Arquitectura y Sistemas Operativos

UTN MAR DEL PLATA

GESTIÓN DE MEMORIA

Requisitos de la Gestión de Memoria

Requisitos que la gestión de memoria debe satisfacer:

- Reubicación
- Protección
- Compartición
- Organización lógica
- Organización fija.

REUBICACIÓN

La memoria principal disponible se comparte generalmente entre varios procesos, por lo que es bueno poder intercambiar los procesos en la memoria principal para maximizar la utilización del procesador proporcionando un gran número de procesos para la ejecución.

Una vez que un programa se lleva a la memoria secundaria (disco), sería bastante limitante colocarlo en la misma región de memoria principal donde se hallaba anteriormente, cuando éste se trae de nuevo a memoria. Podría ser necesario **reubicar** el proceso es un área de memoria diferente.

PROTECCIÓN

Cada proceso debe protegerse contra interferencias no deseadas por parte de otros procesos, sean accidentales o intencionadas. 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 escritura. 2

Normalmente, un proceso de usuario no puede acceder a cualquier porción del sistema operativo, ni al código ni a los datos, es decir, un programa de un proceso no puede saltar a una instrucción de otro proceso. El procesador debe ser capaz de abortar tales instrucciones en el punto de ejecución. 1

COMPARTICIÓN

Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal. Procesos que estén cooperando en la misma tarea podrían necesitar compartir el acceso a la misma estructura de datos.

Por tanto, el sistema de gestión de la memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial.

Organización Lógica

La memoria del computador se organiza como un espacio de almacenamiento lineal o unidimensional, compuesto por una secuencia de bytes o palabras. A nivel físico la memoria secundaria esta organizada de manera similar.
Esta organización no se corresponde a la forma en la cual los programas se construyen normalmente.

Es una forma de organizar la memoria (Lineal, Unidimensional o compuesta) y de asignar esa memoria.

Organización Lógica

La mayoría de los programas se construyen en módulos. Si el S.O. y el hardware del computador pueden tratar de forma efectiva los programas y los datos en la forma de módulo, entonces se logran varias ventajas:

1. Los módulos se pueden escribir y compilar independientemente.
2. Con una sobrecarga adicional modesta, se puede proporcionar diferentes grados de protección a los módulos (solo lectura, solo ejecución).
3. Es posible introducir mecanismos por los cuales 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.

La herramienta que más adecuadamente satisface estos requisitos es la segmentación.

Organización Física

La memoria del computador se divide en al menos dos niveles:

- Memoria principal.
- Memoria secundaria.

La memoria principal proporciona un acceso rápido a un coste relativamente alto. Además la memoria principal es volátil, es decir, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y más barata que la memoria principal y generalmente no es volátil.

La memoria secundaria de larga capacidad puede proporcionar almacenamiento para programas y datos a largo plazo, mientras que una memoria principal más pequeñas contiene programación y datos actualmente en uso.

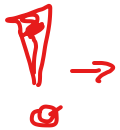
Organización Física

La organización del flujo de información entre la memoria principal y secundaria supone una de las preocupaciones principales del sistema. La responsabilidad para este flujo podría asignarse a cada programador en particular, pero no es practicable o deseable por dos motivos:

El programador no tiene que estar a cargo del flujo de informacion entre la memoria principal y la secundaria por dos razones :

Organización Física

1. La memoria principal disponible para un programa más sus datos podría ser insuficiente. En este caso, el programador debería utilizar una técnica conocida como **superposición (overlaying)**, en la cual los datos y programas 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. Esto malgasta tiempo del programador.
2. En un entorno multiprogramado, el programador no conoce en tiempo de codificación cuánto espacio estará disponible o dónde se localizará dicho espacio.



Por lo tanto, está claro que la tarea de mover la información entre los dos niveles, debería ser responsabilidad del sistema.

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 casi todos los sistemas multiprogramados modernos, esto implica el uso de un esquema sofisticado denominado memoria virtual. Dicha memoria se basa en una o ambas de las siguientes técnicas básicas: segmentación y paginación.

Antes de ver éstas técnicas, veremos algunas más sencillas que no utilizan memoria virtual, ellas son particionamiento, ahora obsoleto y paginación simple y segmentación simple.

Las técnicas utilizados a continuación son mas sencillas pero no usan memoria virtual y están obsoletos:

son : PARTICIONAMIENTO, PAGINACION SIMPLE Y SEGMENTACION SIMPLE

Particionamiento fijo

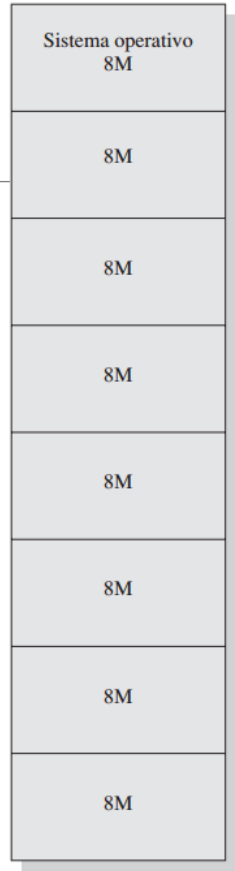
En la mayoría de los esquemas para la gestión de la memoria, se puede asumir que el sistema operativo ocupa alguna porción fija de la memoria principal y que el resto de la memoria principal está disponible para múltiples procesos. El esquema más simple para gestionar la memoria disponible es repartirla en regiones con límites fijos.

