Sistemas Operativos

Administración de Memoria I

Curso 2025

Facultad de Ingeniería, Universidad de la República, Uruguay

Agenda

1. Introducción
2. Áreas de memoria de un proceso
3. Carga dinámica, enlace dinámico y bibliotecas compartidas
4. Asociación de direcciones
5. Tipos de direccionamiento
6. Asignación de memoria
7. Swapping

Introducción

Introducción

La administración de la memoria es una de las tareas más importantes del sistema operativo.

En los sistemas operativos multiprogramados es necesario mantener varios programas en memoria al mismo tiempo.

Existen varios esquemas para administrar la memoria, que requieren distinto soporte del hardware.

El sistema operativo es responsable de las siguientes tareas:

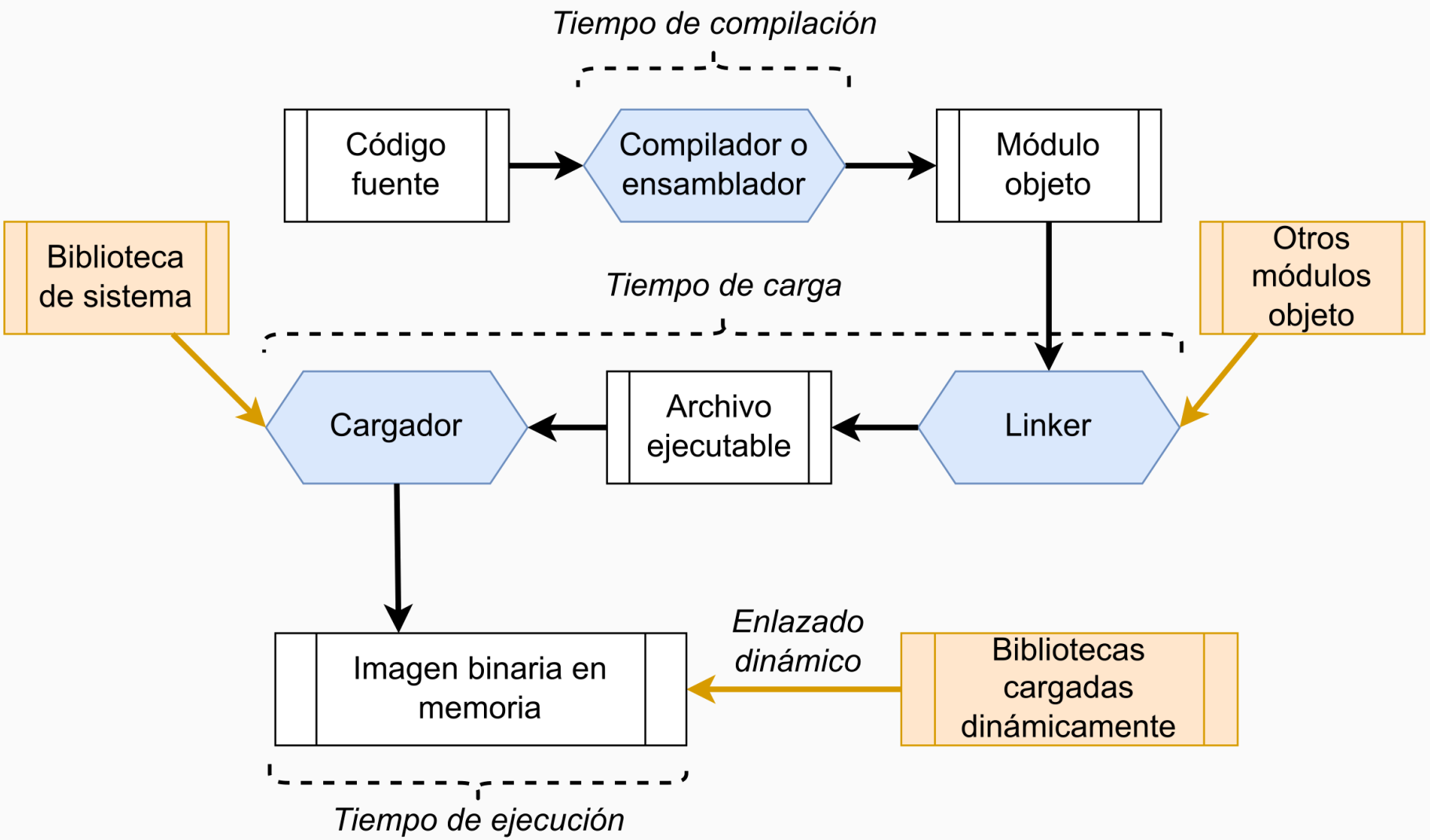
- Mantener qué partes de la memoria están siendo utilizadas y por qué procesos.
- Decidir cuáles procesos serán cargados a memoria cuando exista espacio de memoria disponible.
- Asignar y quitar espacio de memoria según sea necesario

Conceptos básicos: preparación de un programa para ejecutar

Los programas son escritos en lenguajes de alto nivel y deben pasar por distintas etapas antes de ser ejecutados:

- Compilación (*compile*): Traducción del código fuente del programa a un código objeto, por parte del compilador.
- Enlace (*link*): Enlace de varios códigos objeto en un archivo ejecutable, por parte del linker.
- Carga (*load*): Asignación del archivo ejecutable a la memoria principal del sistema por parte del loader.

