Sistemas Operativos

Grado en Ingeniería Informática
2015-2016

MÓDULO 5: Gestión de memoria

Basado en: Sistemas Operativos
J. Carretero [et al.]

Contenido

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Asignación contigua
  - Intercambio (*swapping*)
- Memoria virtual
  - Políticas de intercambio
- Gestión de memoria en Linux

Objetivos del gestor de memoria

- El SO multiplexa recursos entre procesos
  - Cada proceso cree que tiene una máquina para él solo
  - Gestión de procesos: Reparto de procesador
  - Gestión de memoria: Reparto de memoria
- Objetivos:
  - Ofrecer a cada proceso un espacio lógico propio
  - Dar soporte a las regiones del proceso
  - Proporcionar a los procesos mapas de memoria muy grandes
  - Maximizar el grado de multiprogramación
  - Proporcionar protección entre procesos
  - Permitir que procesos compartan memoria

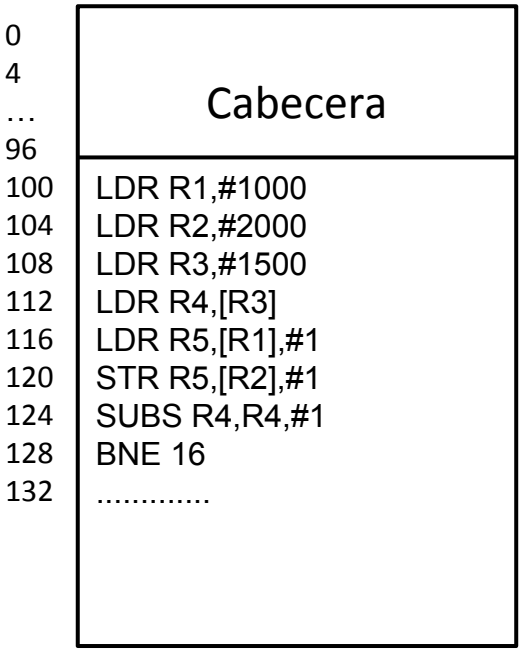
Espacios lógicos independientes

- No se conoce la posición de memoria donde un programa se ejecutará
- Código en ejecutable genera referencias entre 0 y N
- Ejemplo: Archivo ejecutable de un programa que copia un vector

```
for (i=0; i<tam; i++)
        Vdest[i]=Vorg[i];
```

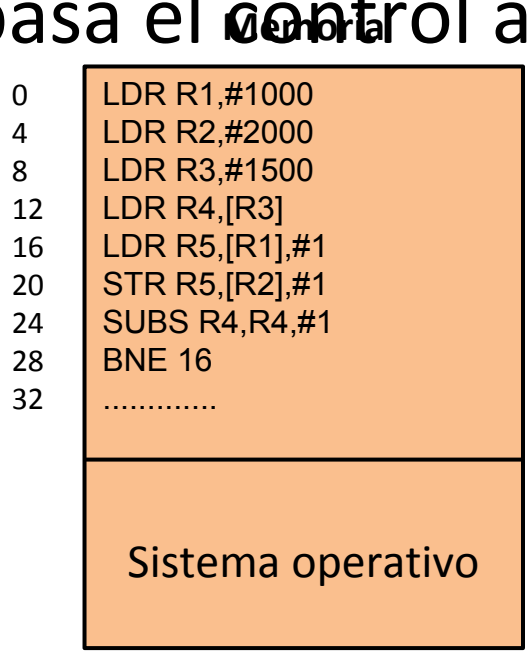
  - –Cabecera de 100Bytes
  - –Instrucción ASM 4Bytes
  - –Vector origen a partir de dirección 1000
  - –Vector destino a partir de dirección 2000
  - –Tamaño del vector en dirección 1500

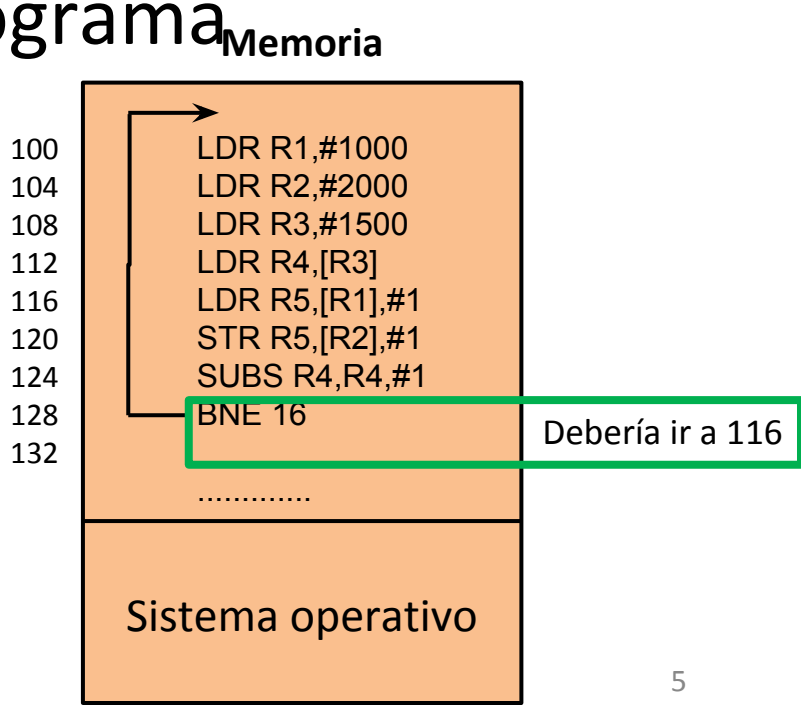
Fichero ejecutable



Ejecución en SO monoprogramado

- Supongamos el SO en direcciones más altas
- Programa se carga en dirección 0
  - Si no se puede, habrá que ajustar las referencias
- Se le pasa el control al programa





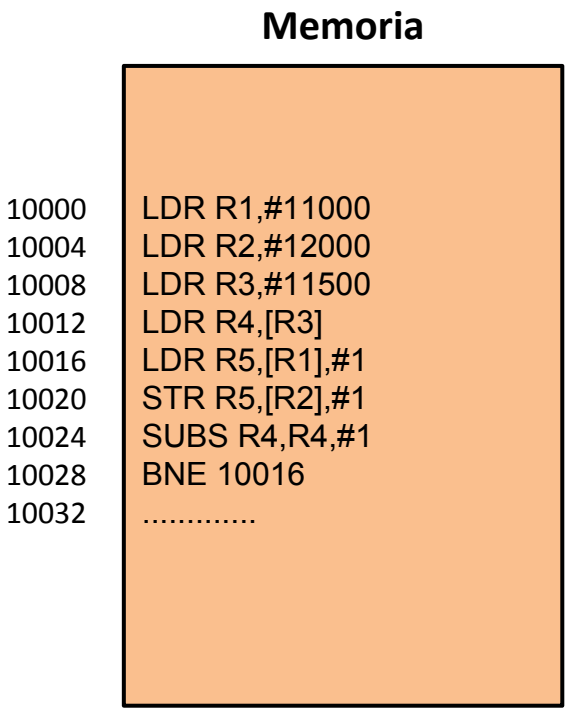
Reubicación

- Traducir direcciones lógicas a físicas
  - Dir. lógicas: direcciones de memoria generadas por el programa
  - Dir. físicas: direcciones de mem. principal asignadas al proceso
- Necesaria en SO con multiprogramación:

  Traducción(IdProc, dir_lógica) $\rightarrow$ *dir_física*
- Reubicación crea espacio lógico independiente para proceso
  - SO debe poder acceder a espacios lógicos de los procesos
- Dos alternativas de reubicación:
  - Software
  - Hardware
- Ejemplo: Programa tiene asignada memoria a partir de 10000
  - Sumar 10000 a direcciones generadas

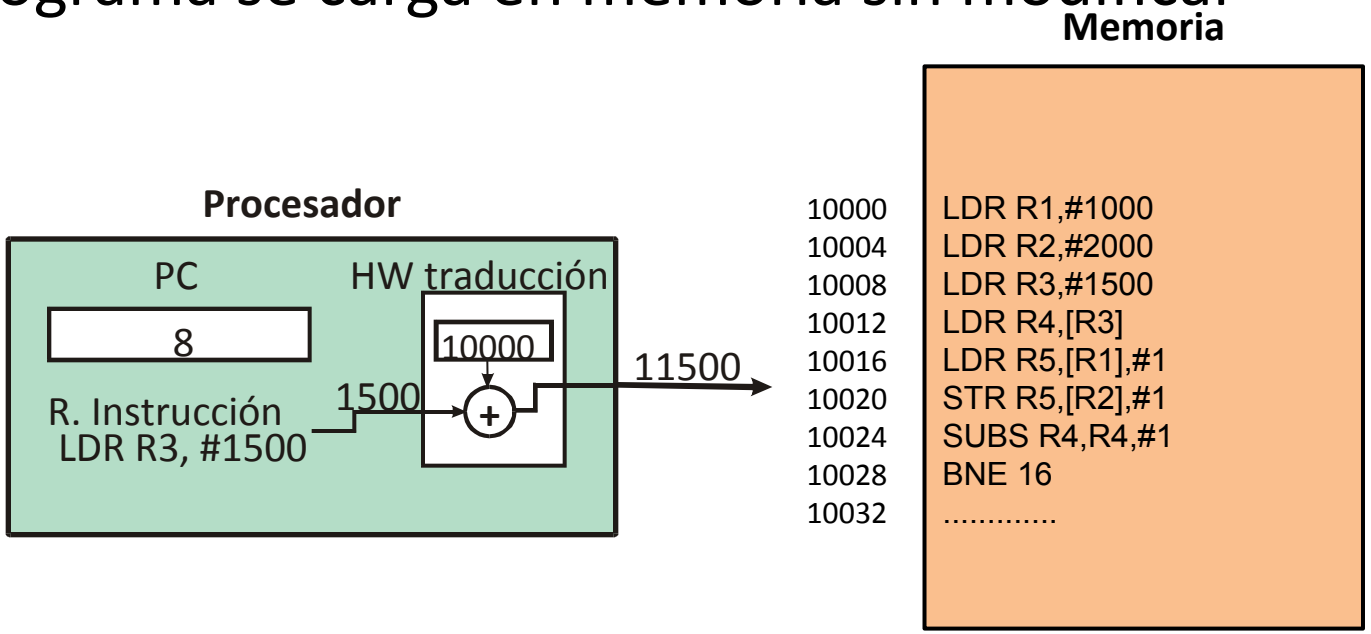
Reubicación software

- Traducción de direcciones durante carga del programa
- Programa en memoria distinto del ejecutable
- Desventajas:
  - No asegura protección
  - No permite mover programa en tiempo de ejecución



Reubicación hardware

- Hardware (MMU) encargado de traducción
- SO se encarga de:
  - Almacena por cada proceso su función de traducción
  - Especifica al hardware qué función aplicar para cada proceso
- El programa se carga en memoria sin modificar

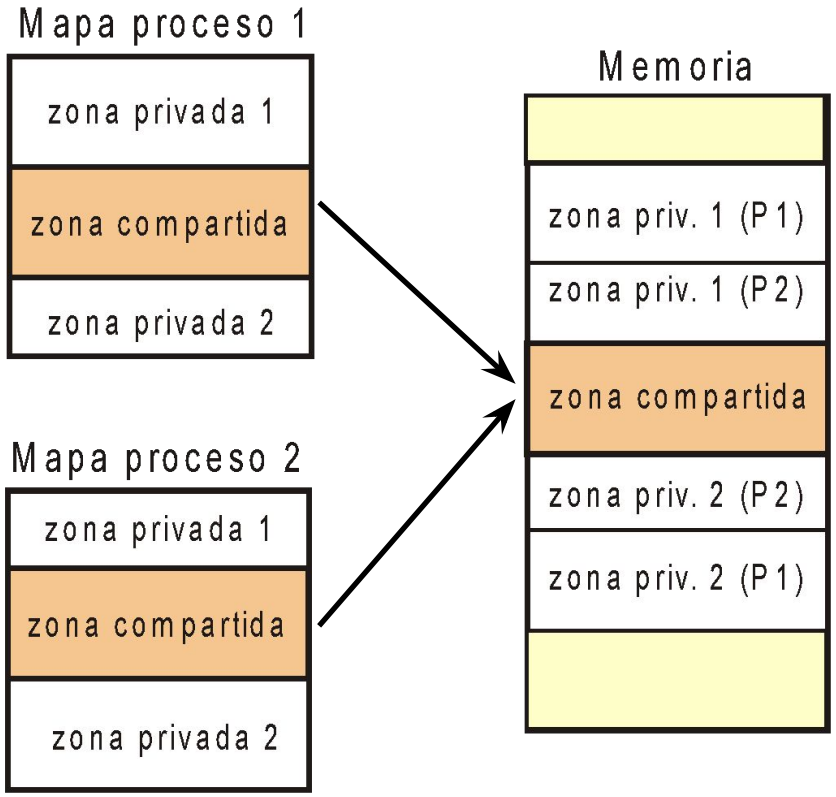


Protección

- Monoprogramación: Proteger al SO
- Multiprogramación: Además procesos entre sí
- La traducción debe crear espacios disjuntos
- Es necesario validar todas las direcciones que genera el programa
  - La detección debe realizarla el HW del procesador
  - El tratamiento lo hace el SO
- En sistemas con mapa de E/S y memoria común:
  - Traducción permite impedir que procesos accedan directamente a dispositivos de E/S

