segundo método consiste en conservar en memoria una **lista enlazada de segmentos** asignados y libres. Si, por ejemplo, les llamamos "A" a los asignados y "L" a los libres, nos da la lista de la siguiente figura.



El **sistema compañero** consiste en ir dividiendo la memoria en fragmentos iguales, de manera que la asignación y desasignación son más rápidas, pues cuando se busca un bloque determinado sólo se analizan los de tamaños más próximos.

5.2. Problemas y soluciones

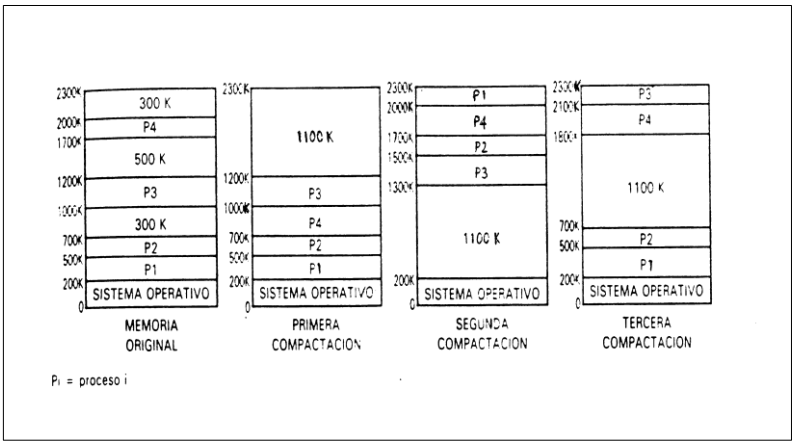
- Si tenemos un hueco de 3.500 bytes y debemos cargar un proceso de 3.495 bytes, obtendremos un hueco de 5 bytes que no podremos utilizar y además gastaremos memoria para anotar la dirección de dicho hueco.

La solución pasa por crear asignaciones de tamaño fijo, produciendo así una pequeña fragmentación interna, pero evitando el problema antes planteado.

- Si tenemos varios huecos no contiguos pero cada uno de ellos menor que el tamaño que el proceso solicita.

Esto se puede resolver mediante un proceso denominado **compactación,** que consiste en reunir todos los huecos en uno sólo. Será posible sólo si la reubicación es dinámica en el momento de la ejecución, utilizando registros base y límite.

La siguiente figura muestra la compactación de una memoria por tres métodos.



6. PAGINACIÓN

Cuando un proceso no entraba en memoria porque el tamaño de los huecos era menor que el tamaño de dicho proceso proponíamos la solución de la compactación, pero dicha solución requiere tiempo y no siempre es viable.

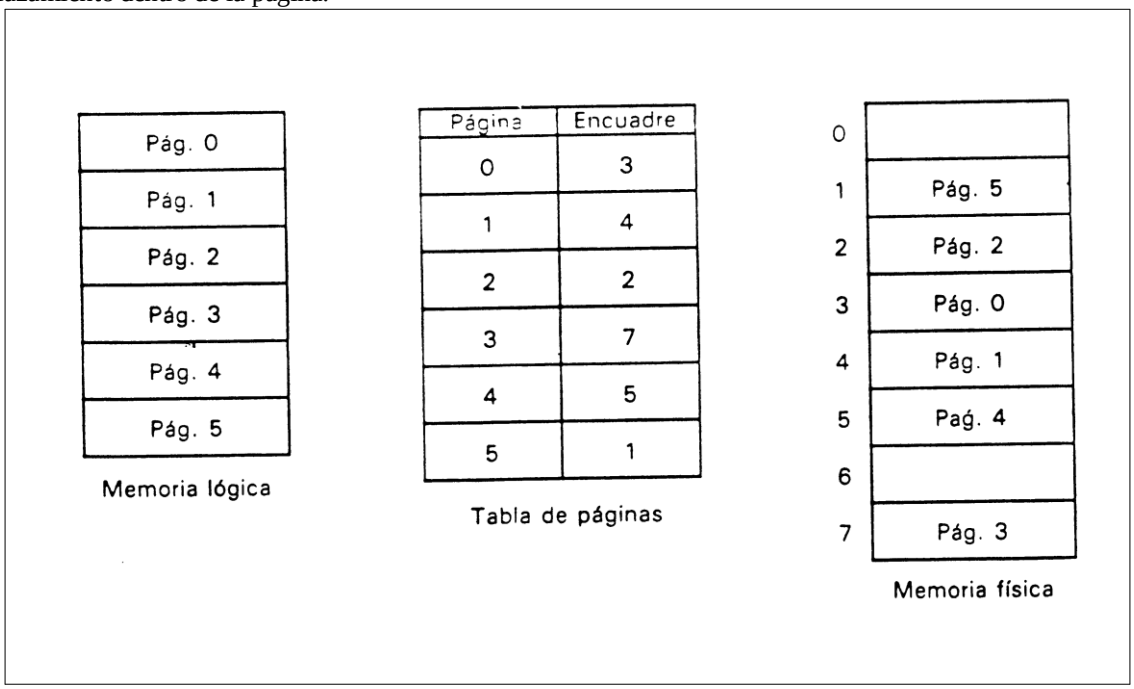
La gestión de memoria paginada es una gestión basada en la asignación de memoria física no contigua, es decir, que las partes de un proceso lógico puedan estar situadas en áreas no contiguas de la memoria física.