Conceptos básicos: preparación de un programa para ejecutar



Conceptos Básicos: programas y archivos

Compilador: genera un archivo objeto para cada archivo fuente.
El archivo objeto no es completo, ya que utiliza información de otros archivos (e.g., llamados a funciones externas).

Enlazador (*linker*): combina todos los archivos objeto de un programa en un único archivo objeto.

Sistema operativo: carga los programas en memoria, permite compartir la memoria entre varios procesos y brinda mecanismos a los procesos para obtener más memoria en forma dinámica.

Bibliotecas dinámicas: proveen rutinas cargadas en tiempo de ejecución (e.g., biblioteca del entorno de ejecución de C).

Conceptos básicos: preparación de un programa para ejecutar

El linker surgió ante la necesidad de modularizar y reutilizar código. Resuelve las referencias externas y las posiciones relativas de los símbolos en los diferentes módulos, formando un módulo consolidado.

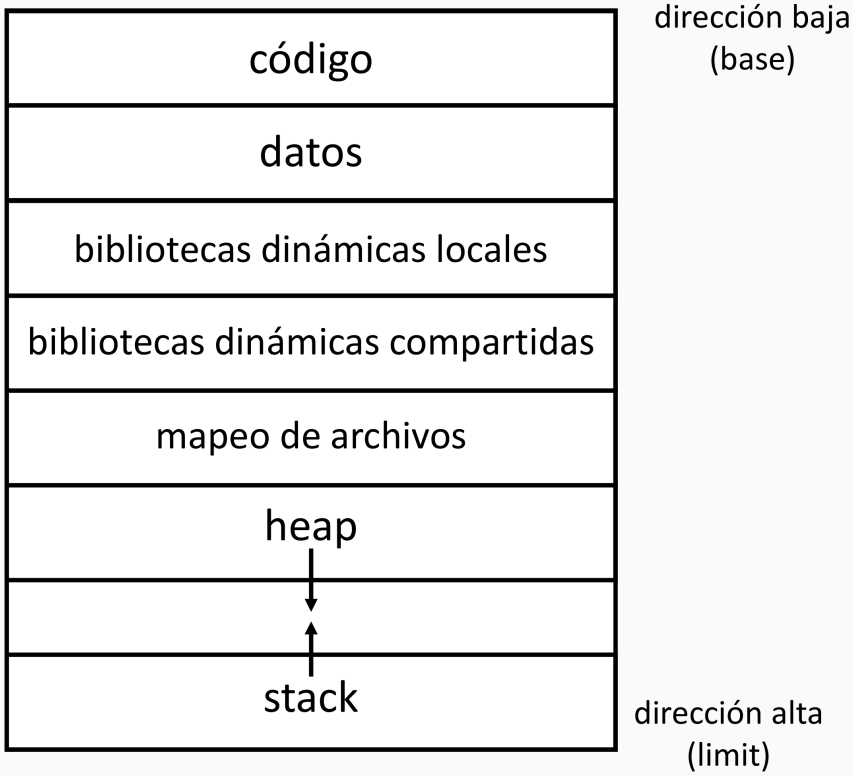
Cuando se crea un proceso, el loader del sistema crea en memoria el espacio necesario para las áreas (de código, datos, etc.) y las carga con la información.

El compilador, el linker, el sistema operativo y las bibliotecas dinámicas deben cooperar para administrar la información y realizar la asignación.

Áreas de memoria de un proceso

Áreas de memoria

La memoria de un proceso (en ejecución) se estructura en diferentes áreas:



Carga dinámica, enlace dinámico y bibliotecas compartidas

Carga dinámica (dynamic loading)

El tamaño de un proceso en memoria está limitado por la cantidad de memoria física del sistema.

Para lograr un mayor nivel de aprovechamiento de la memoria se puede utilizar la carga dinámica.

La carga dinámica implica que una rutina no es cargada en memoria física hasta que sea invocada.

La ventaja de la carga dinámica es que las rutinas que no son utilizadas no son cargadas en memoria física, por lo cual no consumen recursos innecesariamente.

Enlace dinámico (dynamic linking)

En la etapa de enlace de un programa, pueden incorporarse las bibliotecas compartidas al archivo ejecutable generado (enlace estático, static linking).

- En Linux: /usr/lib/libc.a

Otra alternativa es que las bibliotecas compartidas sean cargadas en tiempo de ejecución (enlace dinámico, dynamic linking).

- En Linux: /lib/libc.so
- En Windows: system.dll

Enlace dinámico (dynamic linking)

En los archivos ejecutables se incorporan las bibliotecas estáticas y para las dinámicas se mantiene una referencia.

- En Linux, comando ls:

```
$ ldd /bin/ls
librt.so.1 => /lib/librt.so.1 (0x4001c000)
libc.so.6 => /lib/libc.so.6 (0x40030000)
libpthread.so.0 => /lib/libpthread.so.0 (0x40149000)
/lib/ld-linux.so.2 (0x40000000)
```

El enlace dinámico, utilizado en conjunto con la carga dinámica, permite un uso más eficiente de la memoria, dado que las bibliotecas dinámicas se cargan una única vez en memoria principal.