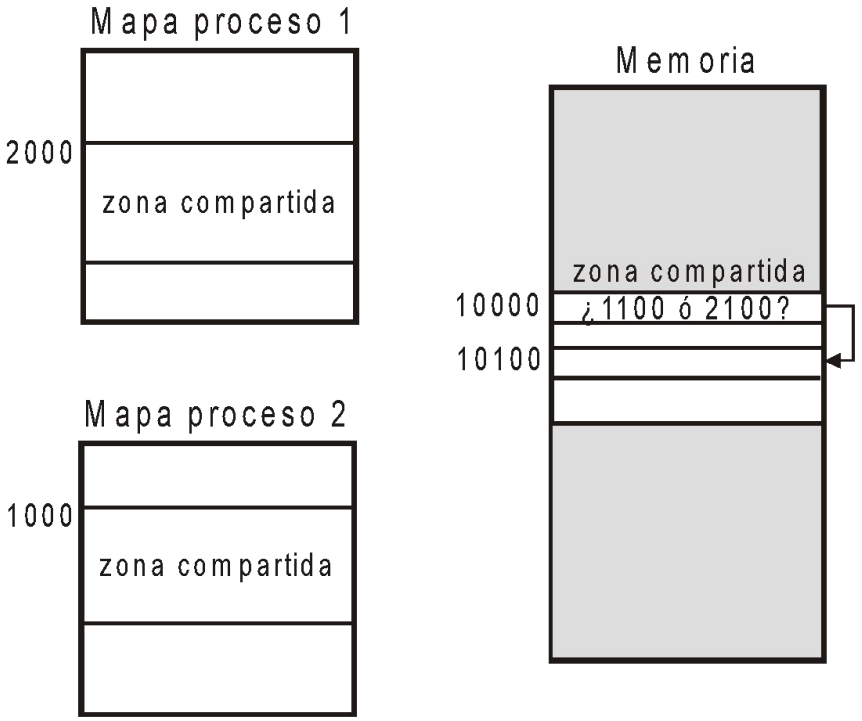
Compartición de memoria

- Direcciones lógicas de 2 o más procesos se corresponden con misma dirección física
- Bajo control del SO
- Beneficios:
  - Procesos ejecutando mismo programa comparten su código
  - Mecanismo de comunicación entre procesos muy rápido
- Requiere asignación no contigua



Problemas al compartir memoria

- Si una posición de la zona compartida contiene referencia absoluta a otra posición de la zona compartida
- Ejemplo con zonas de código:
  - Zona compartida contiene instrucción de bifurcación a instrucción dentro de la zona
- Ejemplo con zonas de datos:
  - Zona contiene una lista con punteros



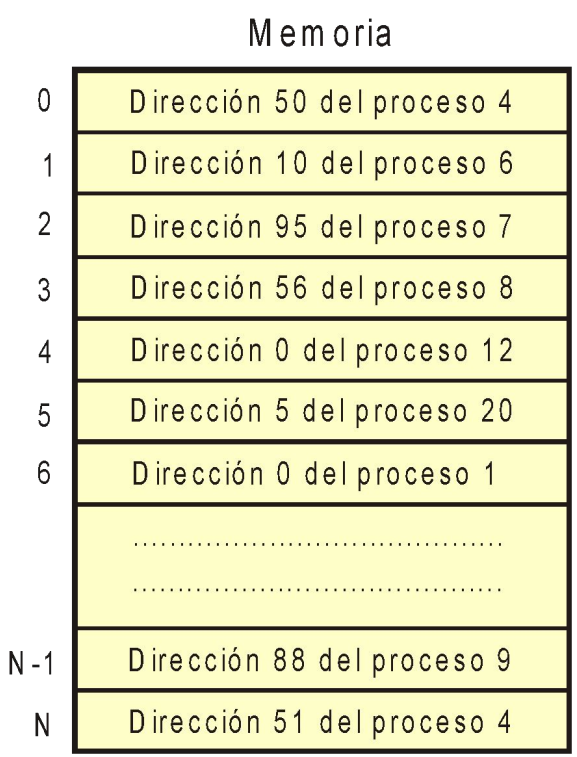
Soporte de regiones

- Mapa de proceso no homogéneo
  - Conjunto de regiones con distintas características
  - Ejemplo: Región de código no modificable
- Mapa de proceso dinámico
  - Regiones cambian de tamaño (p.ej. pila)
  - Se crean y destruyen regiones
  - Existen zonas sin asignar (huecos)
- Gestor de memoria debe dar soporte a estas características:
  - Detectar accesos no permitidos a una región
  - Detectar accesos a huecos
  - Evitar reservar espacio para huecos
- SO debe guardar una tabla de regiones para cada proceso

Maximizar rendimiento

- Reparto de memoria maximizando grado de multiprogramación
- Se "desperdicia" memoria debido a:
  - "Restos" inutilizables (fragmentación)
  - Tablas requeridas por gestor de memoria
- Menor fragmentación implica tablas más grandes
- Compromiso → Paginación
- Uso de memoria virtual para aumentar grado de multiprogramación

Aprovechamiento de memoria óptimo e irrealizable



Mapas de memoria muy grandes para procesos

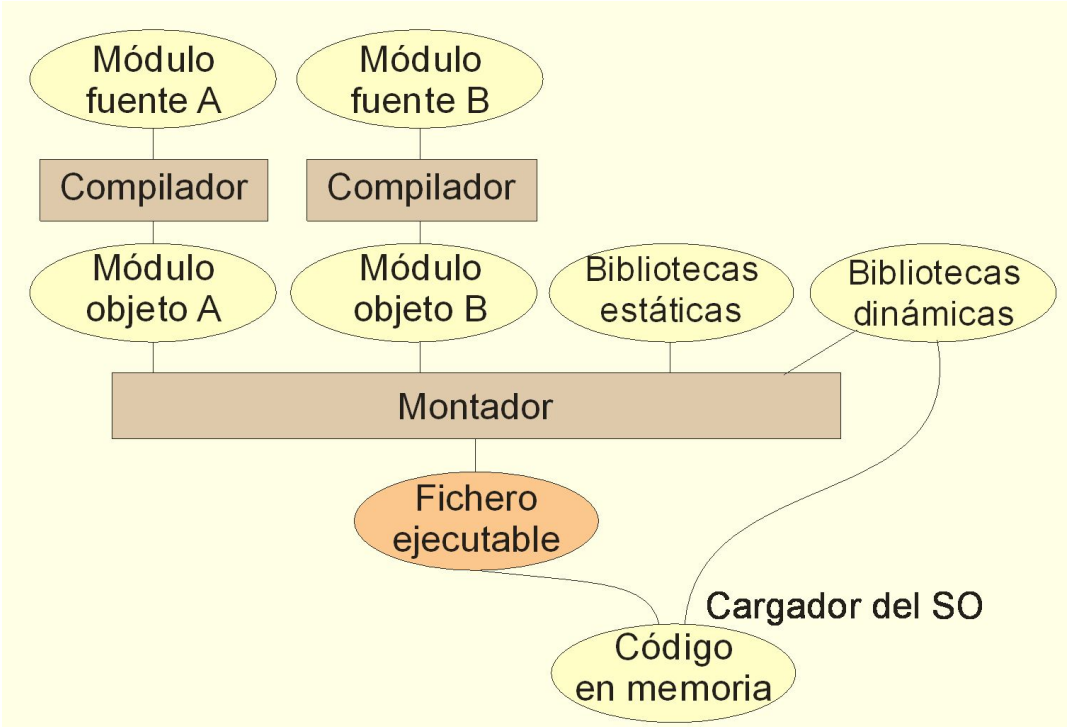
- Procesos necesitan cada vez mapas más grandes
  - Aplicaciones más avanzadas o novedosas
- Resuelto gracias al uso de memoria virtual
- Antes se usaban *overlays*:
  - Programa dividido en fases que se ejecutan sucesivamente
  - En cada momento sólo hay una fase residente en memoria
  - Cada fase realiza su labor y carga la siguiente
  - No es transparente ya que toda la labor la realizaba el programador

Contenido

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Asignación contigua
  - Intercambio (*swapping*)
- Memoria virtual
  - Políticas de intercambio
- Gestión de memoria en Linux

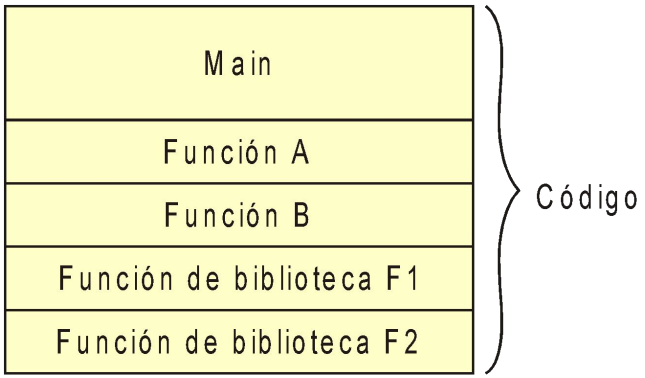
Fases en generación de ejecutable

- Aplicación: conjunto de módulos en lenguaje de alto nivel
- Procesado en dos fases: Compilación y Montaje
- Compilación:
  - Resuelve referencias dentro cada módulo fuente
  - Genera módulo objeto
- Montaje (o enlace):
  - Resuelve referencias entre módulos objeto
  - Resuelve referencias a símbolos de bibliotecas
  - Genera ejecutable incluyendo bibliotecas



Bibliotecas de objetos

- Colección de módulos objeto relacionados (sistema o usuario)
- Estáticas: el *linker* enlaza los módulos objeto del programa y de las bibliotecas creando un ejecutable autocontenido
- Desventajas del montaje estático:
  - Ejecutables grandes
  - Código de biblioteca repetido en muchos ejecutables y memoria
  - Actualización de biblioteca implica volver a montar



Bibliotecas dinámicas

- Carga y montaje en tiempo de ejecución
- El ejecutable contiene:
  - Nombre de la biblioteca
  - Rutina de carga y montaje en tiempo de ejecución
- En la 1ª referencia a un símbolo de la biblioteca:
  - Se carga y monta biblioteca correspondiente
  - Ajusta instrucción que realiza referencia para que próximas referencias accedan a símbolo de biblioteca
    - Problema: Se modificaría el código del programa
    - Solución típica: Referencia indirecta mediante una tabla

Pros y Contras

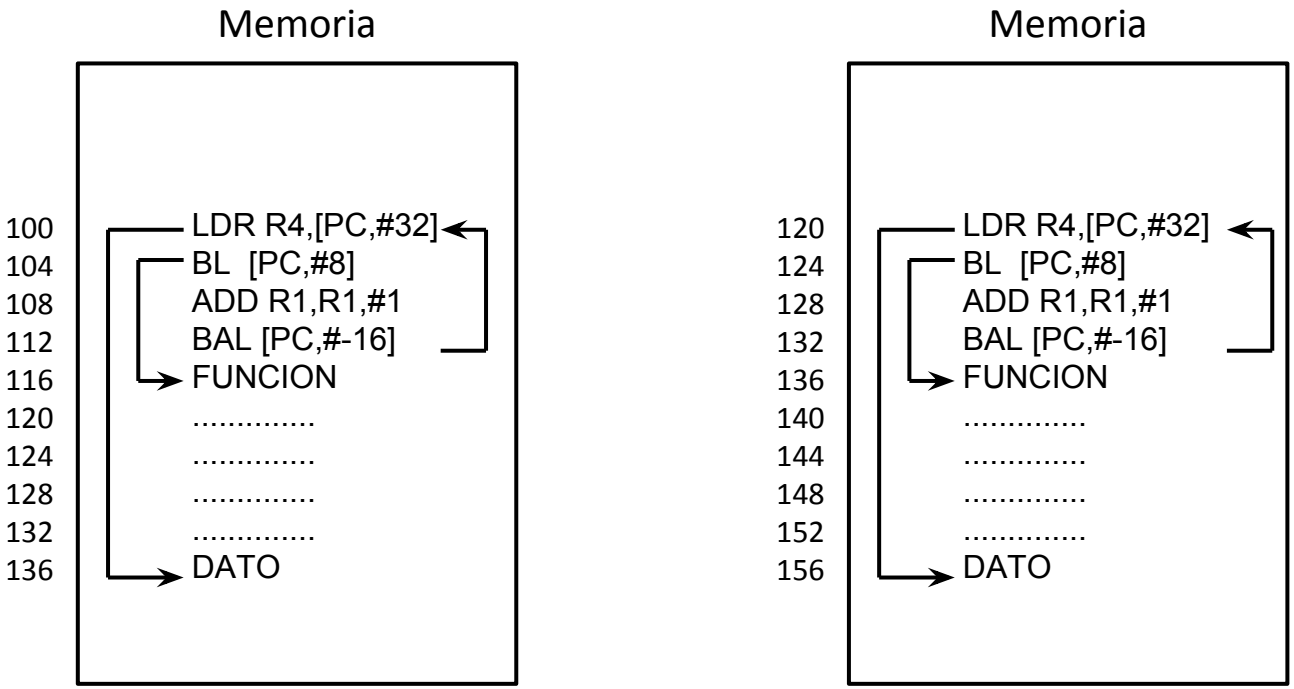
- Ventajas:
  - Menor tamaño ejecutables
  - Código de rutinas de biblioteca sólo en archivo de biblioteca
  - Procesos pueden compartir código de biblioteca
  - Actualización automática de bibliotecas: Uso de versiones
- Desventajas:
  - Mayor tiempo de ejecución debido a carga y montaje
    - Tolerable, compensa el resto de las ventajas
  - Ejecutable no autocontenido
- Uso de bibliotecas dinámicas es transparente
  - Mandatos de compilación y montaje igual que con estáticas

Compartir bib. dinámicas

- Biblioteca dinámica contiene referencias internas
  - Problema de zona compartida con autoreferencias
- Tres posibles soluciones:
  1. A cada bib. dinámica se le asigna rango de direcciones fijo
     - Inconveniente: poco flexible
  2. En montaje en t. de ejecución se reajustan autoreferencias
     - Inconveniente: impide compartir código de biblioteca
  3. Crear biblioteca con código independiente de posición (PIC)
     - Se genera código con direccionamiento relativo a registro
     - Inconveniente (tolerable): dir. relativo menos eficiente

Compartir bib. dinámicas

- Código independiente de posición (PIC)



Recuerda: comandos útiles

- /sbin/ldconfig –p
  - Muestra la lista de todas las librerías cargadas.
- ldd
  - Permite ver las librerías con las que hemos enlazado

Ejemplo: ldd

saludo.c

```
#include <stdio.h>

int main (void) {
   char nombre[100];

   printf("Escribe tu nombre: ");
   if (scanf(" %s", nombre) != 1) {
      printf("Error o Fin \n");
      return 1;
   } else {
      printf("Hola %s\n",
nombre);
      return 0;
      }
}
```