Particionamiento fijo – Tamaños de partición

Veremos dos alternativas. Una posibilidad consiste en hacer uso de particiones del mismo tamaño.
Cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la 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 a **swap** a un proceso de cualquiera de las particiones y cargar otro proceso.

interrumpe un proceso bloqueado y lo manda a la memoria secundaria

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

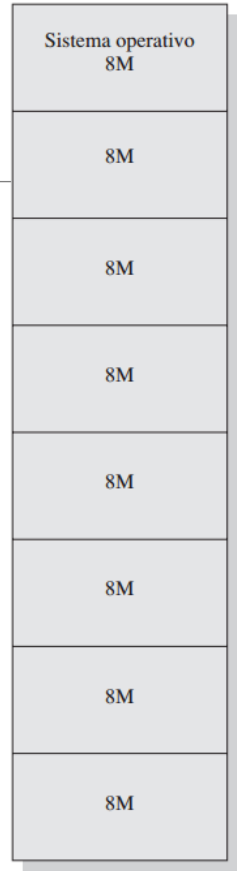


(a) Participaciones de igual tamaño

Particionamiento fijo – Tamaños de partición

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

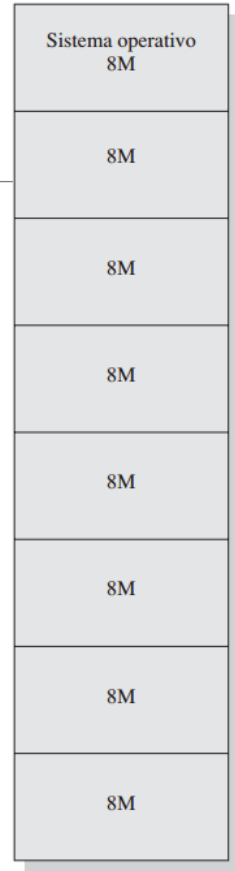
- Un programa podría ser demasiado grande para caber en una partición. En este caso, el programador debe diseñar el programa con el uso de **overlays**, de forma que solo se necesite una porción del programa en memoria principal en un momento dado. Cuando se necesita un módulo que no está presente, el programa debe cargar dicho módulo en la partición del programa, superponiendo a cualquier programa o datos que haya allí.



(a) Participaciones de igual tamaño

Particionamiento fijo – Tamaños de partición

- La utilización de la memoria principal es extremadamente ineficiente. Cualquier programa ocupa una partición entera. Por ejemplo, podría haber un programa cuya longitud sea 2 Mb, como tratamos de particiones fijas ocuparía una partición de, en este caso, 8 Mb. Se puede observar que hay espacio interno malgastado, debido a que el bloque de datos cargado es menor que la partición. Este fenómeno se lo conoce como **fragmentación interna.**

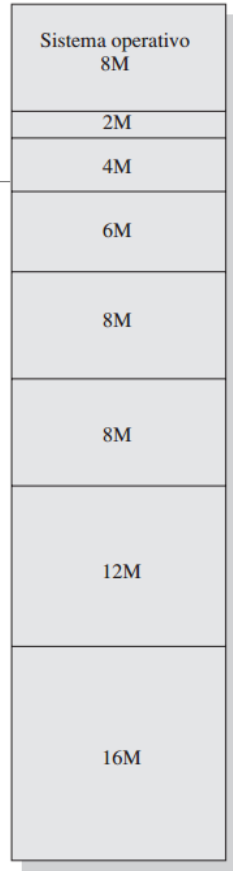


(a) Participaciones de igual tamaño

Particionamiento fijo – Tamaños de partición

Ambos problemas se pueden mejorar, aunque no resolver, utilizando particionamiento de tamaños diferentes. Siguiendo este ejemplo, un programa de 16 Mb se puede acomodar sin **overlays**.

Las particiones más pequeñas de 8 Mb permiten que los programas más pequeños se puedan acomodar disminuyendo la fragmentación interna.



(b) Participaciones de distinto tamaño

Particionamiento fijo – Algoritmos 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.

Si todas las particiones se encuentran ocupadas por procesos que no están listos para ejecutar, uno de ellos es llevado al disco para dejar espacio para un nuevo proceso.

¿Cuál de los procesos es el que se lleva a disco? 🤔

Es una decisión del planificador.