Enlace dinámico (dynamic linking)

A diferencia de la carga dinámica, el enlace dinámico de bibliotecas compartidas generalmente requiere la intervención del sistema operativo.

Dado que la memoria de un proceso está protegida, solamente el sistema operativo puede verificar si la rutina necesaria está alojada en el espacio de memoria de otro proceso o si puede permitir que múltiples procesos accedan a las mismas direcciones de memoria.

Asociación de direcciones

Asociación de direcciones (address binding)

Un programa de usuario pasa por varias etapas antes de ejecutarse.

Las direcciones de memoria se pueden representar de diferentes maneras durante las distintas etapas.

Las direcciones en el programa fuente son generalmente simbólicas (variables). El compilador vincula estas direcciones simbólicas a direcciones reubicables (e.g.,“14 bytes desde el comienzo de este módulo”).

El cargador vincula las direcciones reubicables a direcciones absolutas (e.g., `0x00074014`). Cada enlace es una asignación de un espacio de direcciones a otro.

Asociación de direcciones (address binding)

La asociación de instrucciones y datos a direcciones de memoria se puede realizar en las diferentes etapas.

Tiempo de compilación (*compile time*)

El programa se asigna a un lugar específico y conocido de la memoria física. Se genera *código absoluto*: las direcciones de memoria son referenciadas en forma absoluta.

Si la ubicación de inicio cambia, debe recompilarse el código. Los programas .COM de MS-DOS utilizan este mecanismo.

Asociación de direcciones (address binding)

La asociación de instrucciones y datos a direcciones de memoria se puede realizar en las diferentes etapas.

Tiempo de carga (*load time*)

En tiempo de compilación no se sabe dónde residirá el proceso en la memoria. El compilador debe generar código reubicable.

Las direcciones de memoria se referencian en forma relativa.
La asociación se retrasa hasta el momento de la carga.
Si la dirección inicial cambia, solo se necesita recargar el código de usuario para tomar en cuenta el valor modificado.

Asociación de direcciones (address binding)

Tiempo de ejecución (*execution time*)

Un programa puede variar su ubicación en memoria física durante su ejecución.

Si el proceso puede moverse durante su ejecución (desde un segmento de memoria a otro), la asignación debe retrasarse hasta el tiempo de ejecución.

Se requiere hardware especial disponible para que este esquema funcione.

La mayoría de los sistemas operativos de uso general utilizan este método.

Tipos de direccionamiento

Tipos de direccionamiento

Se definen dos tipos de direccionamiento:

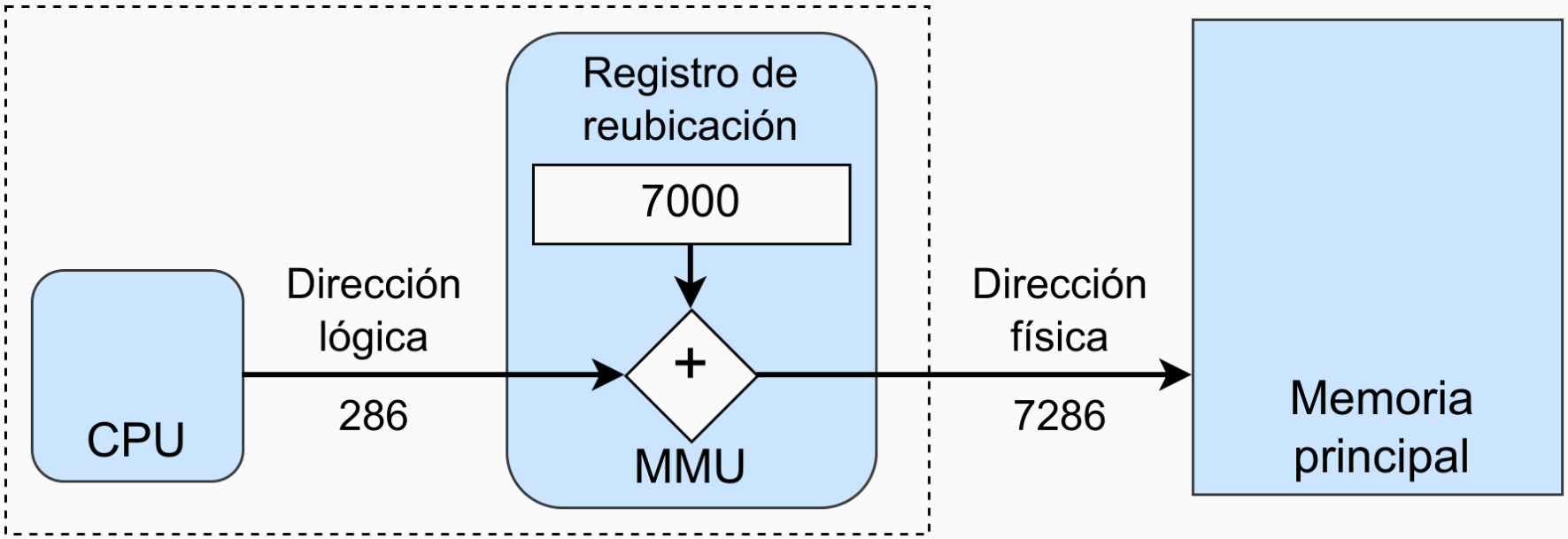
- Direccionamiento físico (*physical address*): La dirección física manipulada por la unidad de memoria.
- Direccionamiento virtual o direccionamiento lógico (*virtual address*): La dirección lógica generada por asociación de direcciones.

