## **GESTIÓN DE MEMORIA**

La gestión de memoria es una función fundamental del sistema operativo, ya que se encarga de asignar eficientemente el espacio en memoria a los distintos procesos en ejecución. Para que esta gestión sea eficaz, debe cumplir con ciertos requisitos:

### ✅ REQUISITOS DE LA GESTIÓN DE MEMORIA

* **Reubicación**
* **Protección**
* **Compartición**
* **Organización lógica**
* **Organización física**

### REUBICACIÓN

La memoria principal se comparte entre múltiples procesos. Para optimizar el uso del procesador, es conveniente poder mover los procesos dentro de la memoria.  
Cuando un proceso es almacenado temporalmente en la memoria secundaria (como el disco), no siempre puede regresar a la misma posición en la memoria principal. Por eso, el sistema operativo debe ser capaz de **reubicar dinámicamente** un proceso en distintas áreas de la memoria física.

### PROTECCIÓN

Cada proceso debe estar protegido contra accesos no autorizados (intencionales o accidentales) por parte de otros procesos.  
Esto implica:

* Restringir la lectura o escritura de memoria de un proceso desde otro.
* Impedir que un proceso ejecute instrucciones pertenecientes a otro.
* Proteger el sistema operativo de accesos por parte de procesos de usuario.

El procesador debe ser capaz de **detectar y abortar** instrucciones que violen estas reglas.

### COMPARTICIÓN

El sistema debe permitir que varios procesos accedan a zonas comunes de memoria, siempre de forma controlada.  
Esto es útil cuando:

* Los procesos cooperan entre sí.
* Comparten estructuras de datos.

El acceso compartido no debe comprometer la seguridad ni la integridad de los datos.

### ORGANIZACIÓN LÓGICA

Aunque físicamente la memoria es una secuencia lineal de bytes o palabras, los programas no se construyen de forma lineal, sino en **módulos**.

El sistema operativo, junto con el hardware, puede manejar los programas como conjuntos de módulos, lo cual trae varias ventajas:

* Los módulos se pueden desarrollar y compilar por separado.
* Se pueden aplicar distintos niveles de **protección** (solo lectura, solo ejecución).
* Es posible **compartir módulos** entre procesos, lo que se ajusta a la manera en que el usuario organiza sus programas.

La técnica que permite esta organización lógica de manera más eficaz es la **segmentación**.

### ORGANIZACIÓN FÍSICA

La memoria de un computador se divide en al menos dos niveles:

* **Memoria principal (RAM):** rápida pero costosa y volátil.
* **Memoria secundaria (disco):** más lenta, más barata y no volátil.

La **memoria secundaria** sirve para almacenar programas y datos a largo plazo, mientras que la **memoria principal** contiene los datos y programas actualmente en uso.

Uno de los mayores desafíos del sistema operativo es gestionar el **flujo de información entre estos dos niveles**. Aunque se podría delegar esta responsabilidad en el programador, esto no es práctico porque:

* La memoria disponible puede no ser suficiente, y el uso de técnicas como la **superposición (overlaying)** implica una gran carga para el programador.
* En un entorno multiprogramado, el programador no puede saber cuánta memoria tendrá disponible ni en qué ubicación.

Por eso, el **sistema operativo debe encargarse automáticamente de mover información entre memoria principal y secundaria**, optimizando el uso de los recursos disponibles.

### PARTICIONAMIENTO DE LA MEMORIA

La gestión de memoria consiste en cargar procesos en la memoria principal para que puedan ejecutarse.  
En sistemas modernos **multiprogramados**, se utiliza un mecanismo avanzado llamado **memoria virtual**, que permite ejecutar programas como si tuvieran acceso a una memoria mucho mayor que la disponible físicamente.  
Esto se logra mediante dos técnicas clave: **segmentación** y **paginación**.

Antes de profundizar en ellas, es útil revisar esquemas más simples que **no usan memoria virtual**, como:

* Particionamiento (hoy en desuso)
* Paginación simple
* Segmentación simple

(Tanto las particiones de tamaño fijo como variable son **ineficientes** en el uso de la memoria)

### PARTICIONAMIENTO FIJO

En este esquema, el sistema operativo **reserva una porción fija** de la memoria para sí mismo, y el resto se divide en **particiones de tamaño fijo** para albergar procesos.

#### Tamaños de partición

Una opción es usar particiones **todas del mismo tamaño**. Un proceso puede ocupar cualquier partición si su tamaño es menor o igual al de la partición disponible.

Si todas están ocupadas, el S.O. puede hacer **swap**: intercambiar un proceso en ejecución por otro en espera.

#### Dificultades

1. **Programas grandes**:  
   Si un proceso es más grande que una partición, se requiere usar **overlays**: dividir el programa en módulos que se cargan y descargan según necesidad. Esto complica la programación.
2. **Baja eficiencia**:  
   Un proceso pequeño igual ocupa una partición completa, lo que genera **fragmentación interna**: pérdida de espacio dentro de la partición no aprovechado por el proceso.