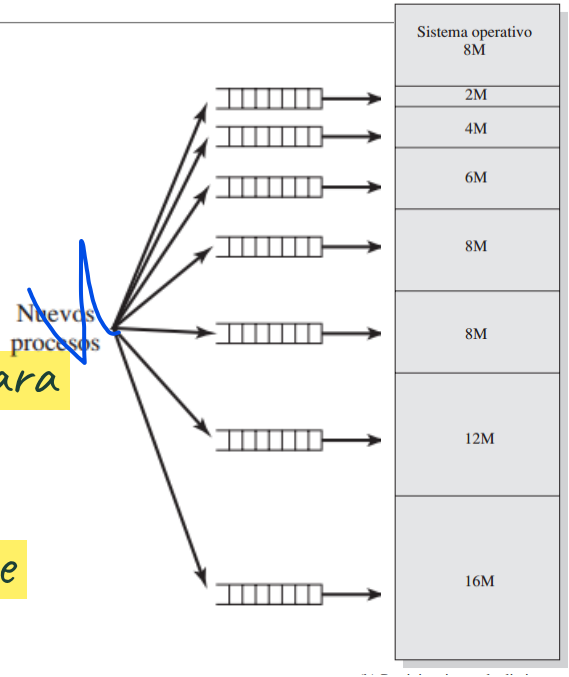
Particionamiento fijo – Algoritmos de ubicación

Con **particionamiento de diferente tamaño**, hay dos formas posibles de asignar los procesos a las particiones.

La forma más sencilla es asignar cada proceso a la partición más pequeña en la cual cabe.

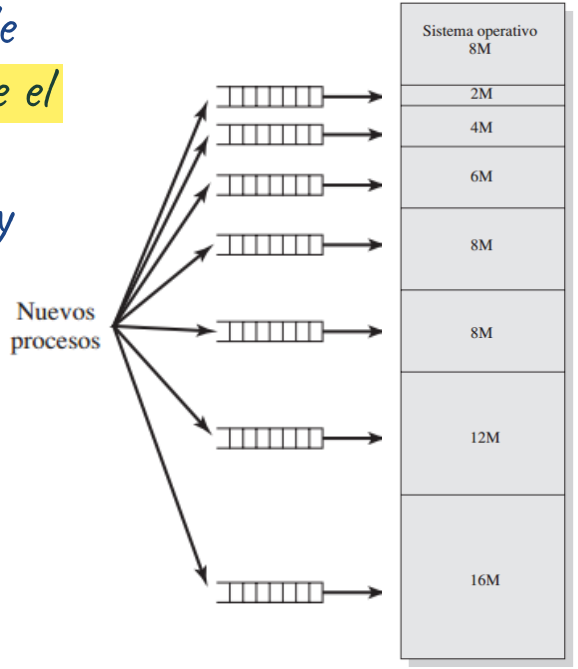
En este caso, se necesita una cola de planificación para cada partición que mantenga los procesos en disco destinados a dicha partición.

La ventaja de esta técnica es que los procesos siempre se asignan de tal forma que se minimiza la fragmentación interna.



Particionamiento fijo – Algoritmos de ubicación

Aunque esta técnica parece óptima desde el punto de vista de una partición individual, no es óptima desde el punto de vista del sistema completo.
Por ejemplo, si se considera un caso en el que no hay procesos con un tamaño entre 12 y 16 Mb, la partición de 16 Mb quedará sin utilizarse.

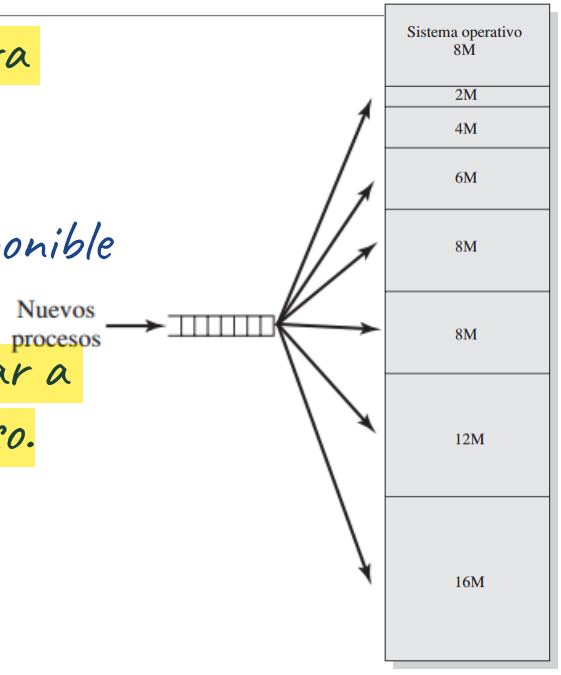


Particionamiento fijo – Algoritmos de ubicación

Una técnica óptima sería emplear una única cola para todos los procesos.

En el momento de cargar un proceso a la memoria principal, se selecciona la partición más pequeña disponible para albergar dicho proceso.

Si todas las particiones están ocupadas, se debe llevar a cano una decisión para enviar a swap a algún proceso.



Particionamiento fijo – Algoritmos de ubicación

El uso de particiones de distinto tamaño proporciona un grado de flexibilidad frente a las particiones fijas.