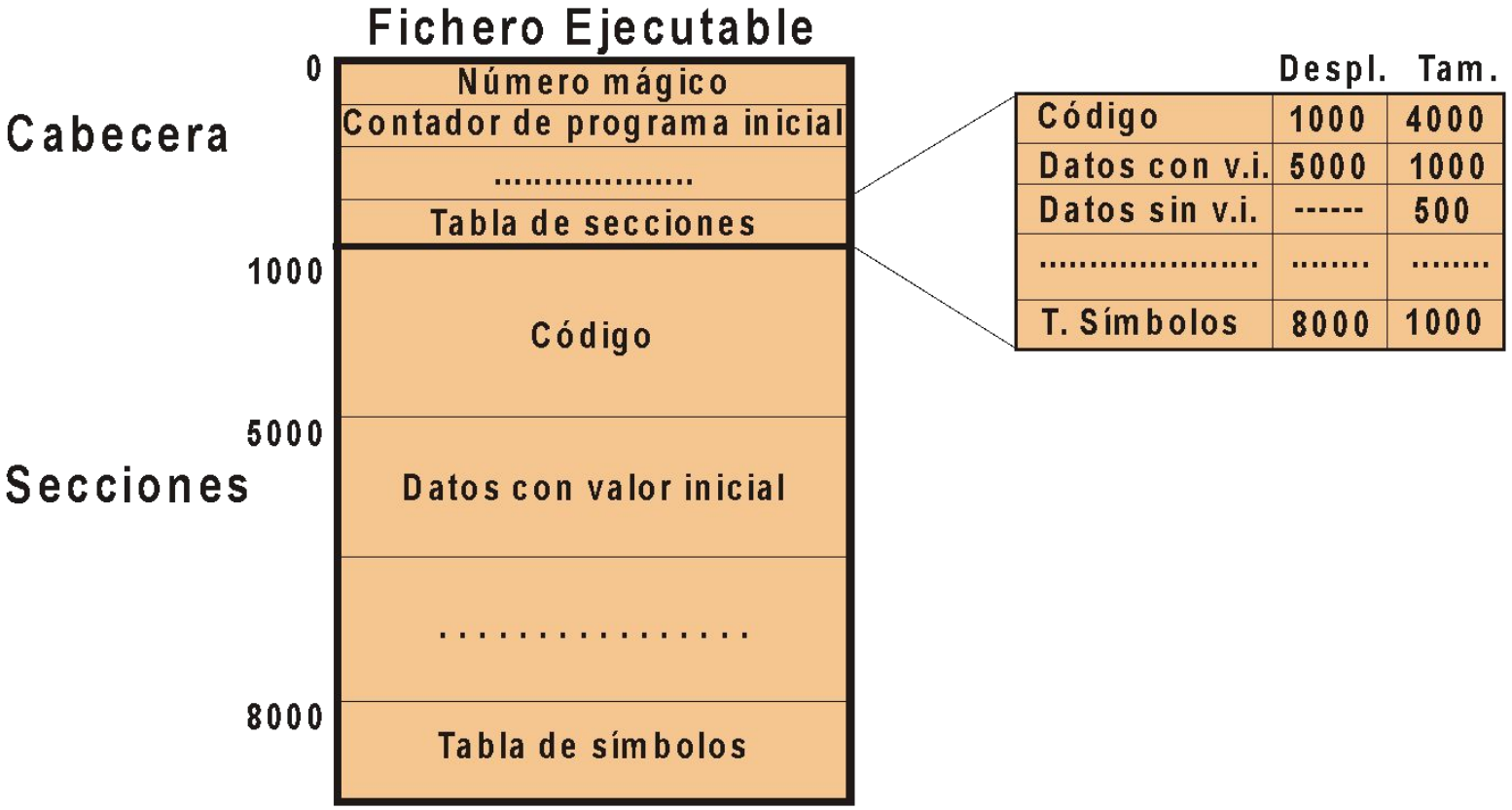
```
usuarioso@debian:~/workspace$ ldd -v saludo
   linux-vdso.so.1 =>  (0x00007fffb7b79000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4f4b470000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f4f4b818000)

   Version information:
   ./saludo:
      libc.so.6 (GLIBC_2.7) => /lib/x86_64-linux-gnu/libc.so.6
      libc.so.6 (GLIBC_2.2.5) => /lib/x86_64-linux-gnu/libc.so.6
   /lib/x86_64-linux-gnu/libc.so.6:
      ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
     ld-linux-x86-64.so.2 (GLIBC_PRIVATE) => /lib64/ld-linux-x86-64.so.2
```

Formato del ejecutable

- En UNIX *Executable and Linkable Format* (ELF).
- Estructura simplificada:
  - Cabecera:
    - Número mágico que identifica a ejecutable (0x7f+'ELF')
    - Punto de entrada del programa
    - Tabla de secciones
  - Secciones:
    - Código (text)
    - Datos inicializados
    - Datos no inicializados (sólo se especifica el tamaño)
    - Tabla de símbolos de depuración
    - Tabla de bibliotecas dinámicas

Formato del ejecutable



Variables globales versus locales

- Variables globales (con sección)
  - Estáticas
  - Se crean al iniciarse programa
  - Existen durante ejecución del mismo
  - Dirección fija en memoria y en ejecutable
- Variables locales y parámetros (sin sección)
  - Dinámicas
  - Se crean al invocar función
  - Se destruyen al retornar
  - La dirección se calcula en tiempo de ejecución
  - Recursividad: varias instancias de una variable

Variables globales versus locales

- Ejemplo:

```
int x=8;        /* Variable global con valor inicial */
int y;          /* Variable global sin valor inicial */

extern int j; /* Declaración de variable global a todos
         los módulos */
int j=5;     /* Inicialización y reserva de espacio para
variable global */

static int k; /* Variable visible en este fichero*/

int f(int t){      /* Parámetro */
   int z;     /* Variable local */
     .......
}
main(){
     .......
}
```

Mapa de memoria de un proceso

- Conjunto de regiones:
  - Zona contigua tratada como unidad
  - Posee una información asociada:
    - Dirección de comienzo y tamaño inicial
    - Soporte: donde se almacena su contenido inicial si lo tuviese (fichero)
    - Protección: RWX
    - Uso compartido o privado
    - Tamaño fijo o variable (modo de crecimiento ↑↓)

Recuerda: comandos útiles

- nm / objdump
  - Permiten visualizar partes de un ejecutable
  - Opciones típicas de objdump: -S, -t, -f, -h
  - Ejemplo:

```
usuarioso@debian:~/workspace$ nm -Dsv saludo
         w __gmon_start__
         U __isoc99_scanf
         U __libc_start_main
         U printf
         U puts
```

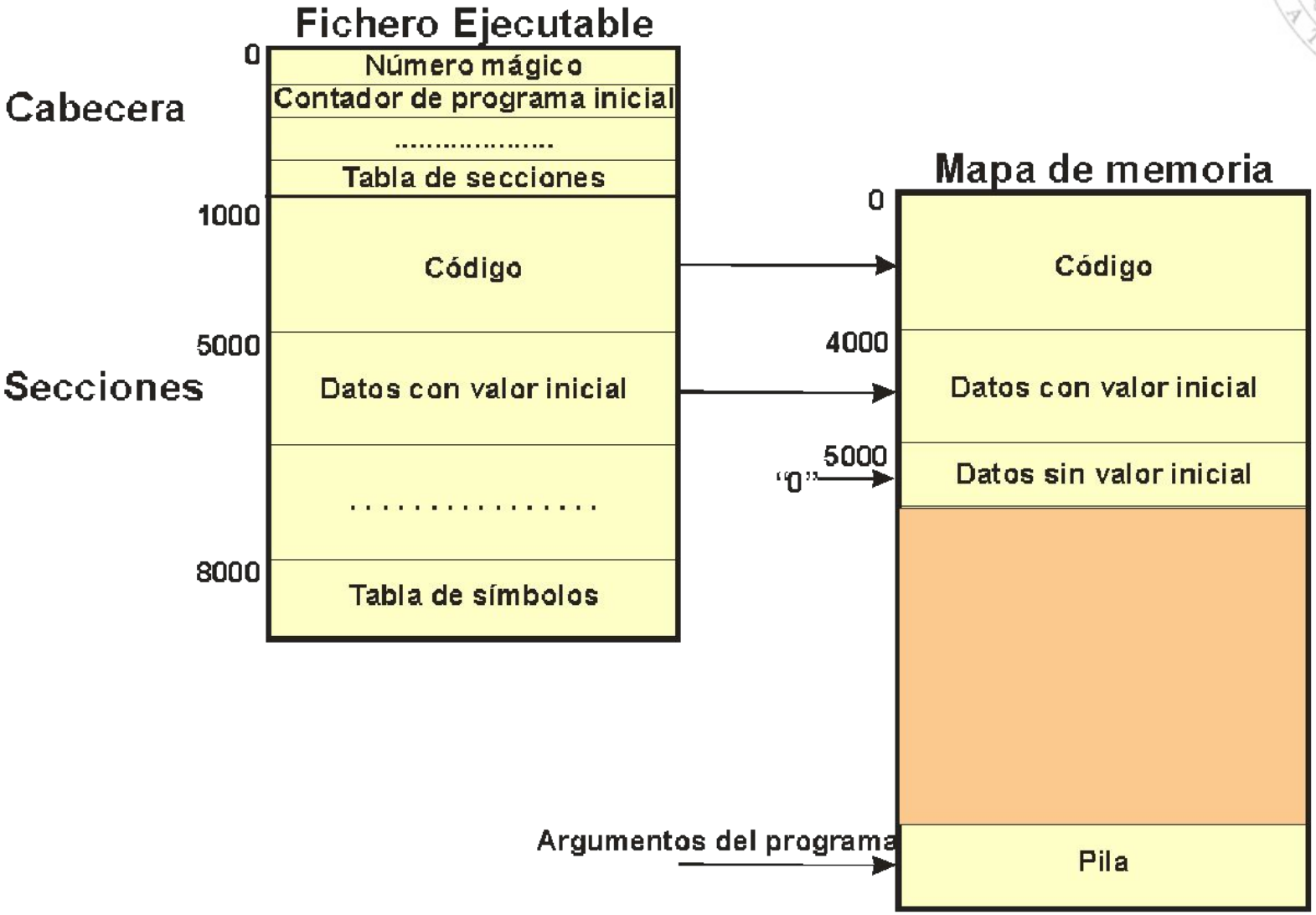
Recuerda: objdump

```
usuarioso@VMWare-debian6:~/workspace$ objdump -x saludo
saludo:     file format elf64-x86-64
saludo
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000004004c0
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 13 .text         0000020c  00000000004004c0  00000000004004c0  000004c0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .rodata       00000032  00000000004006d8  00000000004006d8  000006d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 20 .jcr          00000008  00000000006007f0  00000000006007f0  000007f0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .dynamic      000001e0  00000000006007f8  00000000006007f8  000007f8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .got          00000008  00000000006009d8  00000000006009d8  000009d8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 23 .got.plt      00000038  00000000006009e0  00000000006009e0  000009e0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000600a18  0000000000600a18  00000a18  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000008  0000000000600a28  0000000000600a28  00000a28  2**2
                  ALLOC
```

Crear mapa de memoria desde ejecutable

- Ejecución de un programa: Crea mapa a partir de ejecutable
  - Regiones de mapa inicial → Secciones de ejecutable
- Código
  - Compartida, RX, T. Fijo, Soporte en Ejecutable
- Datos con valor inicial
  - Privada, RW, T. Fijo, Soporte en Ejecutable
- Datos sin valor inicial
  - Privada, RW, T. Fijo, Sin Soporte (rellenar 0)
- Pila
  - Privada, RW, T. Variable, Sin Soporte (rellenar 0)
  - Crece hacia direcciones más bajas
  - Pila inicial: argumentos del programa (*execvp*)

Crear mapa desde ejecutable



Otras regiones del mapa de memoria

- Durante ejecución de proceso se crean nuevas regiones
  - Mapa de memoria tiene un carácter dinámico
- Región de Heap
  - Soporte de memoria dinámica (`malloc` en C)
  - Privada, RW, T. Variable, Sin Soporte (rellenar 0's)
  - Crece hacia direcciones más altas
- Archivo proyectado
  - Región asociada al archivo proyectado
  - T. Variable, Soporte en Archivo
  - Protección y carácter compartido o privado especificado en proyección

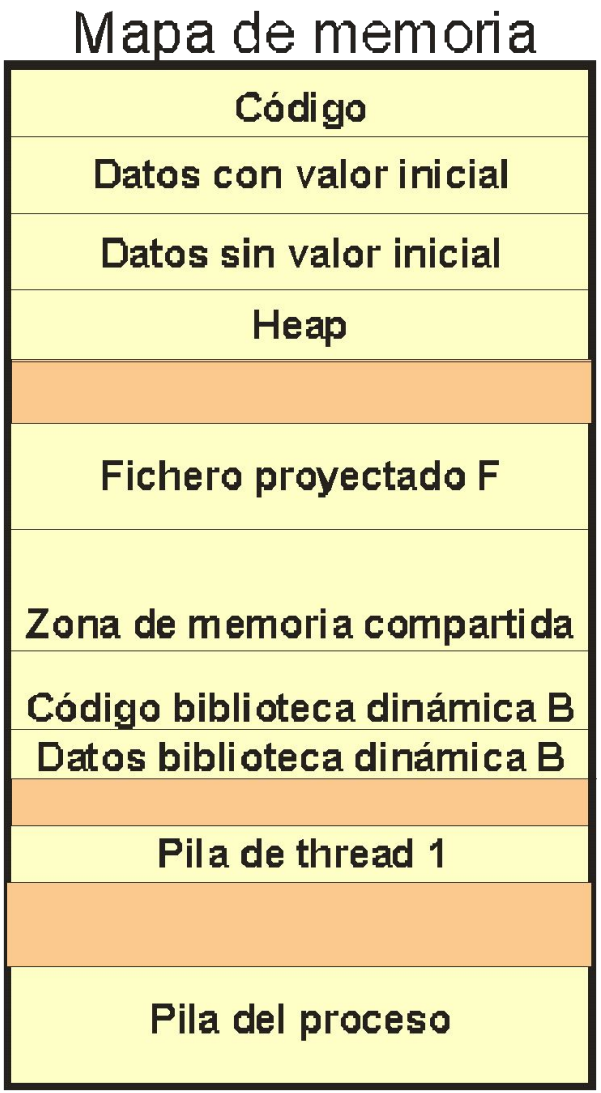
Otras regiones del mapa de memoria

- Memoria compartida
  - Región asociada a la zona de memoria compartida
  - Compartida, T. Variable, Sin Soporte (rellenar 0's)
  - Protección especificada en proyección
- Pilas de threads
  - Cada pila de thread corresponde con una región
  - Mismas características que pila del proceso
- Regiones de bibliotecas dinámicas
  - Se crean regiones asociadas al código y datos de la biblioteca