Para la asociación de direcciones en tiempo de compilación y en tiempo de carga, las direcciones lógicas y físicas coinciden. No es así para la asociación en tiempo de ejecución.

Tipos de direccionamiento

La traducciones de direcciones lógicas a físicas son hechas por la Unidad de Manejo de Memoria (Memory Management Unit, MMU).

Los procesos solo manipulan direcciones lógicas y no visualizan las físicas, que solamente son vistas por la MMU.



Reubicación

Los sistemas operativos pueden administrar dónde se almacena cada proceso en la memoria utilizando la técnica de reubicación.

Cuando el sistema operativo carga un proceso le asigna un segmento contiguo de memoria. La dirección física inicial (más baja) del proceso es la dirección base y la dirección física más grande a la que puede acceder el proceso es la dirección límite.

Existen dos métodos de reubicación: estática y dinámica.

Reubicación estática

El sistema operativo ajusta la dirección de memoria de un proceso para reflejar su posición inicial en la memoria (en tiempo de compilación o tiempo de carga).

El proceso ejecuta dentro del espacio que se le ha asignado. Una vez que se completa el proceso de reubicación estática, el sistema operativo ya no puede reubicar el proceso hasta que finaliza.

Reubicación dinámica

Reubicación dinámica

El hardware agrega un registro de reubicación (valor base) a la dirección virtual generada por el compilador.

El registro de reubicación permite la traducción a una dirección
de memoria física. El hardware compara la dirección de memoria con el registro límite (valor más alto disponible en la sección asignada). Si la dirección de memoria es mayor que el límite ocurre un error de direccionamiento.

Reubicación dinámica: ventajas y desventajas

Ventajas:

- El sistema operativo puede mover un proceso si es necesario.
- Los procesos pueden crecer con el tiempo (se pueden reubicar en un bloque de memoria más grande).
- Se realiza por hardware y es simple: requiere dos registros especiales, una adición simple y una comparación simple.

Desventajas:

- Existe un overhead en el procesamiento.
- Los procesos no pueden compartir memoria entre si.
- Los procesos requieren una cantidad fija de memoria física, lo que puede limitar la multiprogramación porque cada proceso activo debe caber en la memoria.

Reubicación: propiedades

La reubicación tiene tres propiedades principales:

- Transparencia: los procesos no son conscientes de que están compartiendo el recurso memoria.
- Seguridad: las referencias a la memoria se verifican para garantizar que se encuentren entre los registros base y límite. Mantiene a otros procesos a salvo de errores.
- Eficiencia: los chequeos de memoria y rangos son rápidos porque se realizan en hardware. Sin embargo, si un proceso crece, moverlo es una operación que requiere tiempo.

Asignación de memoria

Asignación de memoria a nivel del sistema

La memoria está dividida en dos secciones:

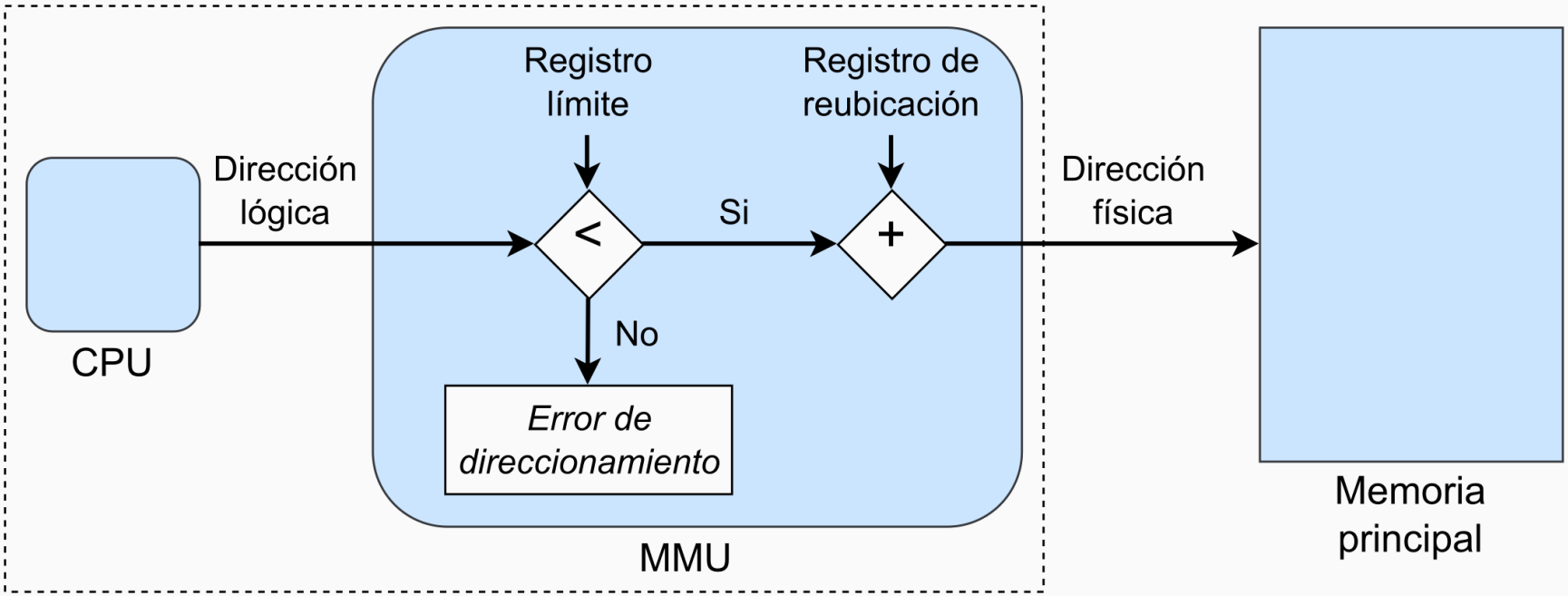
- Sistema operativo residente
- Procesos de usuarios