Se podría decir que los esquemas de particiones fijas son relativamente sencillos y requieren un soporte mínimo por parte del sistema operativo. Sin embargo tiene una serie de desventajas:

- El número de particiones especificadas en tiempo de generación del sistema limita el número de procesos activos del sistema.
- Debido a que los tamaños de las particiones son preestablecidos en tiempo de generación del sistema, puede ser eficiente en entornos donde el requisito de almacenamiento principal de los trabajos es conocido de antemano. Pero eso no ocurre en la mayoría de los casos.

Particionamiento Dinámico

Las particiones son de longitud y número variable.

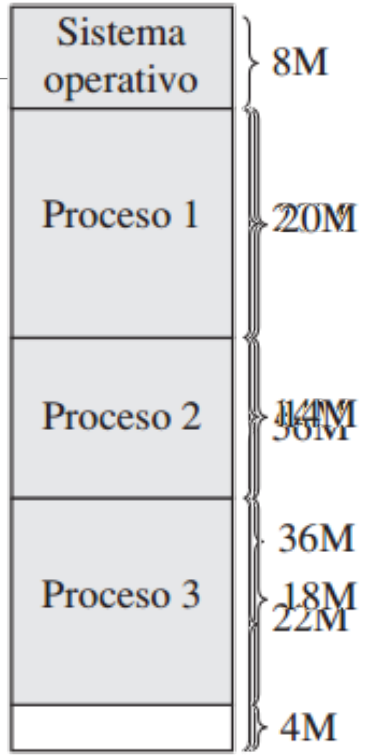
Cuando un proceso llega a la memoria principal, se le asigna exactamente tanta memoria como memoria como requiera y no más.

Un ejemplo:

Inicialmente, la memoria principal esta vacía excepto por el S.O.

Los primeros 3 procesos se cargan justo donde el S.O. finaliza y ocupando el espacio justo.

Esto deja un «hueco» al final demasiado pequeño para un cuarto proceso



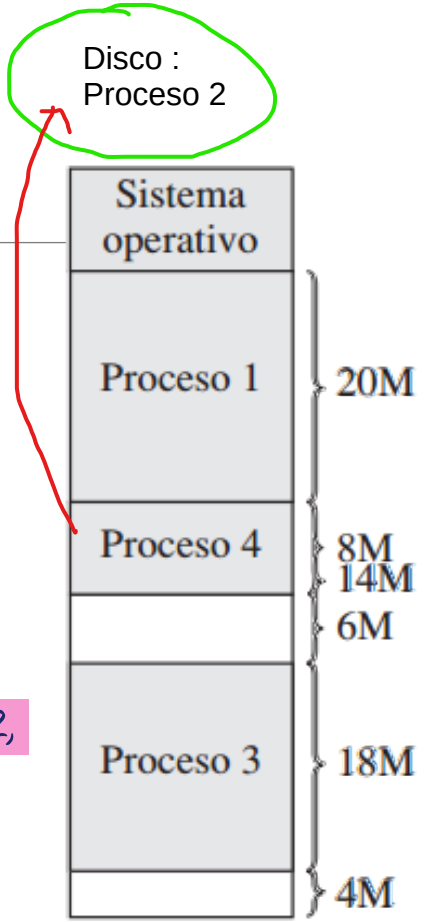
Particionamiento Dinámico

El algún momento, ninguno de los procesos que se encuentra en memoria está disponible.

El S.O. lleva el proceso 2 al disco, lo que deja suficiente espacio para cargar un nuevo proceso.

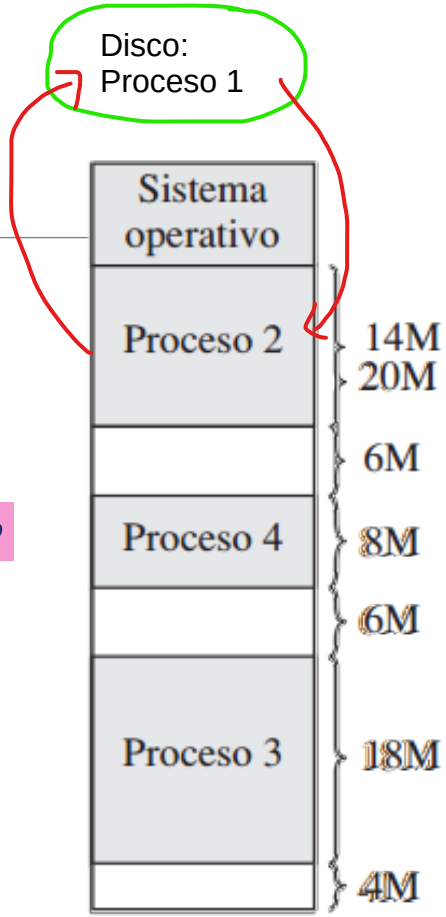
Se carga en memoria principal el proceso 4, debido a que es más pequeño que el proceso 2, se crea otro pequeño hueco.

Posteriormente se alcanza un punto donde ninguno de los procesos de la memoria principal está listo, pero el proceso 2, en estado Suspendido Listo, está disponible



Particionamiento Dinámico

Porque no hay espacio suficiente para el proceso 2, el S.O. lleva a disco el proceso 1.
Se lleva a memoria principal el proceso 2.
Como podemos ver, el método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos pequeños en la memoria.