Características de regiones

Región	Soporte	Protección	Comp/Priv	Tamaño
Código	Fichero	RX	Compartida	Fijo
Dat. con v.i.	Fichero	RW	Privada	Fijo
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo
Pilas	Sin soporte	RW	Privada	Variable
Heap	Sin soporte	RW	Privada	Variable
F. Proyect.	Fichero	por usuario	Comp./Priv.	Variable
M. Comp.	Sin soporte	por usuario	Compartida	Variable

Mapa de memoria de un proceso hipotético



Recuerda: /proc

```
usuarioso@VMWare-debian6:~/workspace$ ps –a |grep saludo
6739 pts/0    S+     0:00 ./saludo

usuarioso@VMWare-debian6:/proc/6739$ cat statm
986 79 61 1 0 48 0

usuarioso@VMWare-debian6:/proc/6739$ cat maps
00400000-00401000 r-xp 00000000 08:01 947007                    /home/usuarioso//tmp/saludo
00600000-00601000 rw-p 00000000 08:01 947007                    /home/usuarioso//tmp/saludo
7fc50c05a000-7fc50c1db000 r-xp 00000000 08:01 668280            /lib/x86_64-linux-gnu/libc-2.13.so
7fc50c1db000-7fc50c3db000 ---p 00181000 08:01 668280            /lib/x86_64-linux-gnu/libc-2.13.so
7fc50c3db000-7fc50c3df000 r--p 00181000 08:01 668280            /lib/x86_64-linux-gnu/libc-2.13.so
7fc50c3df000-7fc50c3e0000 rw-p 00185000 08:01 668280            /lib/x86_64-linux-gnu/libc-2.13.so
7fc50c3e0000-7fc50c3e5000 rw-p 00000000 00:00 0
7fc50c3e5000-7fc50c405000 r-xp 00000000 08:01 668306            /lib/x86_64-linux-gnu/ld-2.13.so
7fc50c5e4000-7fc50c5e7000 rw-p 00000000 00:00 0
7fc50c600000-7fc50c604000 rw-p 00000000 00:00 0
7fc50c604000-7fc50c605000 r--p 0001f000 08:01 668306            /lib/x86_64-linux-gnu/ld-2.13.so
7fc50c605000-7fc50c606000 rw-p 00020000 08:01 668306            /lib/x86_64-linux-gnu/ld-2.13.so
7fc50c606000-7fc50c607000 rw-p 00000000 00:00 0
7fffcfc45000-7fffcfc66000 rw-p 00000000 00:00 0                 [stack]
7fffcfd8a000-7fffcfd8b000 r-xp 00000000 00:00 0                 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0         [vsyscall]
```

Operaciones sobre regiones

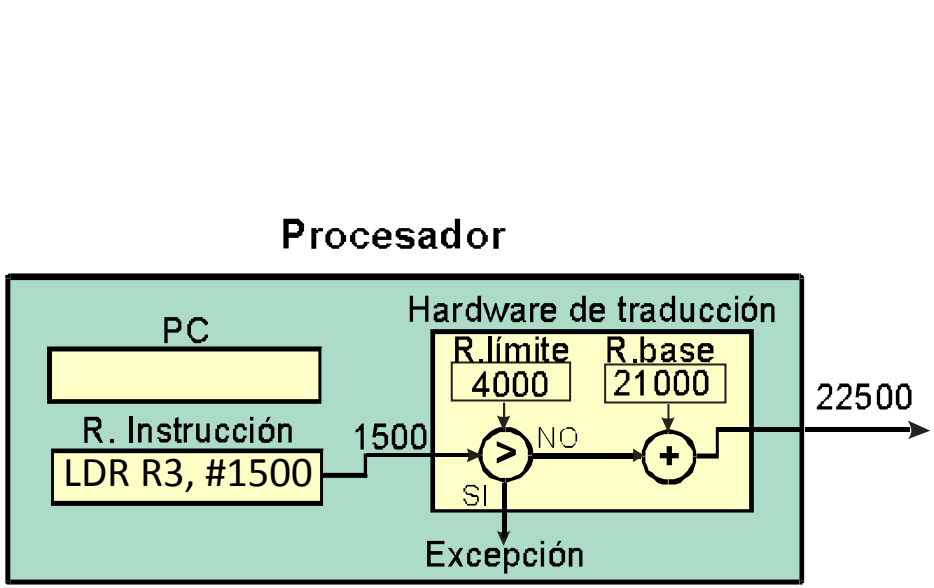
- Para estudiar evolución del mapa de memoria se pueden distinguir las siguientes operaciones:
  - Crear región
    - Implícitamente al crear mapa inicial o por solicitud del programa en t. de ejecución (p.ej. proyectar un archivo)
  - Eliminar región
    - Implícitamente al terminar el proceso o por solicitud del programa en t. de ejecución (p.ej. desproyectar un archivo)
  - Cambiar tamaño de la región
    - Implícitamente para la pila o por solicitud del programa para el heap
  - Duplicar región
    - Operación requerida por el servicio *fork* de POSIX

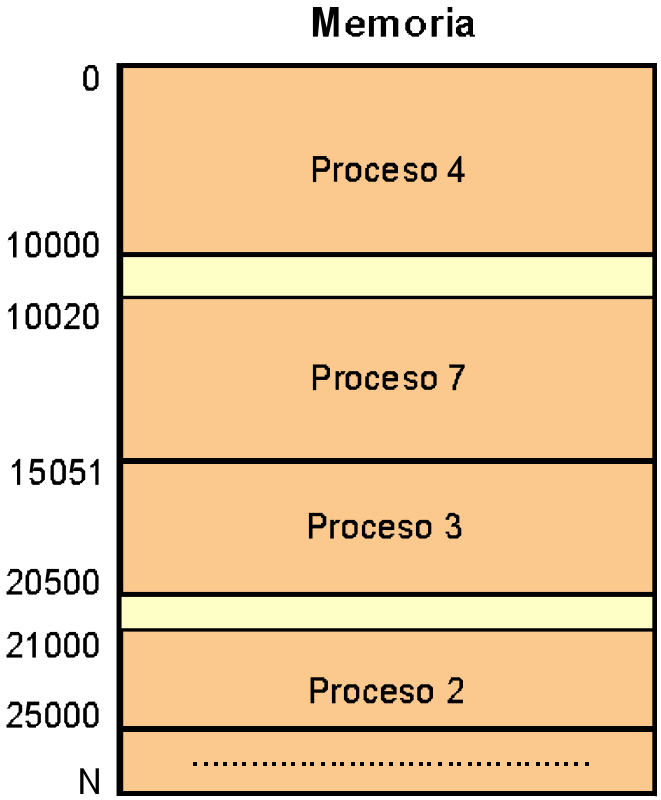
Contenido

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Asignación contigua
  - Intercambio (*swapping*)
- Memoria virtual
  - Políticas de intercambio
- Gestión de memoria en Linux

Asignación contigua

- Mapa de proceso en zona contigua de memoria principal
- Hardware requerido: Regs. valla (R. base y R. límite)
  - Sólo accesibles en modo privilegiado.





Asignación contigua

- SO mantiene información sobre:
  - El valor de regs. valla de cada proc. en su BCP
    - En c. contexto SO carga en regs. valor adecuado
  - Estado de ocupación de la memoria
    - Estr. de datos que identifiquen huecos y zonas asignadas
  - Regiones de cada proceso
- Este esquema presenta fragmentación externa:
  - Se generan pequeños fragmentos libres entre zonas asignadas
  - Posible solución: compactación –› proceso costoso

Política de asignación de espacio

- ¿Qué hueco usar para satisfacer una petición?
- Posibles políticas:
  - Primer ajuste: Asignar el primer hueco encontrado
  - Mejor ajuste: Asignar el menor hueco posible
    - Lista ordenada por tamaño o buscar en toda la lista
  - Peor ajuste : Asignar el mayor hueco con tamaño suficiente
    - Lista ordenada por tamaño o buscar en toda la lista
- Primer ajuste es más eficiente y proporciona buen aprovechamiento de la memoria
- Estrategia más sofisticada: Sistema *Buddy*
  - Listas de huecos con tamaños potencias de 2

Operaciones sobre regiones con a. contigua

- Al crear proceso se le asigna zona de mem. de tamaño fijo
  - Suficiente para albergar regiones iniciales
  - Con huecos para permitir cambios de tamaño y añadir nuevas regiones (p.ej. bibliotecas dinámicas o pilas de threads)
- Difícil asignación adecuada
  - Si grande se desperdicia espacio, si pequeño se puede agotar
- Crear/liberar/cambiar tamaño usan la tabla de regiones para gestionar la zona asignada al proceso
- Duplicar región requiere crear región nueva y copiar contenido
- Limitaciones del hardware impiden compartir memoria y detectar accesos erróneos o desbordamiento de pila

Valoración del esquema de asignación contigua

- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - mediante registros valla
  - Protección:
    - mediante registros valla
  - Compartir memoria:
    - no es posible
  - Soporte de regiones:
    - no existe
    - se reserva espacio para huecos
  - Maximizar rendimiento y mapas grandes
    - mal aprovechamiento de memoria por fragmentación externa
    - no permite memoria virtual

Intercambio

- ¿Qué hacer si no caben todos los programas en mem. principal? → *swapping*
- *Swap*: partición (fichero) de disco que almacena imágenes de procesos
- *Swap out*:
  - Cuando no caben en memoria los procesos activos, se expulsa un proceso de memoria copiando imagen a *swap*
  - Diversos criterios de selección del proceso a expulsar
    - P.ej. Dependiendo de prioridad del proceso
    - Preferiblemente un proceso bloqueado
    - No expulsar si está activo DMA sobre mapa del proceso
  - No es necesario copiar todo el mapa:
    - El código está en el fichero
    - Los huecos no contienen información del proceso

Intercambio

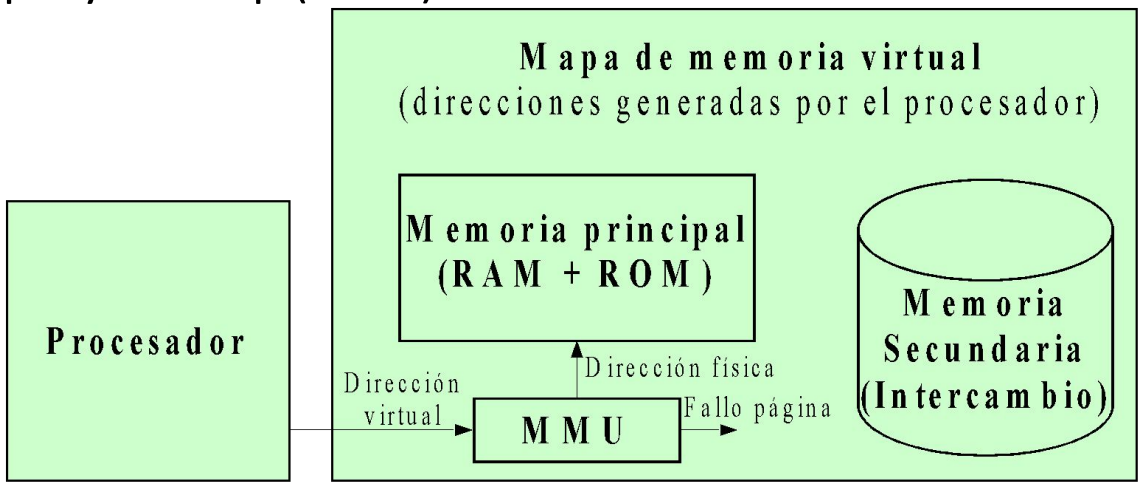
- *Swap in*:
  - Cuando haya espacio en memoria principal, se lee el proceso a memoria copiando la imagen desde *swap*
  - También cuando un proceso lleva un cierto tiempo expulsado
    - En este caso antes de *swap in*, hay *swap out* de otro
- Asignación de espacio en el dispositivo de *swap*:
  - Con preasignación: se asigna espacio al crear el proceso
  - Sin preasignación: se asigna espacio al expulsarlo
- Los procesos activos han de residir completamente en MP.
  - Grado de multiprogamación depende del tamaño de proc. y MP
- Usado en primeras versiones de UNIX
- Solución general –› Uso de esquemas de memoria virtual
  - Aunque se sigue usando integrado con esa técnica

Contenido

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Asignación contigua
  - Intercambio (*swapping*)
- Memoria virtual
  - Políticas de intercambio
- Gestión de memoria en Linux