#### Alternativa: Particiones de distintos tamaños

Aunque no elimina los problemas, **mejora la eficiencia**:

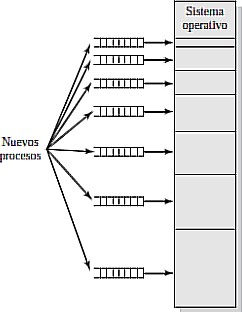
* Programas grandes se colocan en particiones grandes, evitando overlays.
* Programas chicos aprovechan particiones pequeñas, reduciendo la fragmentación interna.

### ****ALGORITMOS DE UBICACIÓN****

#### **Particiones Fijas del Mismo Tamaño**

* La ubicación de procesos es simple:  
  En cuanto hay una partición libre, se carga un proceso en ella.
* Si todas las particiones están ocupadas y los procesos no están listos para ejecutarse, uno de ellos debe ser **llevado al disco** (swap).
* ¿Qué proceso se lleva a disco? ➝ **Lo decide el planificador**.
* **Desventajas**:
  + El número de procesos activos está limitado por la cantidad de particiones definidas al generar el sistema.
  + La eficiencia depende de que los tamaños de los procesos coincidan con el tamaño de las particiones, lo cual rara vez ocurre.

#### **Particiones Fijas de Diferente Tamaño**

* Se pueden asignar procesos a particiones según su tamaño.
* La técnica más común es **asignar cada proceso a la partición más pequeña en la que quepa**.
* Esto **minimiza la fragmentación interna** (espacio desaprovechado dentro de las particiones).
* Se necesita una **cola de planificación por partición**, para organizar los procesos que esperan por una en particular.
* **Desventaja sistémica**:  
  Aunque se minimiza la fragmentación a nivel de cada partición, **no se optimiza el uso global de la memoria**.
  + Ejemplo: Si no hay procesos de entre 12 y 16 MB, una partición de 16 MB queda sin usarse.
* **Ventaja general**:  
  Proporciona más flexibilidad que usar particiones todas del mismo tamaño.

### ****PARTICIONAMIENTO DINÁMICO****

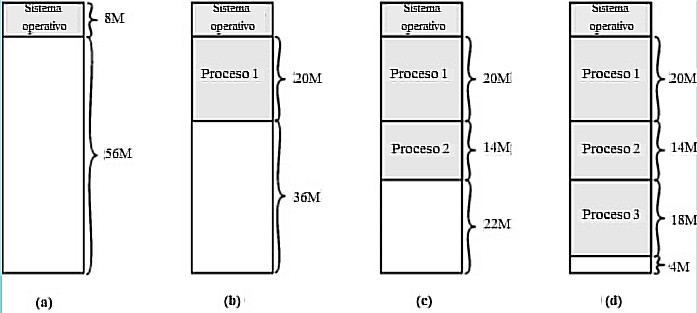
#### 🔹 Características Generales

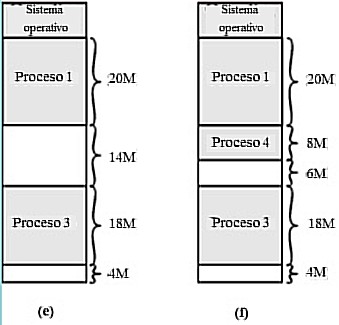
* Las particiones **no son fijas**: tienen **longitud y cantidad variable**.
* A cada proceso se le asigna **exactamente la memoria que necesita**.
* Esto evita la **fragmentación interna,** pero da lugar a **fragmentación externa**.

Un ejemplo:

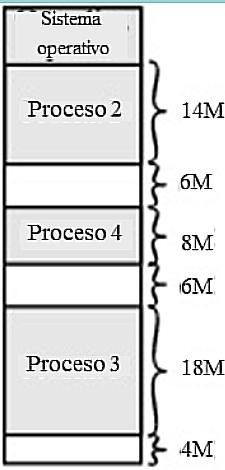
1-La memoria principal solo contiene al **Sistema Operativo**.

**2-Carga de procesos 1, 2 y 3**:  
Se colocan de forma contigua ocupando justo el espacio que requieren.  
Queda un **hueco final muy pequeño**, no apto para nuevos procesos grandes.



3-No hay procesos listos para ejecutar → el S.O. **envía el proceso 2 al disco**.  
Esto **libera espacio** que permite cargar un **nuevo proceso 4**. Como es **más pequeño que el proceso 2**, **queda un hueco nuevo**.

## **4-Proceso 2 disponible otra vez**: El S.O. necesita traer de vuelta al **proceso 2**, pero **no hay suficiente espacio**. ➝ Entonces **lleva el proceso 1 al disco** para liberar espacio.