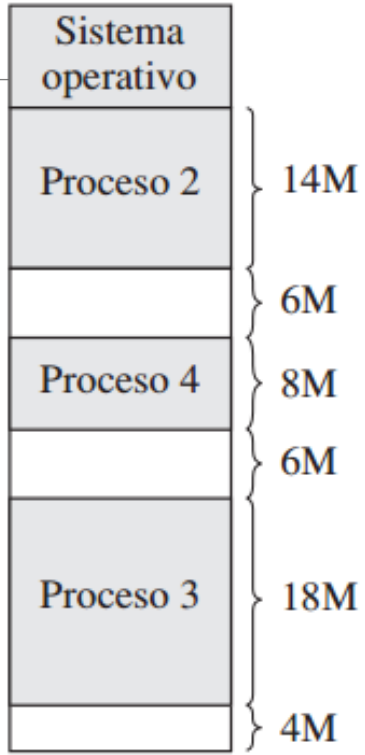


Particionamiento Dinámico

A medida que pasa el tiempo, la memoria se fragmenta cada vez más y la utilización de la memoria decrementa.

Este fenómeno se lo conoce como **fragmentación externa**, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental.

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 bloque.



En pocas palabras unifica todos los espacios libres, esto es una desventaja ya hace un gasto importante de memoria. Porque levanta bloques de memorias completos y los pasa a disco. solamente para eliminar la fragmentacion.

Particionamiento Dinámico – Algoritmo de ubicación

esto

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 a los procesos.

Existen cuatro algoritmos de colocación que puede considerarse:

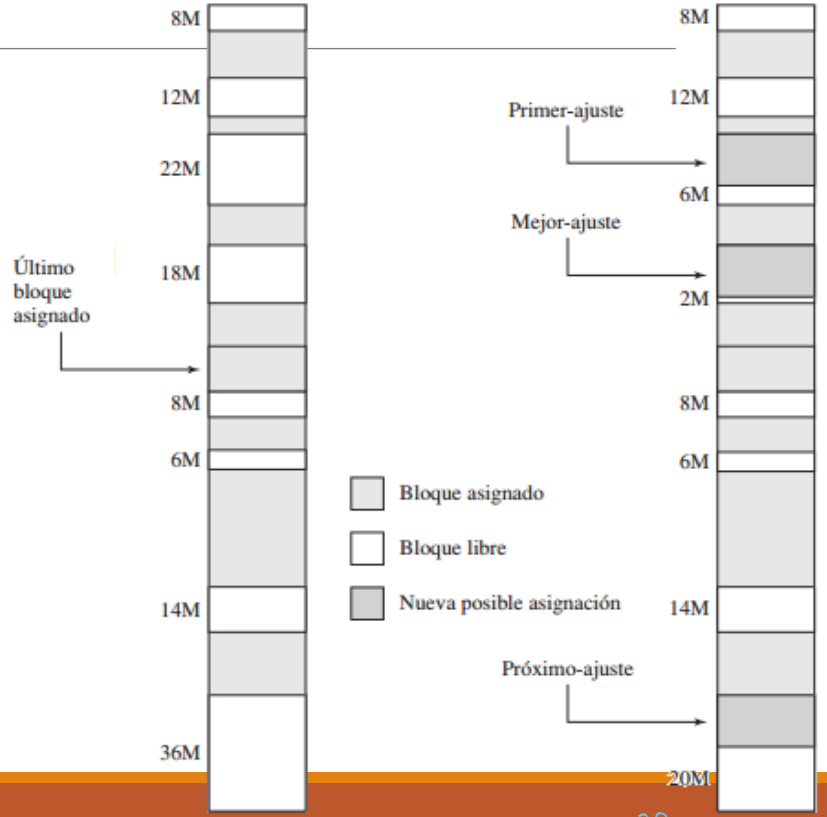
- Mejor-ajuste (best-fit)
- Primer-ajuste (first-fit)
- Siguiente-ajuste (next-fit)
- Peor-ajuste (worst-fit)

Particionamiento Dinámico – Algoritmo de ubicación

- **Mejor-ajuste** escoge el bloque más cercano en tamaño a la petición.
- **Primer-ajuste** comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande.
- **Siguiente-ajuste** comienza a analizar la memoria desde la ultima colocación y elige el siguiente bloque disponible que sea suficientemente grande.
- **Peor-ajuste** escoge el mayor bloque de memoria libre para un proceso.

Particionamiento Dinámico – Algoritmo de ubicación

Un ejemplo de configuración de memoria después de un número de colocaciones e intercambios en disco.
El último bloque que se utilizó fue un bloque de 22 Mb del cual se crea una partición de 14 Mb.
En este ejemplo muestra la diferencia entre los algoritmos, a la hora de satisfacer una petición de asignación de 16 Mb.