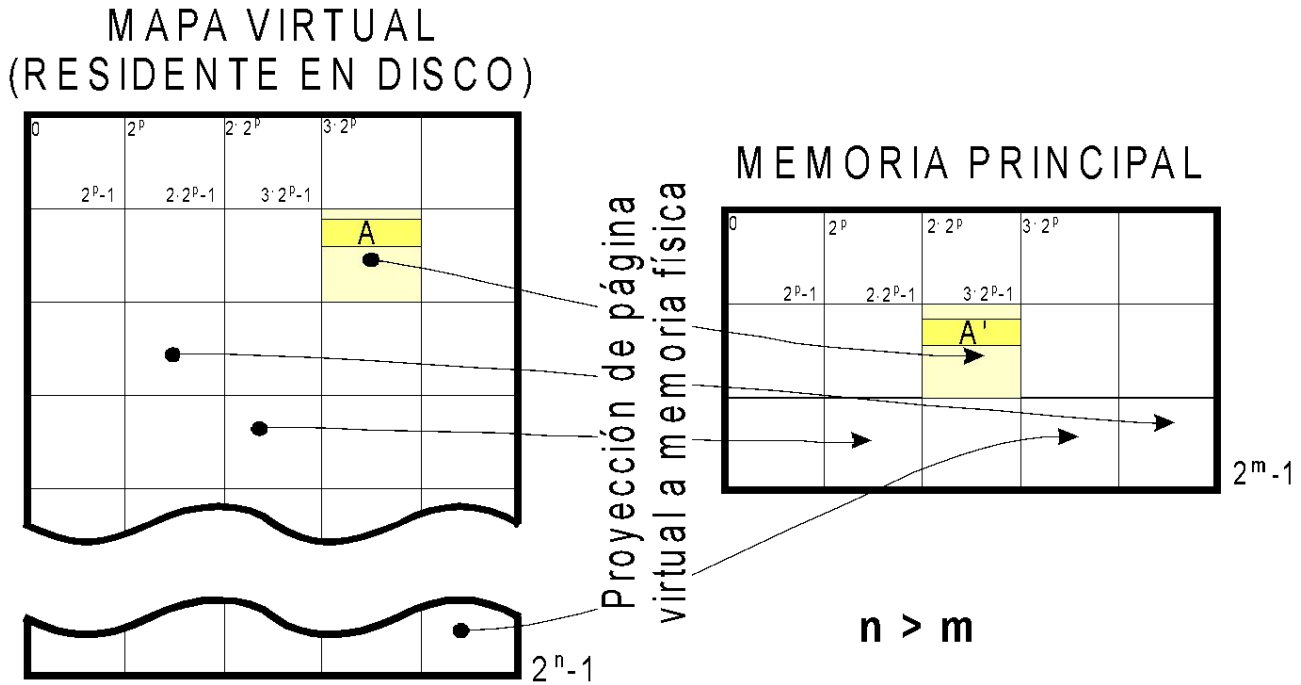
Fundamentos de la memoria virtual

- El procesador utiliza y genera direcciones virtuales
- Parte del mapa de memoria (virtual) está en disco (*swap*) y parte en MP
- La *Memory Management Unit* traduce las direcciones virtuales en físicas
- La MMU produce un fallo de página (*trap*) cuando la dirección no está en memoria principal
- El SO trata el fallo de página, haciendo un transvase entre la memoria principal y el swap (disco)



División de páginas de los espacios de memoria

- El espacio virtual se divide en páginas
- Algunas páginas están en memoria principal
  - El SO se encarga de que estén en memoria principal las páginas necesarias
  - Para ello trata los fallos de página producidos por la MMU



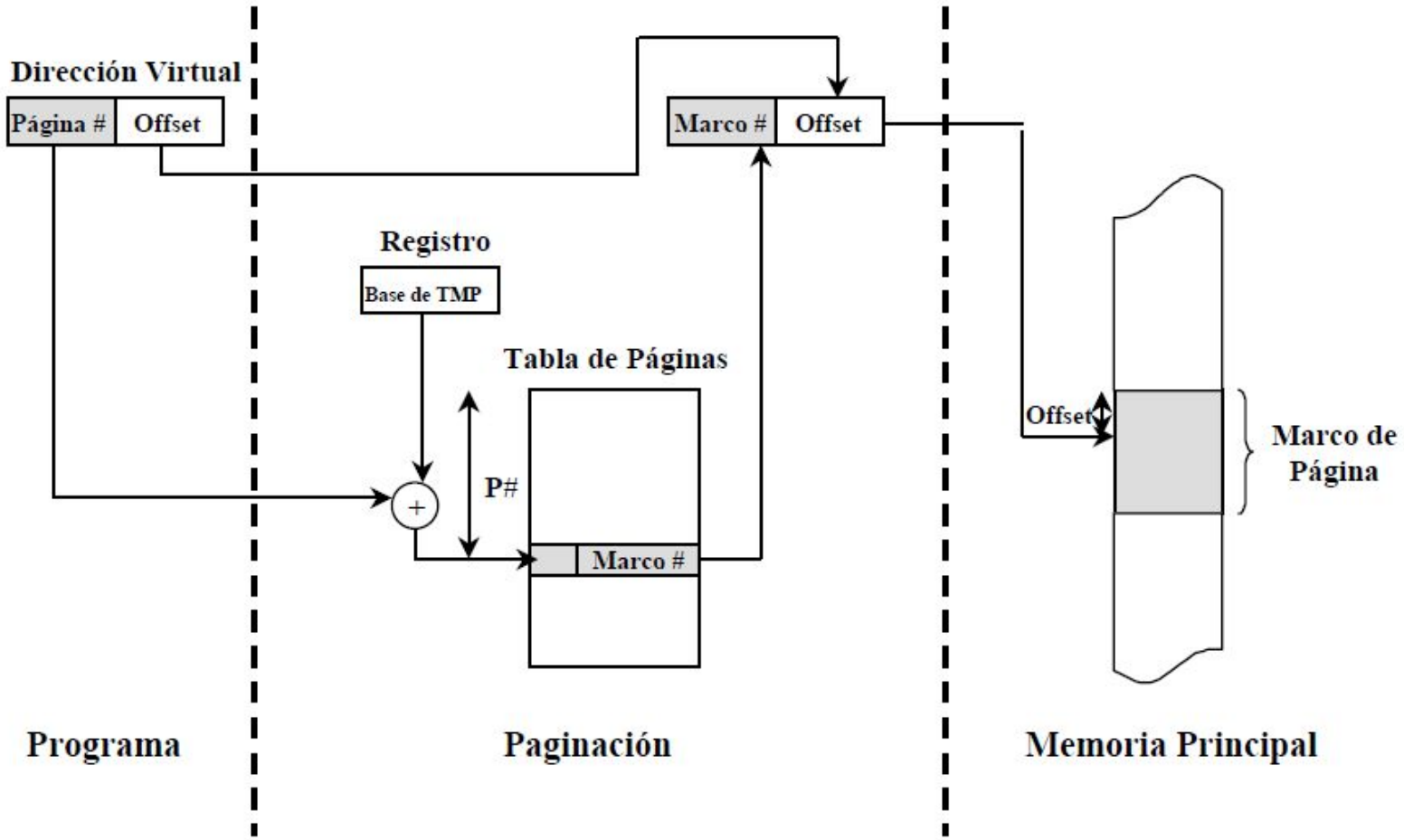
Memoria virtual

- Transferencia de *bloques* entre ambos niveles
  - De M. secundaria a principal: por demanda
  - De M. principal a secundaria: por expulsión
- Aplicable porque los procesos exhiben proximidad de referencias
  - Procesos sólo usan parte de su mapa en intervalo de tiempo
  - M. virtual: intentar que parte usada (*conjunto de trabajo*) resida en M. principal (*conjunto residente*)
- Beneficios:
  - Aumenta el grado de multiprogramación
  - Permite ejecución de programas que no quepan en mem. ppal
- No adecuada para sistemas de tiempo real

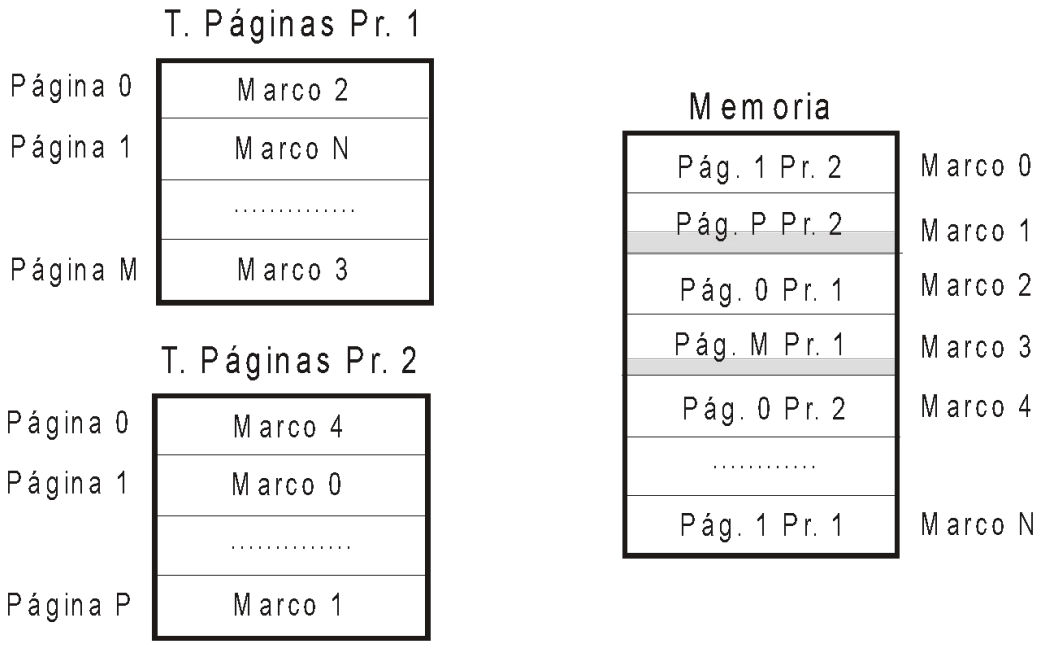
Aspectos hardware

- Asignación no contigua.
- Unidad: página (tamaño potencia de 2)
  - Mapa de memoria del proceso dividido en páginas
  - Memoria principal dividida en marcos (tam. marco=tam. página)
- Dirección lógica: nº página + desplazamiento
- Tabla de páginas (TP):
  - Relaciona cada página con el marco que la contiene
  - MMU usa TP para traducir direcciones lógicas a físicas.
- Típicamente usa 2 TPs:
  - TP usuario:
  - TP sistema:
    - Sólo se permite usar estas direcciones en modo sistema
- Si mapa de E/S y memoria común: dir. lógica –› dir. de E/S

Ejemplo de traducción con tablas de páginas



Fragmentación interna en paginación



- Mem. asignada > Mem. requerida
  - Puede desperdiciarse parte de un marco asignado

Contenido de entrada de TP

- Número de marco asociado
- Información de protección: RWX
  - Si operación no permitida –› Excepción
- Bit de página válida/inválida
  - Usado en mem. virtual para indicar si página presente
  - Si se accede –› Excepción
- Bit de página accedida (*Ref*)
  - MMU lo activa cuando se accede a esta página
- Bit de página modificada (*Mod*)
  - MMU lo activa cuando se escribe en esta página
- Bit de desactivación de cache:
  - se usa cuando la entrada corresponde con direcciones de E/S

Tamaño de la página

- Condicionado por diversos factores contrapuestos:
  - Potencia de 2 y múltiplo del tamaño del bloque de disco
  - Mejor pequeño por:
    - Menor fragmentación
    - Se ajusta mejor al conjunto de trabajo
  - Mejor grande por:
    - Tablas más pequeñas
    - Mejor rendimiento del dispositivo de E/S (disco)
  - Compromiso: entre 2K y 16K

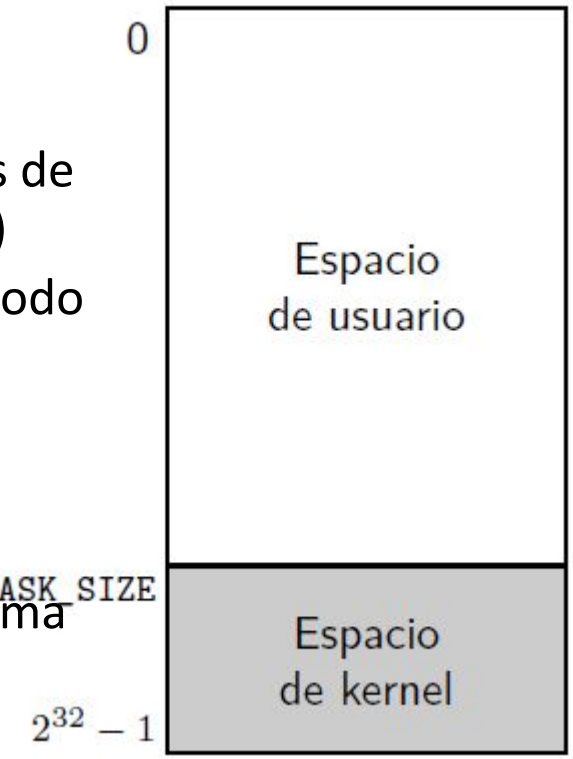
Gestión del SO

- SO mantiene una TP por cada proceso
  - En camb. contexto notifica a MMU cuál debe usar (RIED)
- SO mantiene una única TP para él mismo
  - Proceso ejecutando una llamada al sistema accede directamente a su mapa y al del SO
- SO mantiene tabla de marcos:
  - Estado de cada marco (libre u ocupado, ...)
- SO mantiene tabla de regiones por cada proceso
- Mucho mayor gasto en tablas que con asignación contigua
  - Es el precio de mucha mayor funcionalidad

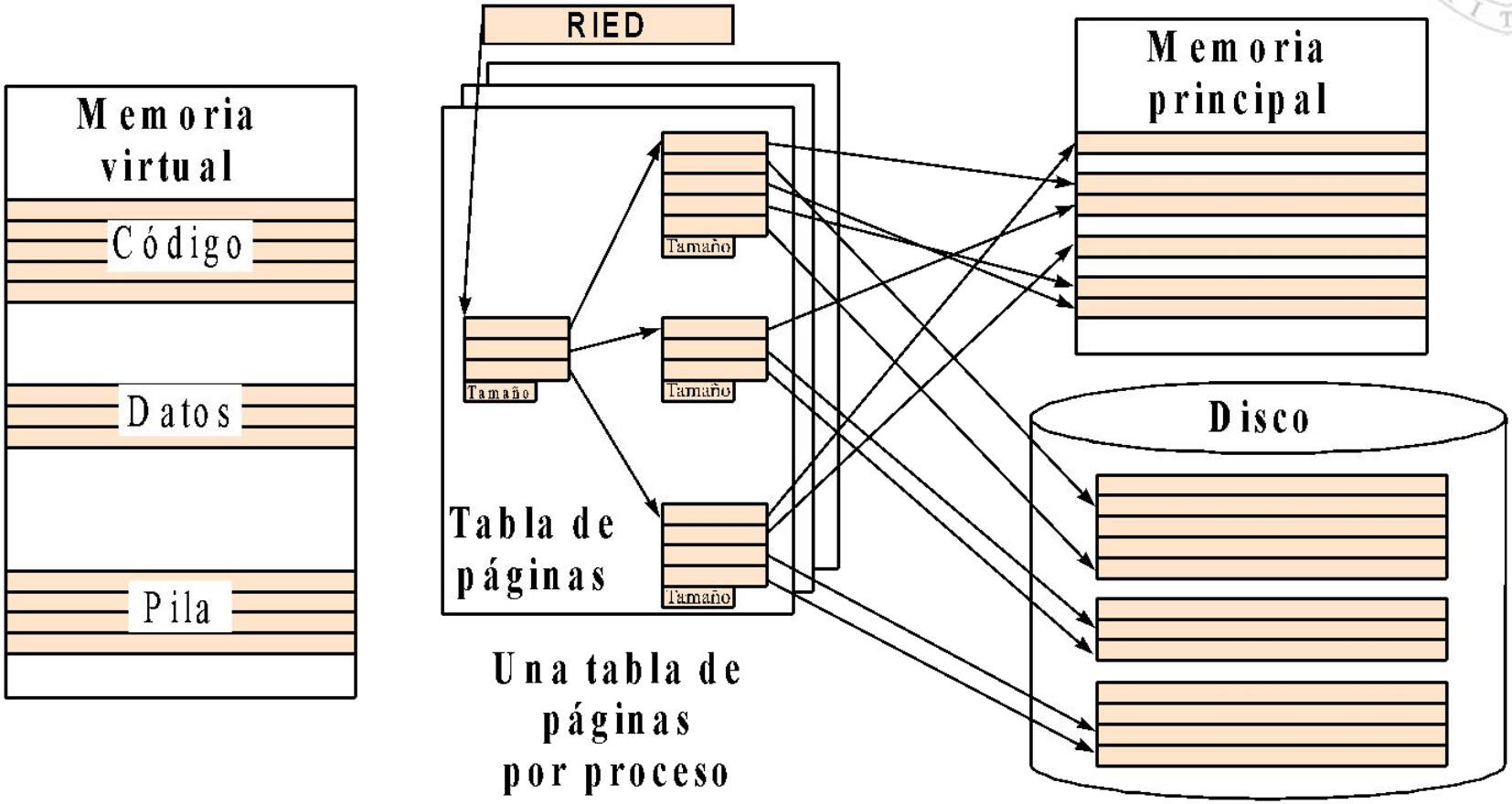
Páginas y mapa de memoria

- El SO gestiona dos tipos de página:
  - Página de usuario: almacena código o datos de un proceso
  - Página de sistema: almacena código o datos (variables y estructuras de datos) del SO