**Proceso 2 entra**:  
Pero ahora hay **varios huecos pequeños** en memoria, esto se llama:

#### Fragmentación Externa

* Con el tiempo, se crean muchos **huecos pequeños entre procesos**, lo que reduce la eficiencia.
* Esto se llama **fragmentación externa**.

#### **Solución: Compactación**

* El sistema operativo **reubica los procesos en memoria** para que estén **contiguos**.
* Así, **toda la memoria libre queda unida en un solo bloque**.
* ⚠️ **Desventaja**: es una operación **costosa en tiempo**.

## **PARTICIONAMIENTO DINAMICO – ALGORITMOS DE UBICACIÓN**

Debido a que **la compactación de memoria consume una gran cantidad de tiempo**, el diseñador del S.O. debe ser inteligente a la hora de decidir cómo

Asignar la memoria a los procesos.

Existen cuatro algoritmos de colocación que puede considerarse:

* **Mejor-ajuste (best-fit)**

Asigna el bloque **más pequeño posible** donde quepa el proceso. Minimiza espacio desperdiciado pero puede dejar huecos inútiles.

* **Primer-ajuste (first-fit)**

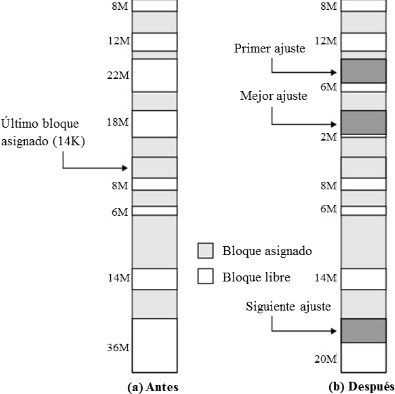
Busca desde el **inicio de la memoria** y asigna el **primer bloque suficientemente grande**. Es rápido.

* **Siguiente-ajuste (next-fit)**

Igual que First-Fit, pero comienza desde **la última posición usada**. Puede distribuir mejor la carga.

* **Peor-ajuste (worst-fit)**

Asigna el **bloque más grande disponible**. Intenta dejar bloques útiles para futuros procesos.

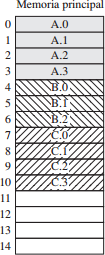
Un ejemplo de configuración de memoria después de un número de colocaciones e intercambios en disco.

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.

## **PAGINACION SIMPLE**

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.

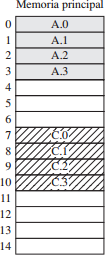
Dichas porciones del proceso, conocidas como **páginas**, se les asigna porciones disponibles de memoria, conocidas como **marcos** o marcos de página.

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 está 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.

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**.

El S.O. mantiene una **tabla de páginas** por cada proceso.

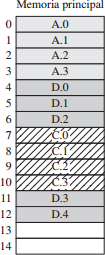
La tabla de páginas muestra la **ubicación del marco por cada página** del proceso.

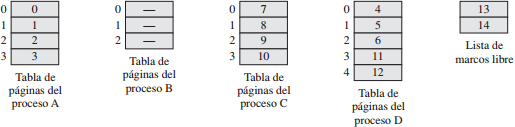
Dentro del proceso, cada dirección lógica está formada por un **número de página** y un **desplazamiento** dentro de la página.

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).

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 más de una partición y no necesariamente contiguas.

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.

**CALCULO DE DIRECCIÓNES**

**Ejercicio de paginación simple, donde hay que convertir direcciones lógicas (DL) en direcciones físicas (DF) usando una tabla de páginas.**

Una dirección lógica (DL) es la dirección que un proceso “cree” que está usando. Pero en realidad, la memoria RAM real se maneja con direcciones físicas (DF).

Para pasar de lógica a física, se usa paginación: la memoria se divide en páginas del mismo tamaño.

--------------------------------------------------------------------------------------------------------

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

**Datos del sistema**:

**-Sistema de 12 bits con paginación simple.**

**-4 bits para el número de página → 2⁴ = 16 páginas posibles.**

**-8 bits para el desplazamiento u offset dentro de cada página → 2⁸ = 256 posiciones por página.**

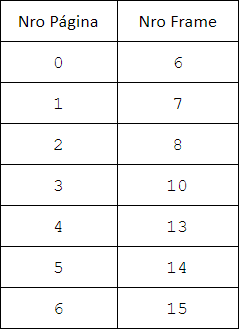
**-Por lo tanto, cada página mide 256 bytes.**

**-El total de direcciones lógicas posibles es 2¹² = 4096.**

**(Todos los cálculos se hacen en base 2 porque las computadoras trabajan internamente en base 2, o sea, con números binarios.)**

--------------------------------------------------------------------------------------------------------