Paginación Simple

Todos los anteriores son ineficientes

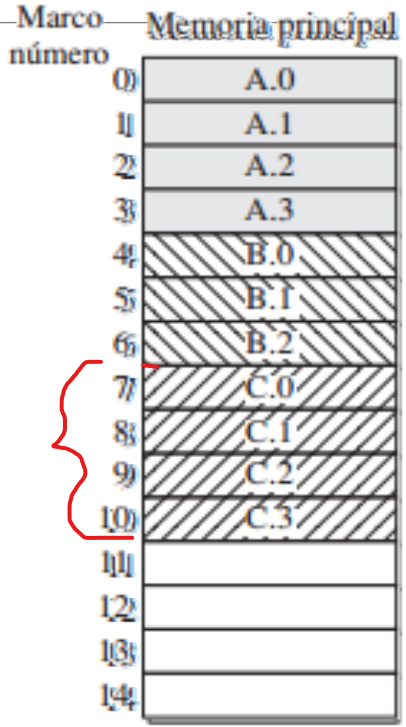
Tanto las particiones de tamaño fijo como variable son ineficientes en el uso de la memoria. Los primeros provocan fragmentación interna y los últimos fragmentación externa.

Supongamos, que la memoria principal se divide en porciones de tamaño fijo relativamente pequeños, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo.

Dichas porciones del procesos, conocidas como **páginas**, se les asigna porciones disponibles de memoria, conocidas como **marcos** o marcos de página.

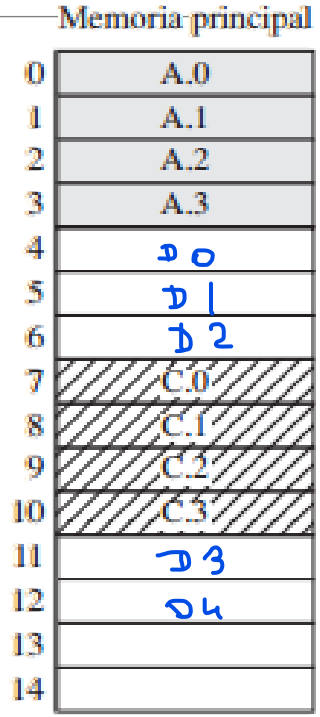
Paginación Simple

Supongamos en un sistema con 15 marcos del mismo tamaño. El S.O. mantiene una lista de marcos libres. El proceso A, almacenado en disco esta formado por cuatro páginas. En el momento de cargar este proceso, el S.O. encuentra cuatro marcos libres y carga las cuatro páginas del proceso A.
El proceso B formado por tres páginas y el proceso C formado por cuatro, se cargan a continuación.



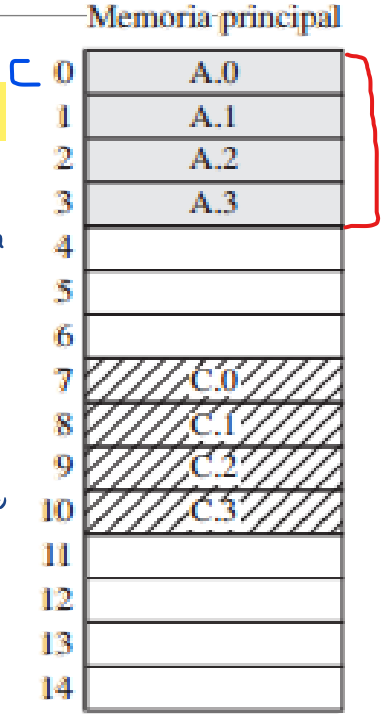
Paginación Simple

En un instante el proceso B se suspende y deja la memoria principal.
El sistema necesita traer a un nuevo proceso, el proceso D que está formado por 5 páginas.
Si vemos este ejemplo no hay 5 marcos contiguos disponibles. ¿Significa que el S.O. no pueda cargar el proceso D?
No, porque se puede utilizar el concepto de dirección lógica.



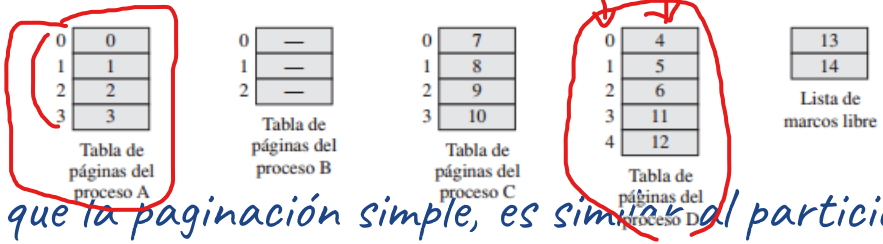
Paginación Simple

El S.O. mantiene una **tabla de páginas** por cada proceso..
La tabla de páginas muestra la ubicación del marco por cada página del proceso.
Dentro del proceso, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página.
Con paginación, la traducción de direcciones lógicas a físicas las realiza el hardware del procesador.
Por lo tanto, presentando una dirección lógica (nro de página, desplazamiento), el procesador utiliza la tabla de páginas para producir una dirección física (nro de marco, desplazamiento).