- Por simplicidad, el SO divide el mapa de memoria (virtual) del proceso en espacio de kernel (páginas de sistema) y espacio de usuario (páginas de usuario)
  - Cuando el proceso ejecuta instrucciones en modo usuario solo puede generar direcciones (referenciar páginas) del espacio de usuario
    - La MMU impide que el proceso acceda a direcciones del esp. de kernel
  - Cuando el proceso invoca una llamada al sistema (paso a modo kernel) el SO accede a ambos espacios



Tablas de páginas



Valoración de la paginación

- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - Mediante TP
  - Protección:
    - Mediante TP
  - Compartir memoria:
    - Entradas corresponden con mismo marco
  - Soporte de regiones:
    - Bits de protección
    - Bit de validez: no se reserva espacio para huecos
  - Maximizar rendimiento y mapas grandes
    - Permite esquemas de memoria virtual

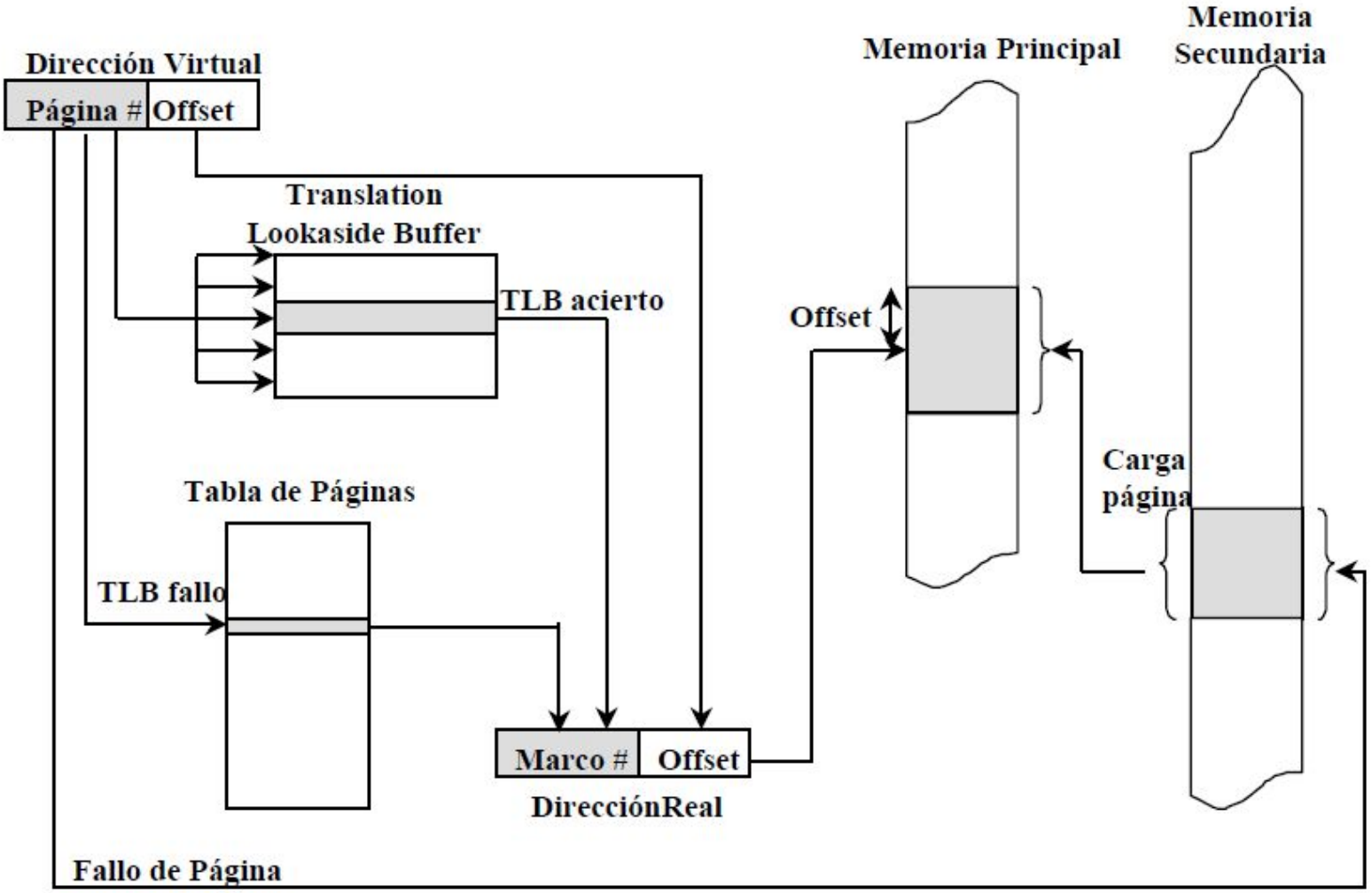
Problemática de las TP

- Eficiencia:
  - Cada acceso lógico requiere dos accesos a memoria principal:
    - A la tabla de páginas + al propio dato o instrucción
  - Solución:
    - *Cache* de traducciones –› TLB
- Gasto de almacenamiento:
  - Tablas muy grandes
    - Ejemplo: páginas 4K, dir. lógica 32 bits y 4 bytes por entrada
      - Tamaño TP: 2^{20} *4 = 4MB/proceso
  - Solución:
    - Tablas multinivel
    - Tablas invertidas

Translation Look-aside Buffer (TLB)

- Memoria asociativa con info. sobre últimas páginas accedidas
  - *cache* de entradas de TP correspondientes a estos accesos
- 2 alternativas:
  - Entradas en TLB no incluyen información sobre proceso
    - Invalidar TLB en cambios de contexto
  - Entradas en TLB incluyen información sobre proceso
    - Registro de UCP debe mantener un ident. de proceso actual
- Gestionada por HW:
  - MMU consulta TLB, si fallo usa la TP en memoria
  - "Casi" trasparente al SO, en un c. contexto:
    - Volcar de TLB a TP para actualizar *Ref* y *Mod*
    - Invalidar TLB si no incluye información del proceso
- Diseño alternativo: TLB gestionada por SW

TLB


Dirección Virtual
Página #
Offset
Translation
Lookaside Buffer
TLB acierto
Tabla de Páginas
TLB fallo
Marco #
Offset
DirecciónReal
Fallo de Página
Memoria Principal
Memoria
Secundaria
Offset
Carga
página

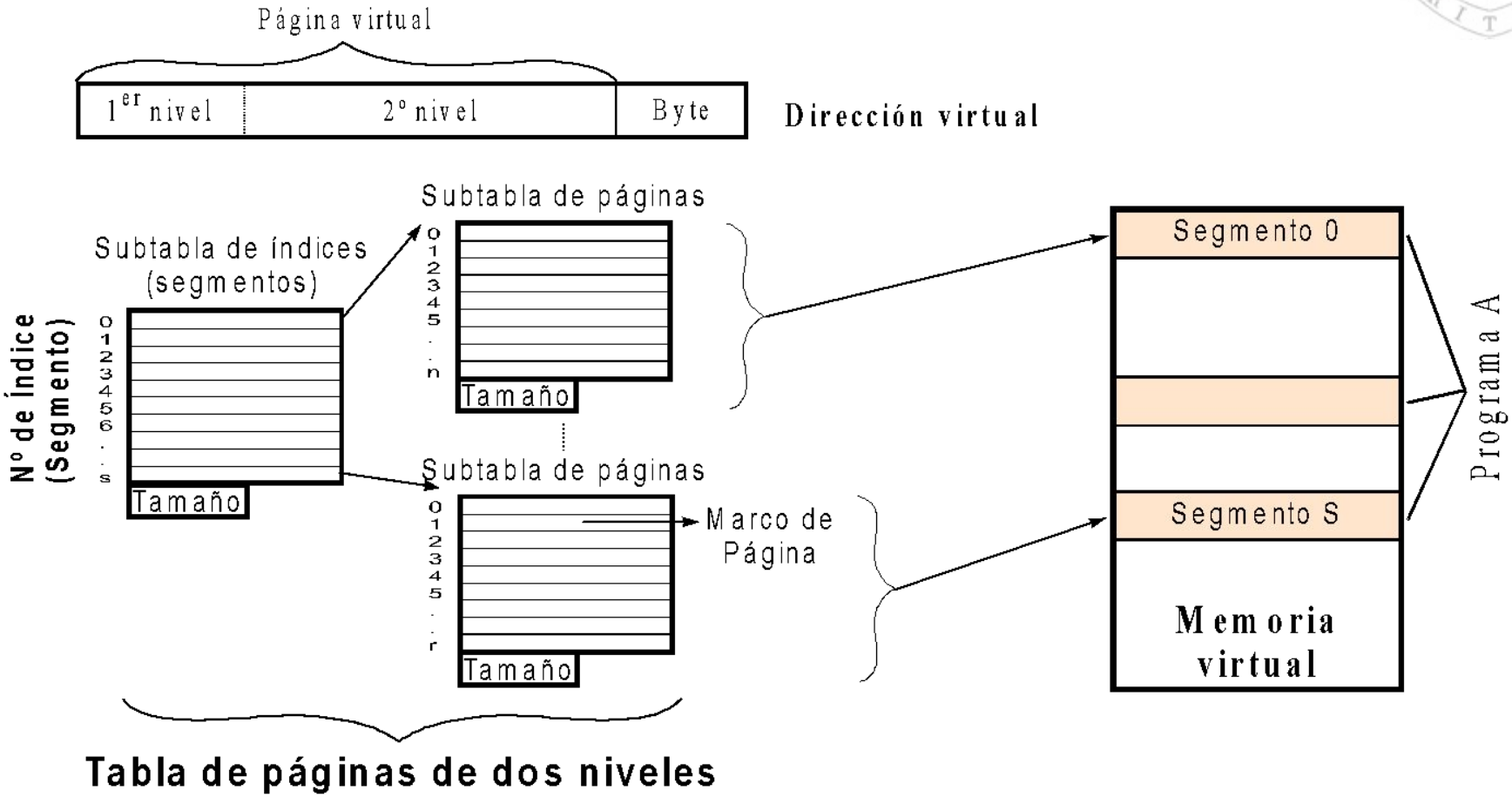
TLB gestionada por software

- Organización alternativa bastante usada actualmente
  - Traspasar al SO parte del trabajo de traducción
- MMU no usa tablas de páginas, sólo consulta TLB
- SO mantiene TPs que son independientes del HW
- Fallo en TLB –› Activa SO
- SO se encarga de:
  - Buscar “a mano” en TP la traducción
  - Rellenar (con posible reemplazo) TLB con la traducción
- Proporciona flexibilidad en diseño de SO pero menor eficiencia

TP multinivel

- Tablas de páginas organizadas en M niveles:
  - Entrada de TP de nivel K apunta a TP de nivel K+1
  - Entrada de último nivel apunta a marco de página
- Dirección lógica especifica la entrada a usar en cada nivel:
  - 1 campo por nivel + desplazamiento
- 1 acceso lógico –› M + 1 accesos a memoria
  - Uso de TLB
- Si todas las entradas de una TP son inválidas:
  - No se almacena esa TP
  - Se pone inválida la entrada correspondiente de la TP superior