Es necesario un mecanismo de protección de memoria entre los procesos y con el sistema operativo.

Se utiliza el registro de reubicación y el registro límite para realizar la verificación de accesos válidos a la memoria.

Toda dirección lógica debe ser menor al valor del registro límite.

Protección de memoria



Esquema de asignación de particiones variables

El sistema operativo define particiones de tamaño variable y las asigna a los procesos.

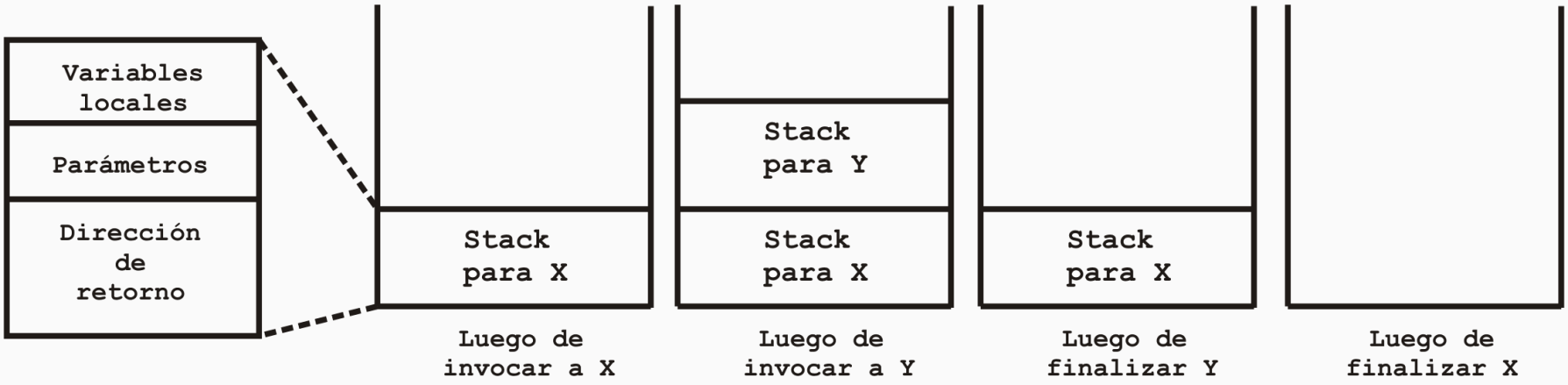
Inicialmente, toda la memoria se considera disponible y asignable. Los procesos indican sus requerimientos de memoria y el sistema operativo les asigna particiones.

El sistema operativo debe conocer las particiones ocupadas y libres. Las estructuras más utilizadas para la implementación son el mapa de bits y la lista encadenada.

Esquema de asignación de particiones variables

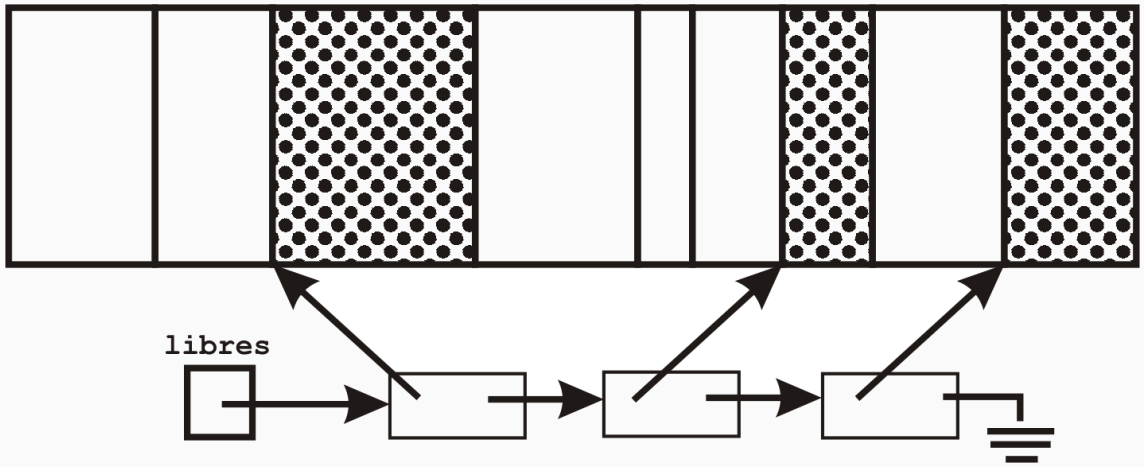
Los procesos requieren memoria dinámicamente, mediante asignaciones en el stack y en el heap.