La memoria principal se divide en un conjunto de **marcos** de igual tamaño. Cada proceso se divide en una serie de **páginas** del mismo tamaño que los marcos. Un proceso se carga situando todas sus páginas en marcos libres pero no necesariamente contiguos.

De esta manera, la fragmentación interna se reduce sólo a una fracción de la última página del proceso. Además, no hay fragmentación externa.

Dado que cada página se corresponde con su bloque, el desplazamiento dentro de la página será el mismo que dentro del bloque.

Ya no será suficiente utilizar un simple registro base. En su lugar, el sistema operativo mantiene una **tabla de páginas** para cada proceso. La tabla de páginas muestra la posición del marco de cada página del proceso. Dentro del programa, cada dirección lógica constará de un número de página y de un desplazamiento dentro de la página.



El tamaño de las páginas viene definido por el hardware y suele darse como potencias de 2.

6.1. Planificación de trabajos

Cuando se solicita cargar un proceso de tamaño *t* el S.O. debe asignar *e* bloques, donde *e* se obtiene dividiendo *t* entre el tamaño de página y sumándole 1. Si, por ejemplo, la página es de 32 bytes y tenemos la dirección 200, significa que se necesitan 7 bloques: int (200/32) +1 (int es la parte entera).

Como se ve, el S.O. asigna un número entero de bloques. Si el tamaño del proceso no es múltiplo entero del tamaño de página, el último bloque no se utiliza en su totalidad, produciéndose una fragmentación, que es siempre menor que el tamaño de página.

Por tanto, una vez calculado el número de bloques, se procede a cargar el proceso sobre los mismos construyendo la tabla de páginas de dicho proceso.

La implementación de la tabla de páginas se puede hacer de diversas formas, dependiendo en general de su tamaño y la rapidez de acceso que se requiera.

Es posible declarar mediante bits asociados a cada página, el acceso de la misma, es decir si es de lectura o escritura y por tanto compartida por varios programas.

7. SEGMENTACIÓN

En este caso, cada proceso se divide en una serie de **segmentos**. No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. Un proceso se carga situando todos sus segmentos en particiones dinámicas que no tienen por qué ser contiguas.

Se puede definir un segmento como un agrupamiento lógico de la información. Los segmentos no tienen por qué ser del mismo tamaño, ni contiguos, aunque sí la información que contienen.

Mientras que la paginación es transparente al programador, la segmentación es generalmente visible y se proporciona como una comodidad para la organización de los programas y datos.

El usuario indicará la *dirección lógica* mediante dos números, el número del segmento y el número del desplazamiento.