Teniendo un proceso con 7 páginas y la siguiente tabla de páginas calcular las direcciones físicas a partir de las relativas:



DL (dirección lógica) = 321

DL (dirección lógica) = 573

**SOLUCION**:

#### Para DL = 321:

1. **Dividimos por el tamaño de página (256):**  
   321 / 256 = 1 c  
   → Página lógica: **1**

Offset = 321 - (1 × 256) = 65  
→ Offset: **65**

1. **Consultamos la tabla de páginas:**  
   La página lógica 1 está mapeada al **marco 7**.
2. **Calculamos la dirección física:**  
   DF = nroMarco \* tamañoMarco + offset  
   DF = 7 \* 256 + 65 = 1857

Dirección física: **1857**

#### Para DL = 573:

1. **Dividimos por el tamaño de página (256):**  
   573 / 256 = 2 con resto 61  
   → Página lógica: **2**

**Offset: 573 – (2x256)=61**→ Offset: **61**

1. **Consultamos la tabla de páginas:**  
   La página lógica 2 está mapeada al **marco 8**.
2. **Calculamos la dirección física:**   
   DF = nroMarco \* tamañoMarco + offset

DF = 8 \* 256 + 61 = 2109

Dirección física: **2109**

## **SEGMENTACIÓN SIMPLE**

La **segmentación** es una técnica de gestión de memoria donde un programa se divide en **segmentos lógicos**

Cada uno de estos **segmentos** pue**de tener un tamaño distinto, y no es necesario que estén todos juntos en memoria** (pueden estar en distintos lugares, no contiguos).

### Ventajas

* Elimina la **fragmentación interna** (solo se usa lo necesario en cada segmento).
* Es **más flexible** que el particionamiento fijo o dinámico.
* Refleja mejor la estructura lógica de los programas (cada parte se trata por separado).

### Desventajas

* Puede haber **fragmentación externa**, aunque menor que en otros métodos.
* Requiere más información (como la longitud del segmento).
* Más compleja que la paginación en su implementación

### ¿Cómo se arma una dirección?

Una **dirección lógica** en segmentación se compone de dos partes:

* **Número de segmento**
* **Desplazamiento** dentro del segmento

Por ejemplo, si la dirección lógica es (2, 45) significa:  
👉 “andá al **segmento 2** y desplazate **45 bytes** desde el inicio del segmento”.

### ¿Qué es la tabla de segmentos?

Es una estructura que guarda la **información de cada segmento** del proceso.  
Cada entrada de la tabla tiene:

* **Numero de segmento**
* **Memoria base** → la dirección física donde comienza ese segmento en memoria.
* **Longitud** → cuántos bytes ocupa ese segmento.

### ¿Cómo se traduce a dirección física?

Para obtener la dirección física (DF), se hace:

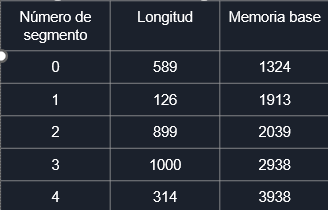
DF = base del segmento + desplazamiento

🛑 **¡Pero cuidado!** El desplazamiento **no puede superar** el tamaño del segmento. Si pasa, se considera **una dirección inválida** (error de segmentación).

**CALCULO DE DIRECCIÓNES**

Sea un sistema de 14 bits con Segmentación. Donde 4 bits son para el número de segmento y 10 bits para el tamaño del segmento y desplazamiento dentro de él.

Y la siguiente tabla de segmentos:



Calcular las direcciones físicas a partir de las siguientes lógicas:

A.(3, 875)

B.(4, 200)

C.(1, 348)

--------------------------------------------------------------------------------------------------------

Sistema de **14 bits** para direcciones lógicas.

Se dividen así:

* + **4 bits** → número de segmento → permite hasta 2⁴ = 16 segmentos.
  + **10 bits** → desplazamiento dentro del segmento → permite hasta 2¹⁰ = 1024 bytes por segmento.

**SOLUCION**:

### Dirección lógica (3, 875)

* Segmento = 3
* Desplazamiento = 875
* Según la tabla:
  + Longitud = 1000 → ✅ 875 < 1000 → es válida
  + Base = 2938

Dirección física = 2938 + 875 = 3813

**Resultado: 3813**

### Dirección lógica (4, 200)

* Segmento = 4
* Desplazamiento = 200
* Según la tabla:
  + Longitud = 314 → ✅ 200 < 314 → es válida
  + Base = 3938

**Dirección física = 3938 + 200 = 4138**

**Resultado: 4138**

### Dirección lógica (1, 348)

* Segmento = 1
* Desplazamiento = 348
* Según la tabla:
  + Longitud = 126 → ❌ 348 > 126 → **¡Error!**
  + El desplazamiento **supera el tamaño permitido del segmento**

⛔ **Resultado: Dirección no válida (fuera del segmento)**