El uso del stack (estructura LIFO) es más predecible que el del heap.



Esquema de asignación de particiones variables

La asignación en el heap no es predecible como en el stack



Se genera fragmentación de la memoria.

Los sistemas operativos optan por delegar la administración de esta memoria a bibliotecas de usuario.

Los procesos liberan memoria (explícitamente o cuando terminan): se generan huecos en la memoria.

Estrategias de asignación de memoria

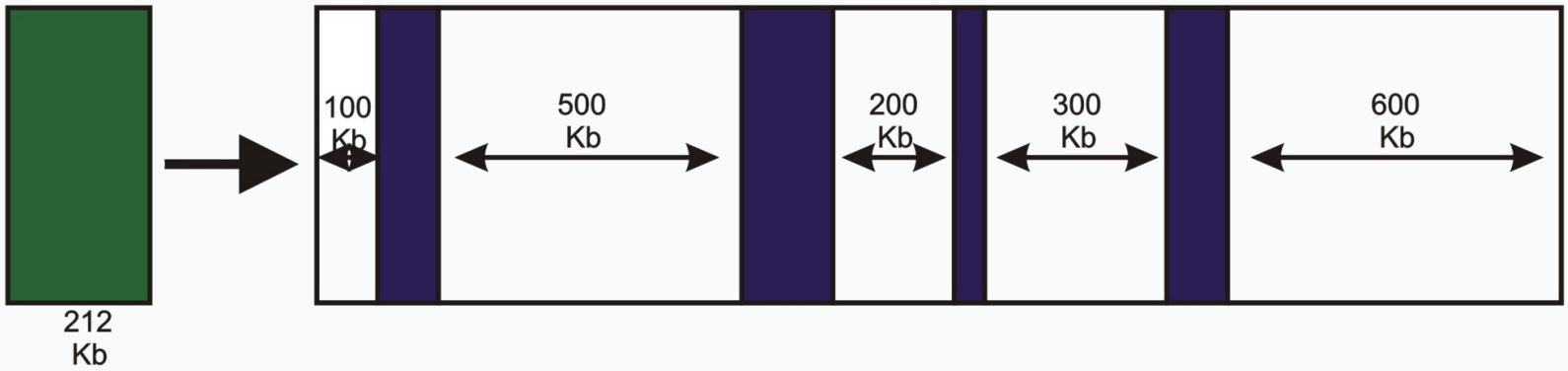
Existen varias estrategias para la asignación de memoria a un proceso:

- First fit: Asigna el primer hueco de memoria libre que satisface la necesidad.
- Best fit: Asigna el mejor hueco de memoria libre que exista en la memoria principal.
- Worst fit: Asigna el requerimiento en el hueco más grande que exista en la memoria principal.

Estudios de simulación han mostrado que first fit y best fit logran mejores rendimientos en tiempo de asignación y utilización de la memoria que worst fit.