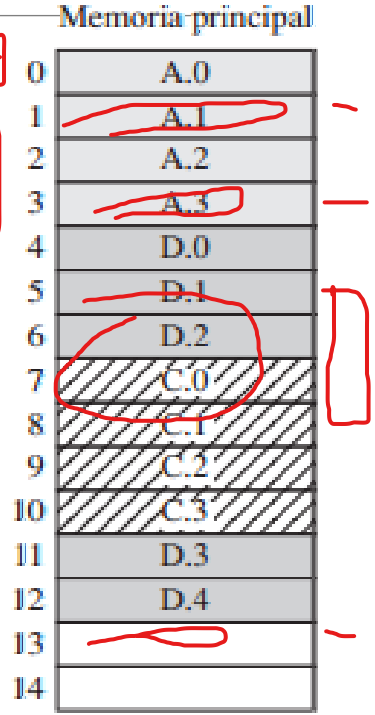


Paginación Simple

Entonces las cinco páginas del proceso D se cargan en los marcos 4, 5, 6, 11 y 12.
Veremos como quedan las diferentes tablas de páginas en este momento dado:



Vemos que la paginación simple, es similar al particionamiento fijo. Las diferencias son que, con la paginación. Las particiones son bastante pequeñas y un programa podría ocupar mas de una partición y no necesariamente contiguas.



Paginación Simple

En resumen, con paginación simple:

- La memoria principal se divide en muchos marcos pequeños de igual tamaño.
- Cada proceso se divide en páginas de igual tamaño.
- Procesos pequeños requieren menos páginas
- Procesos mayores requieren más páginas
- Cuando un proceso se trae a la memoria, todas sus páginas se cargan en los marcos disponibles, y se establece una tabla de páginas.
- Esta técnica resuelve muchos de los problemas inherentes en el particionamiento.

Paginación Simple

Veamos un ejemplo de calculo de direcciones:

Sea un sistema de 12 bits con Paginación Simple. Donde 4 bits están destinados para cantidad de páginas y 8 bits para desplazamiento.

Teniendo un proceso con 7 páginas y la siguiente tabla de páginas:

Calcular las direcciones físicas a partir de las relativas:

a. 321 — D : R (3, 1)

b. 573 → D, R

Nro Página	Nro Frame
0	6
1	7
2	8
3	10
4	13
5	14
6	15

1 BYTES → 0, 1

4 BYTES

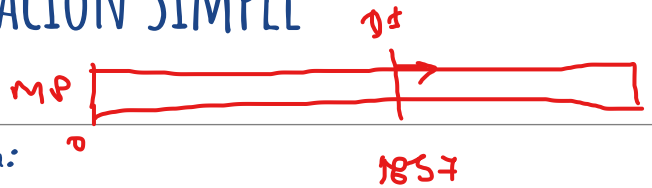
$2^4 = 16$

$2^8 = 256$ BYTES

$321 / 256 = 1,25390625$

$0,25390625 \cdot 256 = 65$

Paginación Simple



Solución:

Esquema paginación simple de 12 bits, de los cuales 4 indican la cantidad máxima de páginas por proceso -> $2^4 = 16$.

Quedan 8 bits para desplazamiento máximo dentro de la página o marco -> $2^8 = 256$, que corresponde al tamaño.

Realizamos una división entera para obtener el nro de página y desplazamiento: 321/256, donde el cociente indica el nro de página y el resto el desplazamiento.

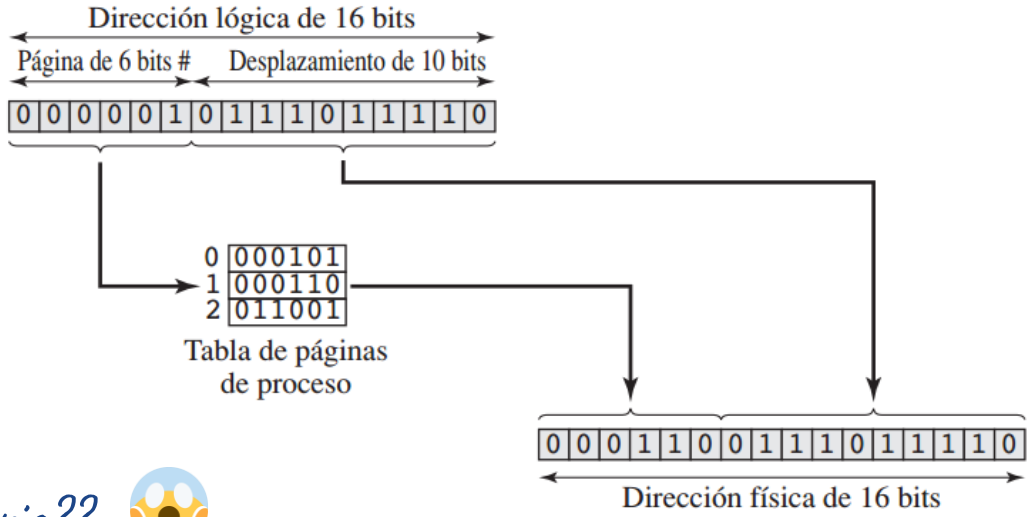
En este caso nro de pagina es 1 y el desplazamiento 65.