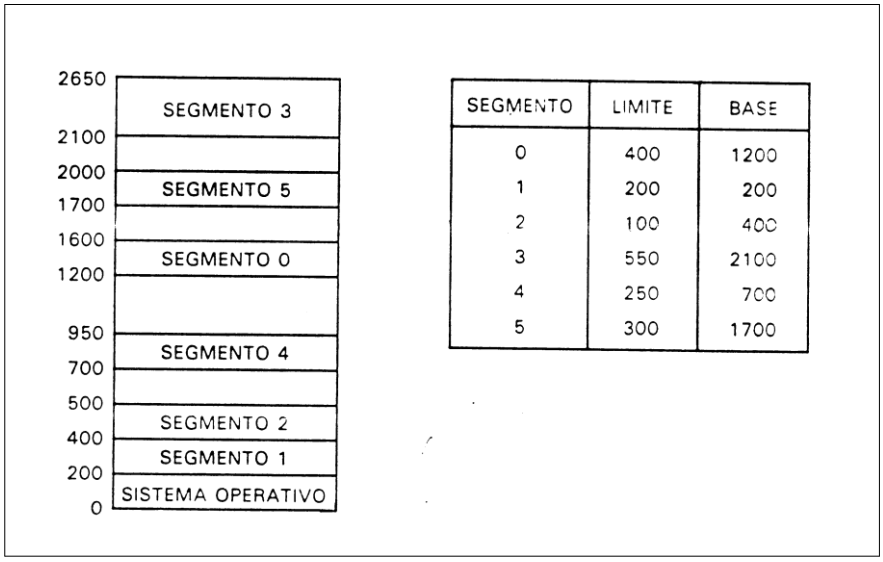
No existirá fragmentación interna, pero sí se necesita compactación.

Los segmentos requieren reubicación dinámica y formas de protección avanzadas.

7.1. Tabla de segmentos

Cuando se quiere cargar un proceso de forma segmentada, el S.O. asigna memoria para los segmentos requeridos. Dichos segmentos podrán ir físicamente separados y por ello el sistema tendrá que anotar la base o comienzo de cada segmento, así como su tamaño. El conjunto de estos pares de números (base y límite) formará una tabla llamada **tabla de segmentos**.

Mediante dicha tabla se traducirán los números de segmento y desplazamiento a una *dirección física*: el número de segmento se utiliza como índice en la tabla de donde se obtendrán la base y el límite del segmento.



SEGMENTO	LIMITE	BASE
0	400	1200
1	200	200
2	100	400
3	550	2100
4	250	700
5	300	1700

Para la *traducción de direcciones* hay que dar los siguientes pasos:

1. Extraer el número de segmento (de los n bits más significativos de la dirección lógica).
2. Emplear el número de segmento como índice en la tabla de segmentos del proceso para encontrar la dirección física de comienzo del segmento.
3. Comparar el desplazamiento (expresado por los m bits menos significativos) con la longitud del segmento. Si el desplazamiento es mayor que la longitud, la dirección no es válida.
4. La dirección física buscada es la suma de la dirección física de comienzo del segmento más el desplazamiento.

7.2. Protección

La forma natural de protección se apoya en los registros base y límite. Estas se complementan con los bits que indican los derechos de acceso, así se les protege de las funciones no permitidas.

8. SISTEMAS COMBINADOS

Desde los años 60 se han construido sistemas que combinan la paginación y la segmentación. De estas formas de combinación surgen la **paginación segmentada** y la **segmentación paginada**.

En un sistema con paginación y segmentación combinadas, el espacio de direcciones de un usuario se divide en varios segmentos según criterio del programador. Cada segmento se vuelve a dividir en páginas de tamaño fijo, que tienen la misma longitud que un marco de memoria principal. Si el segmento tiene menor longitud que la página, dicho segmento ocupará sólo una página. Desde el punto de vista del programador, una dirección lógica también está formada por un número de segmento y un desplazamiento en el segmento. Desde el punto de vista del sistema, el desplazamiento del segmento se ve como un número de página dentro del segmento y un desplazamiento dentro de la página.

9. BIBLIOGRAFÍA

Stallings, W. ***Sistemas operativos*** Prentice Hall, 2ed., 1997

Tanenbaum, Andrew S. ***Sistemas operativos: Diseño e implementación*** Prentice Hall, 1990