Tablas de páginas de dos nivles



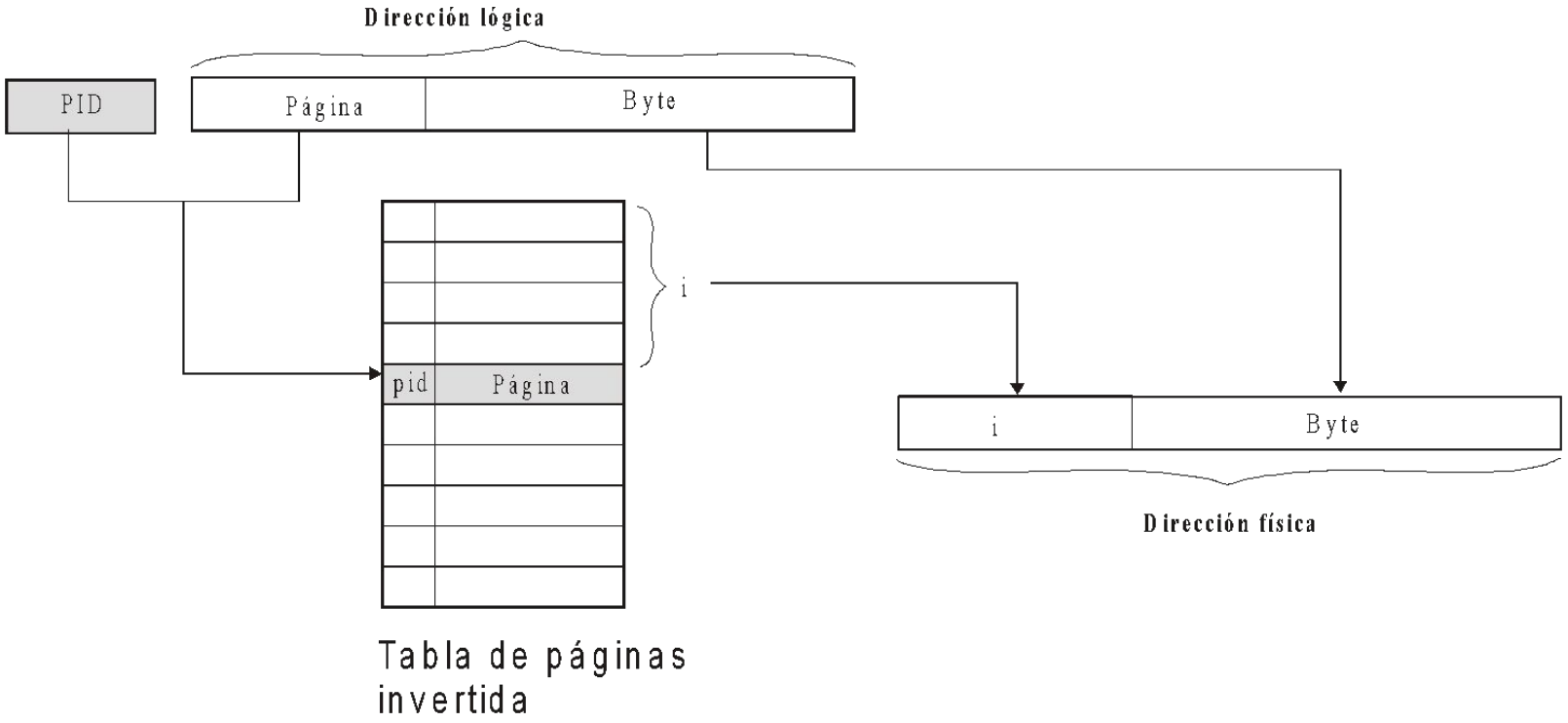
Ventajas de tablas multinivel

- Si proceso usa una parte pequeña de su espacio lógico
  - Ahorro en espacio para almacenar TPs
- Ejemplo: Proceso que usa 12MB superiores y 4MB inferiores
  - 2 niveles, páginas de 4K, dir. lógica 32 bits (10 bits por nivel) y 4 bytes por entrada
    - Tamaño: 1 TP N_1 + 4 TP N_2= 5 * 4KB = 20KB (frente a 4MB)
- Ventajas adicionales:
  - Permite compartir TPs intermedias
  - Sólo se requiere que esté en memoria la TP de nivel superior. TPs restantes en disco y traerse a demanda

TP invertida

- Procesadores actuales espacio lógico enorme (dirs. de 64b)
  - TPs muy grandes incluso usando multinivel
- Posible solución alternativa: Uso de TPs invertidas
  - Una entrada por cada marco indica la página almacenada en él
    - Tamaño de TP proporcional a memoria principal
  - Es necesario guardar núm. de página e id. de proceso
- Procedimiento de traducción:
  - MMU usa TLB convencional
  - Si fallo en TLB –› MMU busca traducción en TP invertida
- Para evitar búsqueda secuencial en TP invertida:
  - Se organiza como una tabla *hash*
- Difícil implementar compartición
- Hay que tener en cuenta que aunque TP pequeña, SO debe guardar info. de páginas no residentes

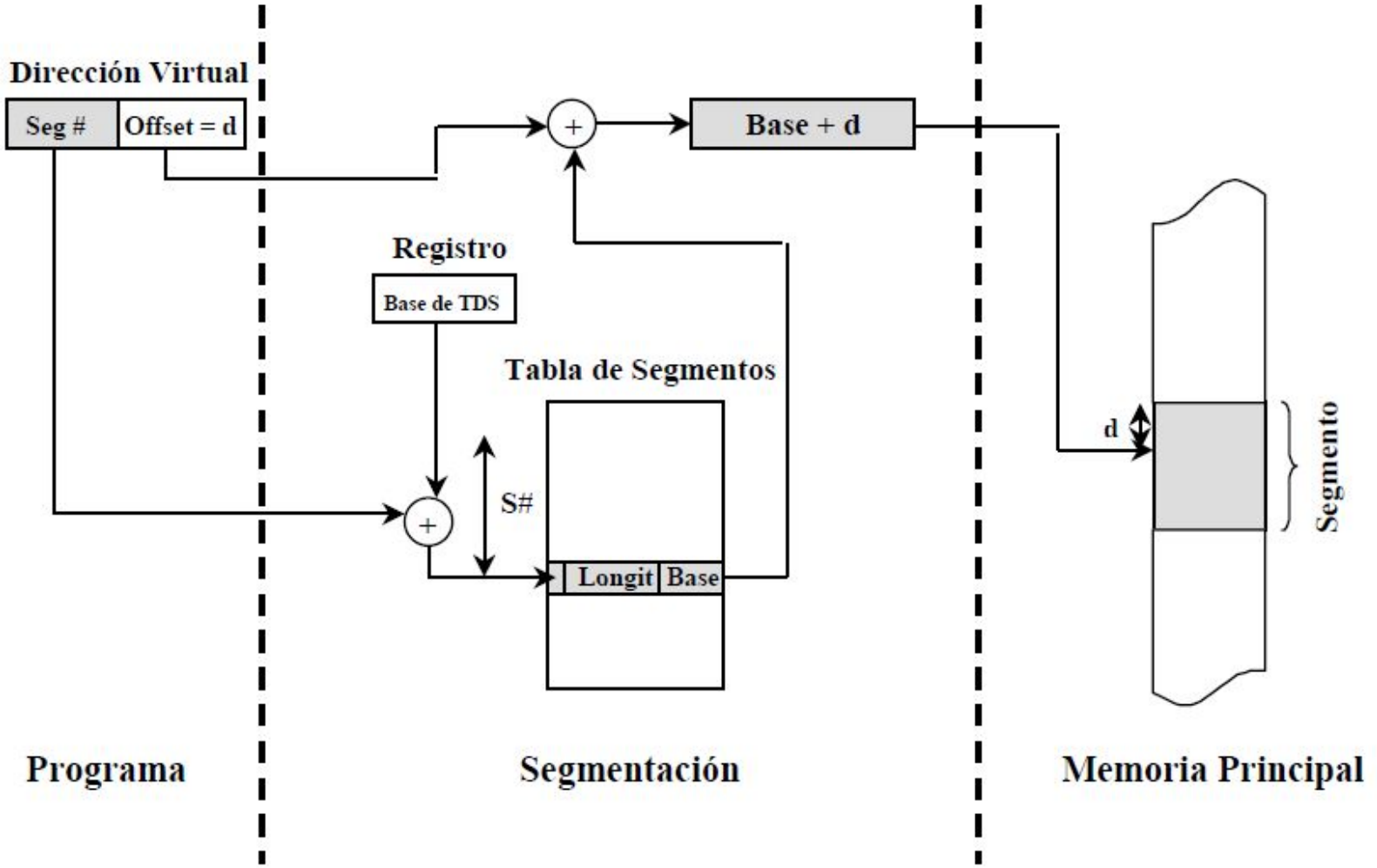
Esquema de traducción con TP invertida



Segmentación

- Esquema HW que intenta dar soporte directo a las regiones
- Generalización de reg base y límite: 1 pareja por cada segmento
- Dirección lógica: núm. de segmento + dirección en el segmento
- MMU usa una tabla de segmentos (TS)
- S.O. mantiene una TS por proceso
  - en c.contexto notifica a MMU cuál debe usar
- Entrada de TS contiene (entre otros):
  - r. base y límite del segmento
  - protección: RWX
- Fragmentación externa: segmento es la unidad de asignación
- S.O. mantiene información sobre estado de la memoria:
  - Estructuras de datos que identifiquen huecos y zonas asignadas

Esquema de traducción con segmentación



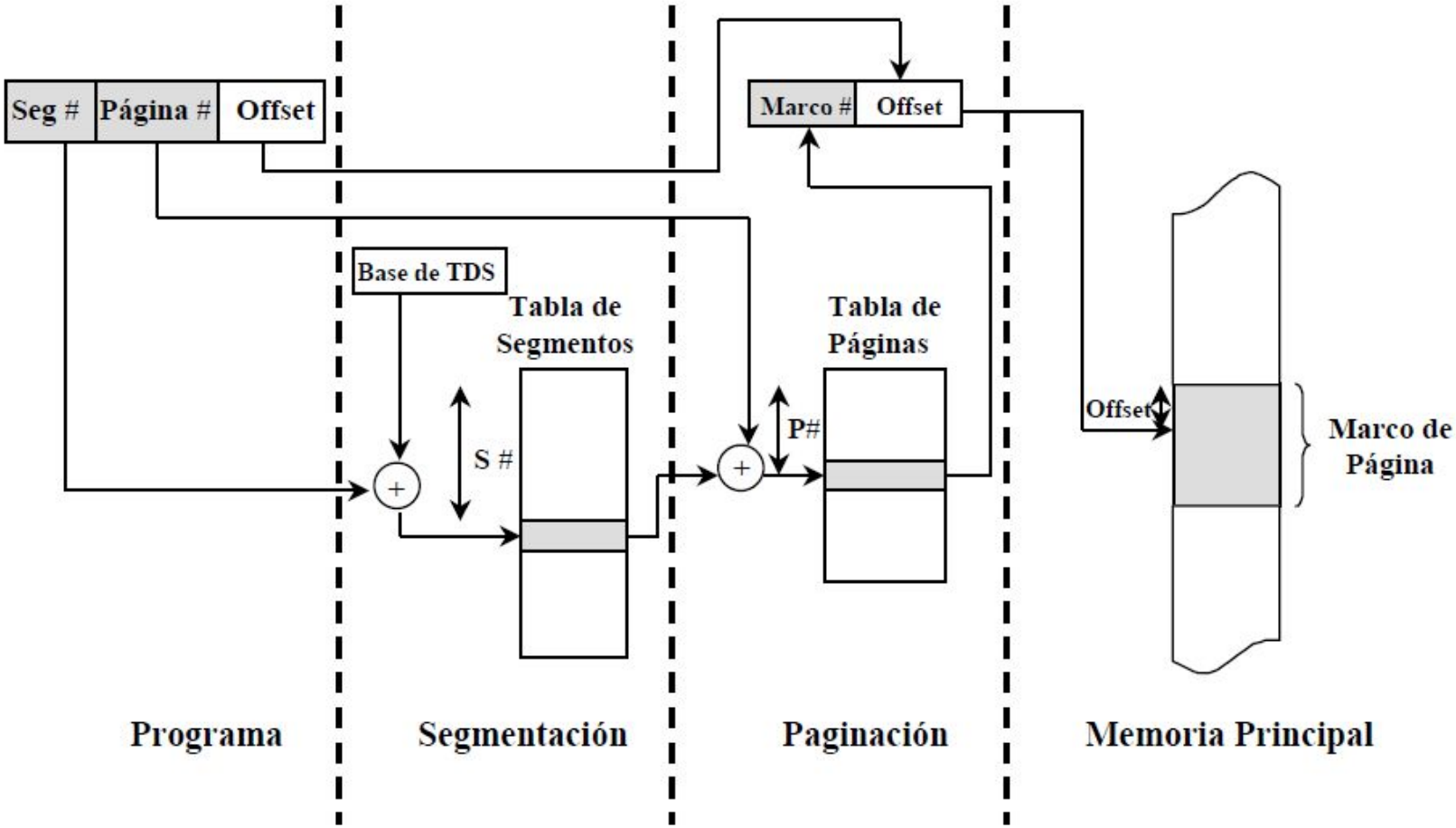
Valoración de la segmentación

- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - mediante TS
  - Protección:
    - mediante TS
  - Compartir memoria:
    - compartir segmentos: misma entrada en diferentes TS
  - Soporte de regiones:
    - bits de protección
  - Maximizar rendimiento y mapas grandes
    - presenta fragmentación externa
    - no facilita esquemas de memoria virtual debido a tamaño variable de segmentos, no es fácil la carga/descarga de segmentos
- Esquema apenas usado

Segmentación Paginada

- Entrada en TS apunta a una TP para el segmento
- "Lo mejor de los dos mundos"
- Segmentación:
  - Soporte directo de segmentos
  - Facilita operaciones sobre regiones:
    - Establecer protección –› Modificar sólo una entrada de TS
    - Definir compartición de segmento –› entradas de TS apuntando a la misma TP de segmento
- Paginación:
  - Asignación no contigua de segmento
  - Fragmentación interna

Esquema de traducción con seg. paginada



Valoración de la seg. paginada

- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - mediante TS
  - Protección:
    - mediante TS
  - Compartir memoria:
    - compartir segmentos: misma entrada en diferentes TS
  - Soporte de regiones:
    - bits de protección
  - Maximizar rendimiento y mapas grandes
    - permite esquemas de memoria virtual
- Frente a paginación:
  - Facilita al SO gestión de regiones pero requiere HW más complejo

Tratamiento del fallo de página

- Tratamiento de excepción (dirección de fallo en reg.)
  - Si dirección inválida –› Aborta proceso o le manda señal
  - Consulta T. marcos, si no hay ningún marco libre
    - Selección de víctima: pág P marco M
      - Marca P como inválida
    - Si P modificada (bit *Mod* de P activo)
      - Inicia escritura P en mem. secundaria
  - Hay marco libre (se ha liberado o lo había previamente):
    - Inicia lectura de página en marco M
    - Marca entrada de página válida referenciando a M
    - Pone M como ocupado en T. marcos (si no lo estaba)
- Fallo de página puede implicar 2 operaciones en disco

Políticas de administración de MV

- Política de reemplazo:
  - ¿Qué página reemplazar si fallo y no hay marco libre?
  - Reemplazo local
    - Sólo puede usarse para reemplazo un marco asignado al proceso que causa fallo
  - Reemplazo global
    - Puede usarse para reemplazo cualquier marco
- Política de asignación de espacio a los procesos:
  - ¿Cómo se reparten los marcos entre los procesos?
  - Asignación fija o dinámica