Con la tabla de página se obtiene el marco y luego se realiza la siguiente operación: **nroMarco * tamañoMarco + desplazamiento** -> 7*256+65 = 1857 (dirección física relativa).

7 * 256 + 65 = 1857

Paginación Simple

Veremos, entonces, la traducción de direcciones lógicas a físicas:



¿¿En binario?? 😱

Si, también se pueden calcular en binario

Segmentación Simple

Un programa de usuario se puede subdividir utilizando segmentación, en la cual el programa y sus datos asociados se dividen en un número de **segmentos**. No se requiere que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento.

Al igual que la paginación, una dirección lógica está compuesta por dos partes, en este caso, un nro de segmento y un desplazamiento.

Segmentación Simple

Debido al uso de segmentos de distintos tamaño, la segmentación es similar al particionamiento dinámico. Aunque la diferencia comparada con éste último, es que con la segmentación podría ocupar mas de una partición, y éstas no necesitan ser contiguas.
Elimina la fragmentación interna, aunque sufre de externa, pero debido a que el proceso se divide en varias piezas mas pequeñas, la fragmentación externa es mucho menor.

Segmentación Simple

También tenemos una tabla de segmentos, pero a diferencia que en paginación, la tabla contiene una entrada más que corresponde a la longitud o tamaño del segmento.
Al realizar las traducciones de memoria lógica a física, hay que tener en cuenta que el desplazamiento no supere la longitud máxima del segmento, ya que en ese caso no sería una dirección válida.
A continuación, veremos un ejemplo.

Segmentación

Sea un sistema de 14 bits con Segmentación. Donde 4 bits son para segmentos y 10 bits para el tamaño del segmento y desplazamiento dentro de él.
Y la siguiente tabla de segmentos:

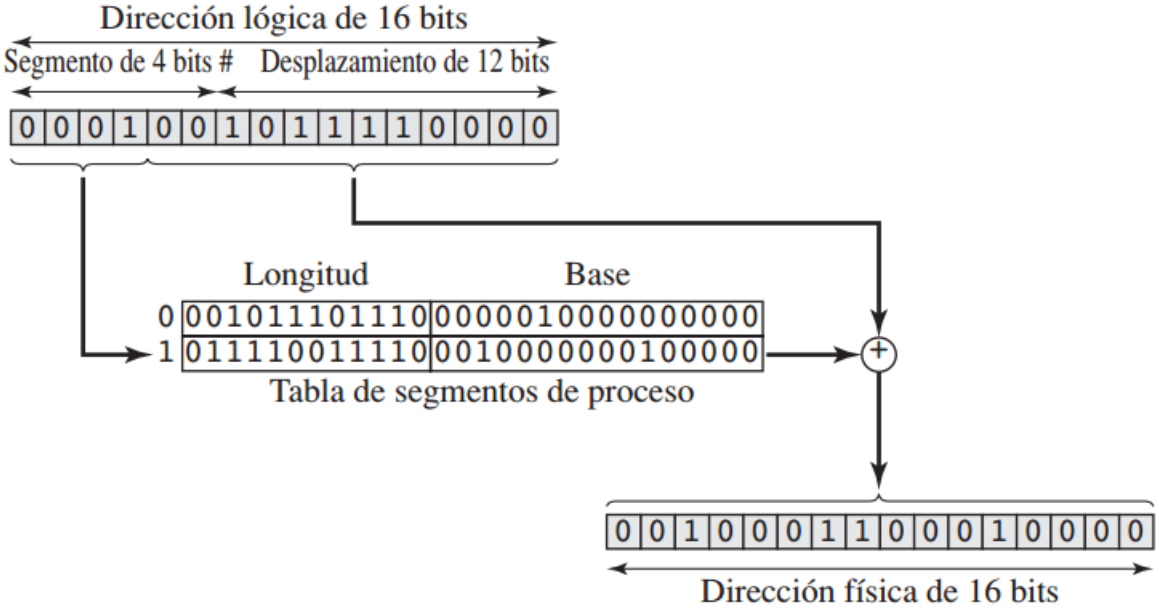
Nro Segmento	Longitud	Memoria Base
0000	1001001101	00010100101100
0001	0001111110	00011101111001
0010	1110000011	00011111110111
0011	1111101000	00101101111010
0100	0100111010	00111101100010

Calcular las direcciones físicas a partir de las siguientes lógicas:

a. (3, 875)
b. (4, 200)
c. (1, 348)

Segmentación Simple

El proceso para calcular la memoria física a partir de la lógica, es la siguiente:



Segmentación Simple

Dirección lógica (3, 875) dónde el nro de segmento en binario corresponde al 0011 (primeros 4 bits), revisamos la tabla dada anteriormente y obtenemos la longitud: 1111101000 (últimos 10 bits) y la memoria base: 001011101111010 (14 bits).

Como para obtener la dirección física es necesario sumar el desplazamiento con la memoria base, entonces realizamos la suma:

875 = 1101101011 (10 bits)

001011101111010

\+ 1101101011

00111011100101 (14 bits que corresponden a la memoria física a partir de la lógica).