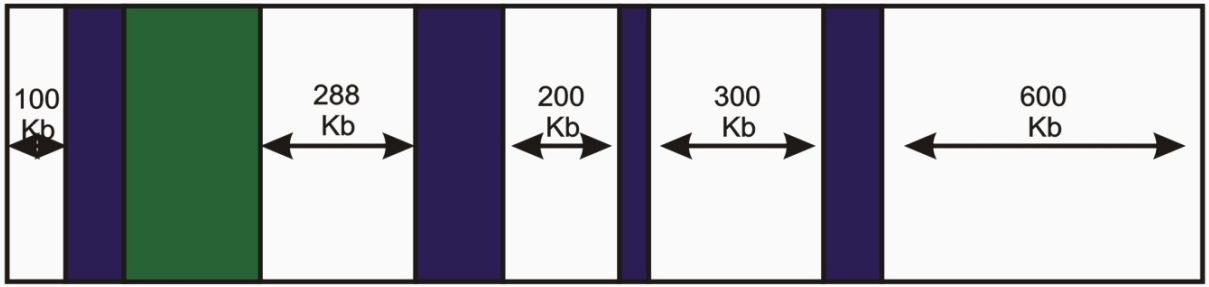
Estrategia de asignación: ejemplo

Asignar un bloque de 212 KB

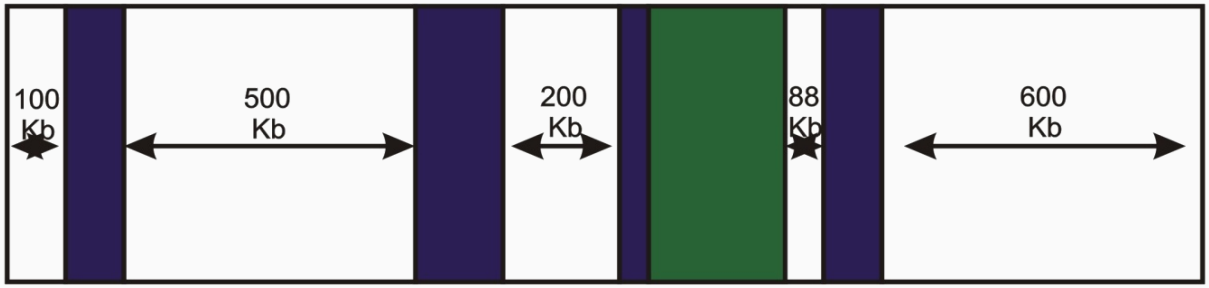


Estrategia de asignación: ejemplo

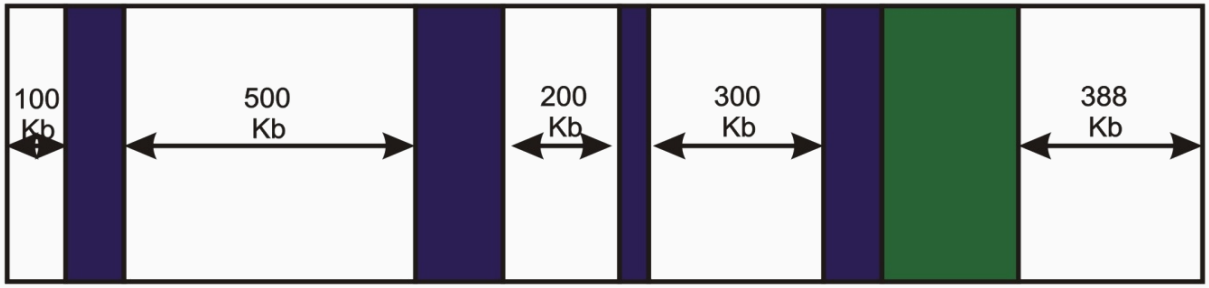
First fit



Best fit



Worst fit



Fragmentación

Las estrategias de asignación sufren el problema de fragmentación externa.

En la memoria se generan muchos huecos pequeños, que no son asignados. La memoria libre está fragmentada en una gran cantidad de huecos pequeños.

La fragmentación externa se da cuando existe suficiente memoria libre en el sistema para satisfacer un requerimiento de memoria, pero no es posible asignarla debido a que no es contigua.

Fragmentación

Regla del 50 %. El análisis estadístico de first fit indica que, incluso optimizando, cada N bloques asignados se pierden $0.5\,N$ bloques debido a la fragmentación (un tercio de la memoria disponible).

Fragmentación interna. Sobrecarga por gestionar huecos muy pequeños. Dado un hueco de 18.464 B y un proceso que solicita 18.462 B. Se genera un hueco de 2 B, la sobrecarga de gestionarlo será sustancialmente mayor que el beneficio.

Para evitar el problema, se divide la memoria física en bloques de tamaño fijo y se asigna la memoria en unidades del tamaño del bloque. La memoria asignada a un proceso puede ser ligeramente mayor que la memoria solicitada. La diferencia es la fragmentación interna: memoria no utilizada, interna a una partición.

Soluciones al problema de fragmentación

Compactación. Barajar el contenido de la memoria para colocar toda la memoria libre en un bloque grande.

Solo es posible si se aplica reubicación dinámica en tiempo de ejecución. Las direcciones se reubican dinámicamente, luego de mover el programa, mover los datos y cambiar el registro de reubicación.

El algoritmo de compactación más simple consiste en mover todos los procesos hacia un extremo de la memoria y todos los huecos en la otra dirección, generando un gran hueco de memoria disponible. Este esquema puede ser costoso.

Soluciones al problema de fragmentación

Espacio de direcciones no contiguo. Permitir que el espacio de direcciones lógicas de los procesos sea no contiguo.

Se permite asignar memoria física a un proceso siempre que exista memoria disponible. Dos técnicas complementarias logran esta solución: la segmentación y la paginación.

Swapping

Swapping

En sistemas multiprogramados, más de un proceso está cargado en memoria principal. Para lograr un mayor nivel de multiprogramación, los procesos que no están ejecutando pueden ser llevados a memoria secundaria temporalmente.

El swapping permite que el espacio total de direcciones físicas de todos los procesos exceda la memoria física real del sistema, aumentando así el grado de multiprogramación en un sistema.

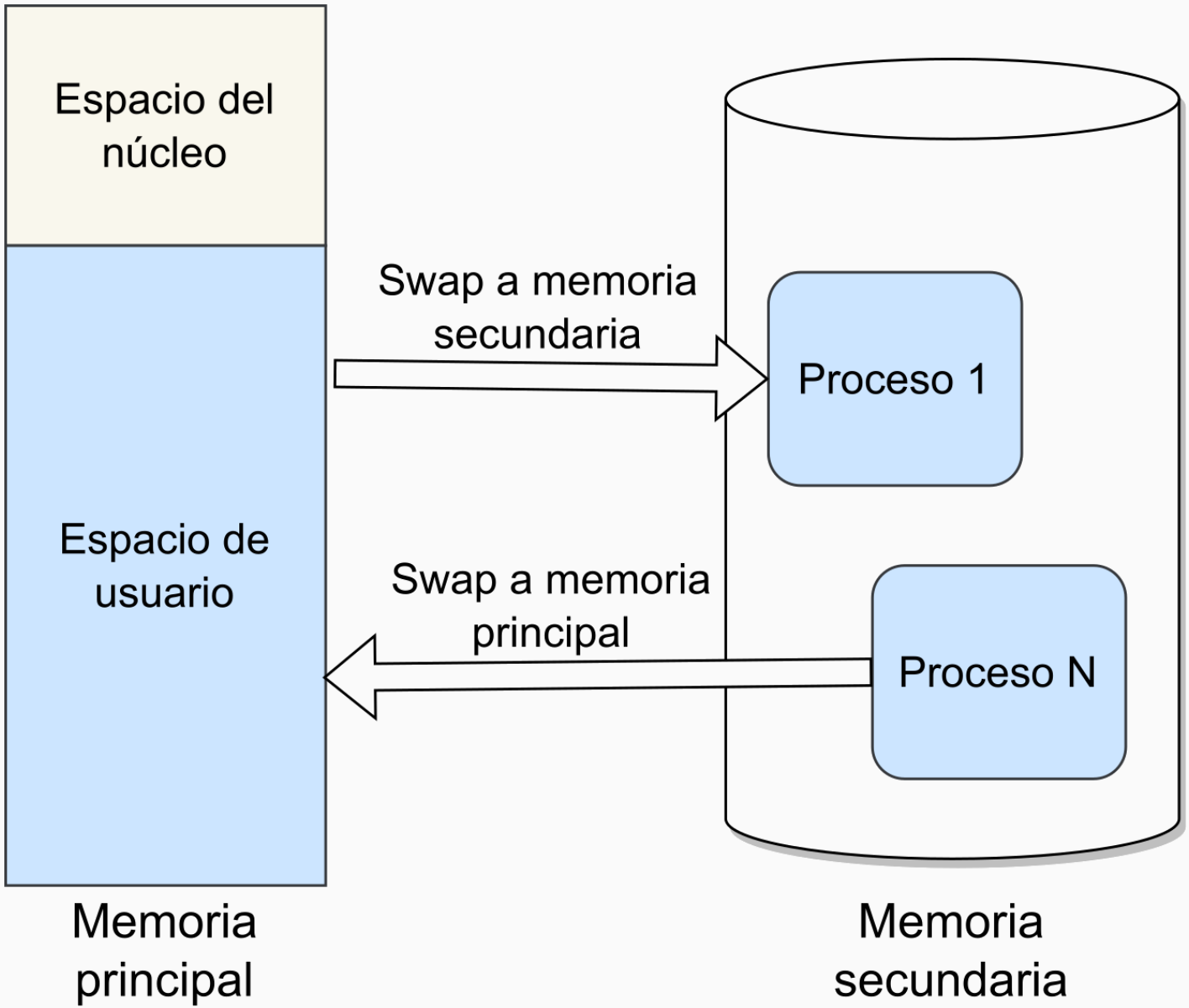
Swapping

La memoria secundaria (*backing store*) es un espacio donde se dispondrán las imagen de memoria de los procesos.

Al mecanismo de llevar un proceso desde memoria principal a memoria secundaria se le denomina swap-out. Al mecanismo inverso se le denomina swap-in.

El mayor tiempo consumido en el swapping es el tiempo de transferencia.

Swapping


Espacio del núcleo
Espacio de usuario
Swap a memoria secundaria
Swap a memoria principal
Proceso 1
Proceso N
Memoria principal
Memoria secundaria

Swapping

El lugar de memoria donde será asignado un proceso en el momento de swap-in depende del método de asociación de direccionamiento (*address binding*) utilizado.

Para asociación en tiempo de compilación o de carga debe ser el mismo lugar que el proceso tenía originalmente.

Para asociación en tiempo de ejecución puede utilizarse cualquier lugar.

Swapping

El sistema mantiene una cola de procesos listos para ejecutarse, cuyas imágenes están en la memoria o en el backing store.

El despachador verifica si el siguiente proceso en la cola está en la memoria. Si no está y no hay una región de memoria libre, intercambia un proceso actualmente en la memoria y el proceso deseado. Luego recarga los registros y transfiere el control al proceso seleccionado.

El tiempo de cambio de contexto es alto. Para un tamaño de proceso de 100 MB y swapping a un disco duro estándar con una velocidad de transferencia de 50 MB/s, la transferencia demanda 2 s para cada proceso.

Swapping

La mayor parte del tiempo de swapping es de transferencia, que es directamente proporcional a la cantidad de memoria intercambiada.

Para reducir los tiempos, el usuario puede informar al sistema los cambios en los requisitos de memoria con llamadas al sistema para solicitar memoria y liberar memoria

Para el swapping, el proceso debe estar inactivo. En especial, no puede estar esperando por E/S pendiente, porque la E/S accede asincrónicamente a la memoria del usuario para los buffers de E/S.

Soluciones: nunca intercambiar un proceso con E/S pendiente o ejecutar operaciones de E/S solo en los buffers del sistema operativo (doble buffer, mayor overhead).

Swapping

El swapping estándar no se utiliza en los sistemas operativos modernos: por los tiempos altos, no es una solución razonable de administración de memoria.

Versiones modificadas de swapping se aplican en UNIX, Linux y Windows.

Variante 1: El swapping está deshabilitado, pero se activa si la memoria libre cae por debajo de una cota. El swapping se detiene cuando aumenta la memoria libre.

Variante 2: Swapping de partes de procesos, en lugar de procesos completos, para reducir el tiempo de intercambio.

Las variantes modificadas de intercambio se usan junto con memoria virtual.