Algoritmos de reemplazo

- Objetivo: Minimizar la tasa de fallos de página.
- Cada algoritmo descrito tiene versión local y global:
  - local: criterio se aplica a las páginas residentes del proceso
  - global: criterio se aplica a todas las páginas residentes
- Algoritmos presentados
  - Óptimo
  - FIFO
  - Reloj (o segunda oportunidad)
  - LRU
- Uso de técnicas de *buffering* de páginas

Algoritmo óptimo

- Criterio: se conoce la página residente que tardará más en accederse –› Irrealizable
- Interés para estudios analíticos comparativos
- Ejemplo:
  - Memoria física de 4 marcos de página
  - Referencias *a b g a d e a b a d e g d e*

Ref	**a**	**b**	**g**	**a**	**d**	**e**	**a**	**b**	**a**	**d**	**e**	**g**	**d**	**e**
0	a	a	a	a	a	a	a	a	a	a	a	g	g	g
1		b	b	b	b	b	b	b	b	b	b	b	b	b
2			g	g	g	e	e	e	e	e	e	e	e	e
3					d	d	d	d	d	d	d	d	d	d

6 fallos

Algoritmo FIFO

- Criterio: página que lleva más tiempo residente
- Fácil implementación:
  - páginas residentes en orden FIFO –› se expulsa la primera
  - no requiere el bit de página accedida (*Ref*)
- No es una buena estrategia:
  - Página que lleva mucho tiempo residente puede seguir accediéndose frecuentemente
  - Su criterio no se basa en el uso de la página
- Anomalía de Belady
  - Se pueden encontrar ejemplos que al aumentar el número de marcos aumenta el número de fallos de página

FIFO

- Ejemplo:
  - Memoria física de 4 marcos de página
  - Referencias *a b g a d e a b a d e g d e*

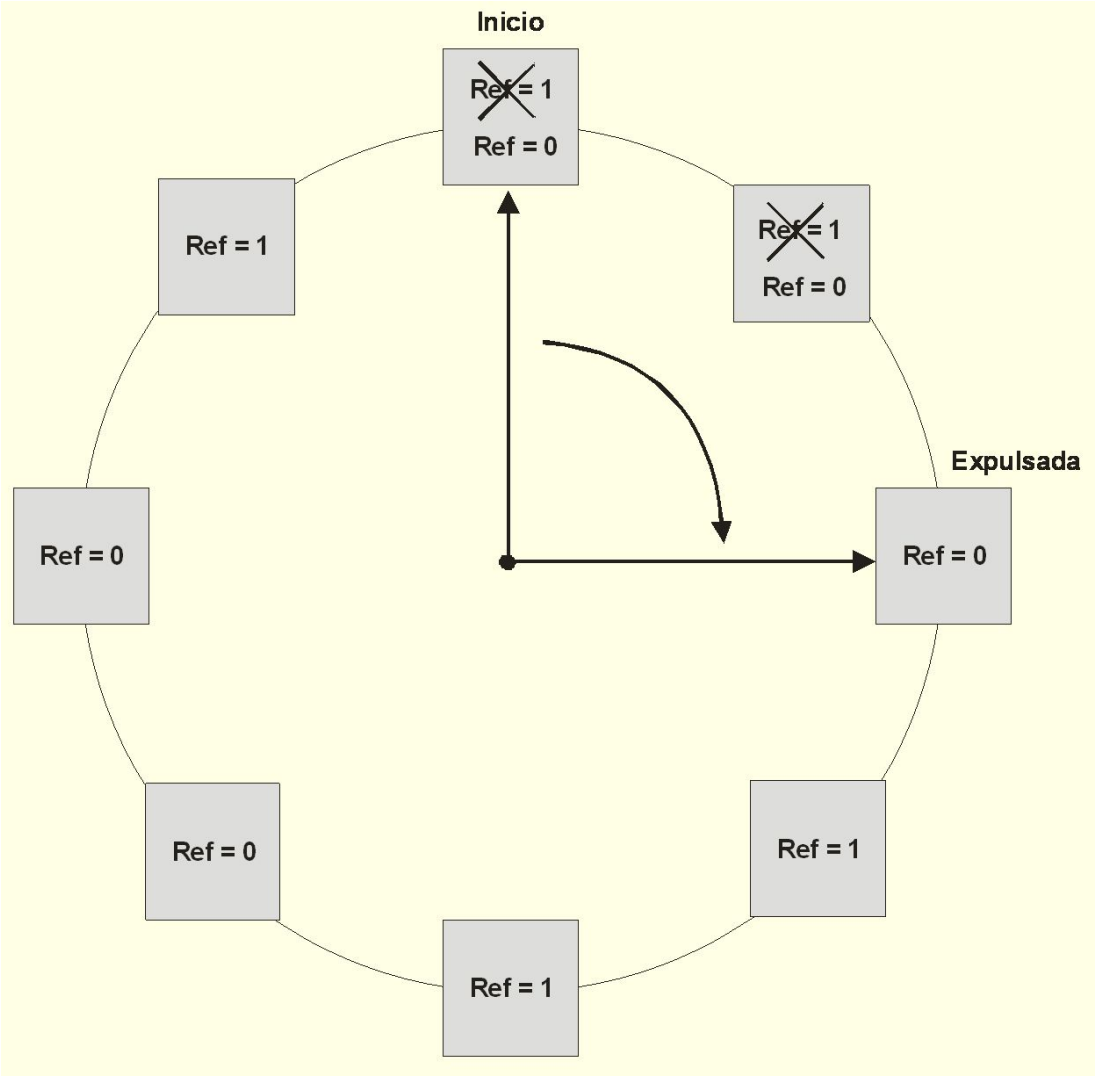
Ref	**a**	**b**	**g**	**a**	**d**	**e**	**a**	**b**	**a**	**d**	**e**	**g**	**d**	**e**
0	a_1	a	a	a	a	e_6	e	e	e	e	e	e	d_{13}	d
1		b_2	b	b	b	b	a_7	a	a	a	a	a	a	e_{14}
2			g_3	g	g	g	g	b_8	b	b	b	b	b	b
3					d_5	d	d	d	d	d	d	g_{12}	g	g

10 fallos

Algoritmo del reloj (o 2ª oportunidad)

- FIFO + uso de bit de referencia *Ref* (de página accedida)
- Criterio:
  - Si página elegida por FIFO no tiene activo *Ref*
    - Es la página expulsada
  - Si lo tiene activo (2ª oportunidad)
    - Se desactiva *Ref*
    - Se pone página al final de FIFO
    - Se aplica criterio a la siguiente página
- Se puede implementar orden FIFO como lista circular con una referencia a la primera página de la lista:
  - Se visualiza como un reloj donde la referencia a la primera página es la aguja del reloj

Algoritmo del reloj (o 2ª oportunidad)



Reloj

- Ejemplo:
  - Memoria física de 4 marcos de página
  - Referencias *a b g a d e a b a d e g d e*

Ref	**a**	**b**	**g**	**a**	**d**	**e**		**a**	**b**	**a**	**d**	**e**	**g**	**d**	**e**
0	↓a_0	↓a_0	↓a_0	↓a_1	↓a_1	a_0	a_0	a_1	a_1	a_1	a_1	a_1	a_0	a_0	a_0
1		b_0	b_0	b_0	b_0	↓b_0	e_0	e_0	e_0	e_0	e_0	e_1	e_0	e_0	e_1
2			g_0	g_0	g_0	g_0	↓g_0	↓g_0	b_0	b_0	b_0	b_0	g_0	g_0	g_0
3					d_0	d_0	d_0	d_0	↓d_0	↓d_0	↓d_1	↓d_1	↓d_0	↓d_1	↓d_1

7 fallos

Algoritmo LRU

- Criterio: página residente menos recientemente usada
- Por proximidad de referencias:
  - Pasado reciente condiciona futuro próximo
- Sutileza:
  - En su versión global: menos recientemente usada en el tiempo lógico de cada proceso
- Difícil implementación estricta (hay aproximaciones):
  - Precisaría una MMU específica
- Posible implementación con HW específico:
  - En entrada de TP hay un contador
  - En cada acceso a memoria MMU copia contador del sistema a entrada referenciada
  - Reemplazo: Página con contador más bajo

Algoritmo LRU

- Ejemplo:
  - Memoria física de 4 marcos de página
  - Referencias *a b g a d e a b a d e g d e*

Ref	**a**	**b**	**g**	**a**	**d**	**e**	**a**	**b**	**a**	**d**	**e**	**g**	**d**	**e**
0	a_1	a	a	a_4	a	a	a_7	a	a_9	a	a	a	a	a
1		b_2	b	b	b	e_6	e	e	e	e	e_{11}	e	e	e_{14}
2			g_3	g	g	g	g	b_8	b	b	b	g_{12}	g	g
3					d_5	d	d	d	d	d_{10}	d	d	d_{13}	d

7 fallos

Buffering de páginas

- Peor caso en tratamiento de fallo de página:
  - 2 accesos a dispositivo
- Alternativa: mantener una reserva de marcos libres
- Fallo de página: siempre usa marco libre (no reemplazo)
- Si número de marcos libres < umbral
  - "demonio de paginación" aplica repetidamente el algoritmo de reemplazo:
    - páginas no modificadas pasan a lista de marcos libres
    - páginas modificadas pasan a lista de marcos modificados
      - cuando se escriban a disco pasan a lista de libres
      - pueden escribirse en tandas (mejor rendimiento)
- Si se referencia una página mientras está en estas listas:
  - fallo de página la recupera directamente de la lista (no E/S)
  - puede arreglar el comportamiento de algoritmos "malos"

Retención de páginas en memoria

- Páginas marcadas como no reemplazables
- Se aplica a páginas del propio SO
  - SO con páginas fijas en memoria es más sencillo
- También se aplica mientras se hace DMA sobre una página
- Algunos sistemas ofrecen a aplicaciones un servicio para fijar en memoria una o más páginas de su mapa
  - Adecuado para procesos de tiempo real
  - Puede afectar al rendimiento del sistema
  - En POSIX servicio `mlock`

Estrategia de asignación fija

- Número de marcos asignados al proceso (conjunto residente) es constante
- Puede depender de características del proceso:
  - tamaño, prioridad,...
- No se adapta a las distintas fases del programa
- Comportamiento relativamente predecible
- Sólo tiene sentido usar reemplazo local
- Arquitectura impone nº mínimo:
  - Por ejemplo: instrucción MOVE /DIR1, /DIR2 requiere un mínimo de 3 marcos en memoria:
    - Instrucción y dos operandos deben estar residentes para ejecutar la instrucción

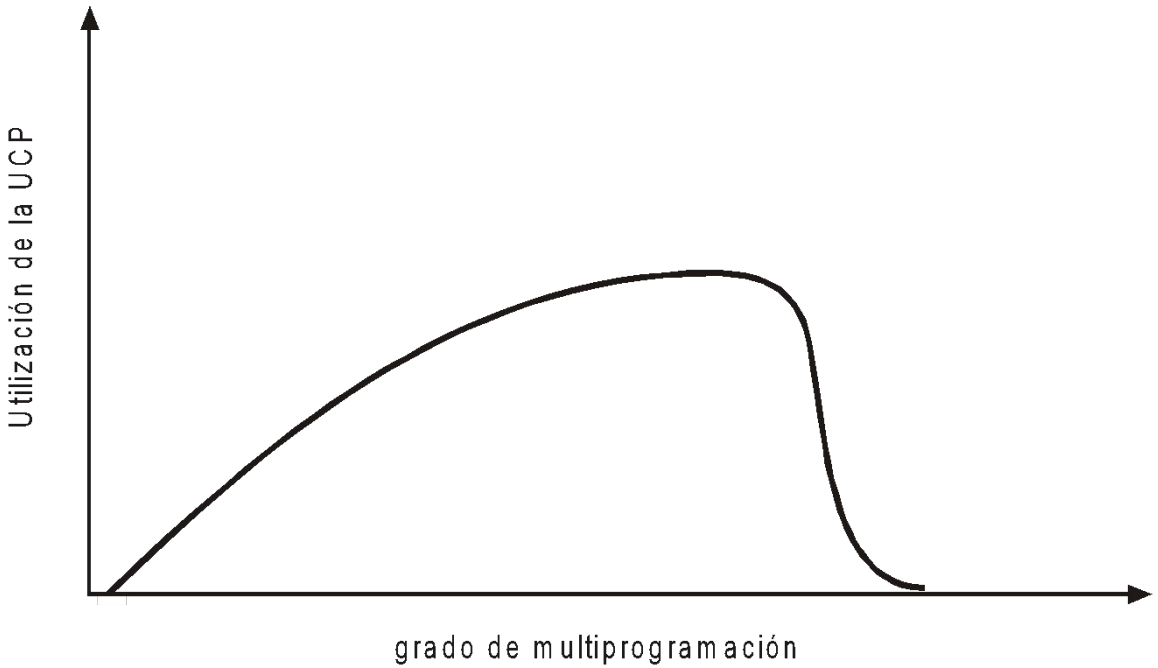
Estrategia de asignación dinámica

- Número de marcos varía dependiendo de comportamiento del proceso (y posiblemente de los otros procesos)
- Asignación dinámica + reemplazo local
  - Proceso va aumentando o disminuyendo su conjunto residente dependiendo de su comportamiento
  - Comportamiento relativamente predecible
- Asignación dinámica + reemplazo global
  - Procesos se quitan las páginas entre ellos
  - Comportamiento difícilmente predecible

Hiperpaginación (*Thrashing*)

- Tasa excesiva de fallos de página de un proceso o en el sistema
- Con asignación fija: Hiperpaginación en P_i
  - Si conjunto residente de P_i < conjunto de trabajo P_i
- Con asignación variable: Hiperpaginación en el sistema
  - Si nº marcos disponibles < Σ conjuntos de trabajo de todos
  - Grado de utilización de UCP cae drásticamente
  - Procesos están casi siempre en colas de dispositivo de paginación
  - Solución (similar al *swapping*): Control de carga
    - Disminuir el grado de multiprogramación
    - Suspender 1 o más procesos liberando sus páginas residentes
  - Problema: ¿Cómo detectar esta situación?

Hiperpaginación en el sistema

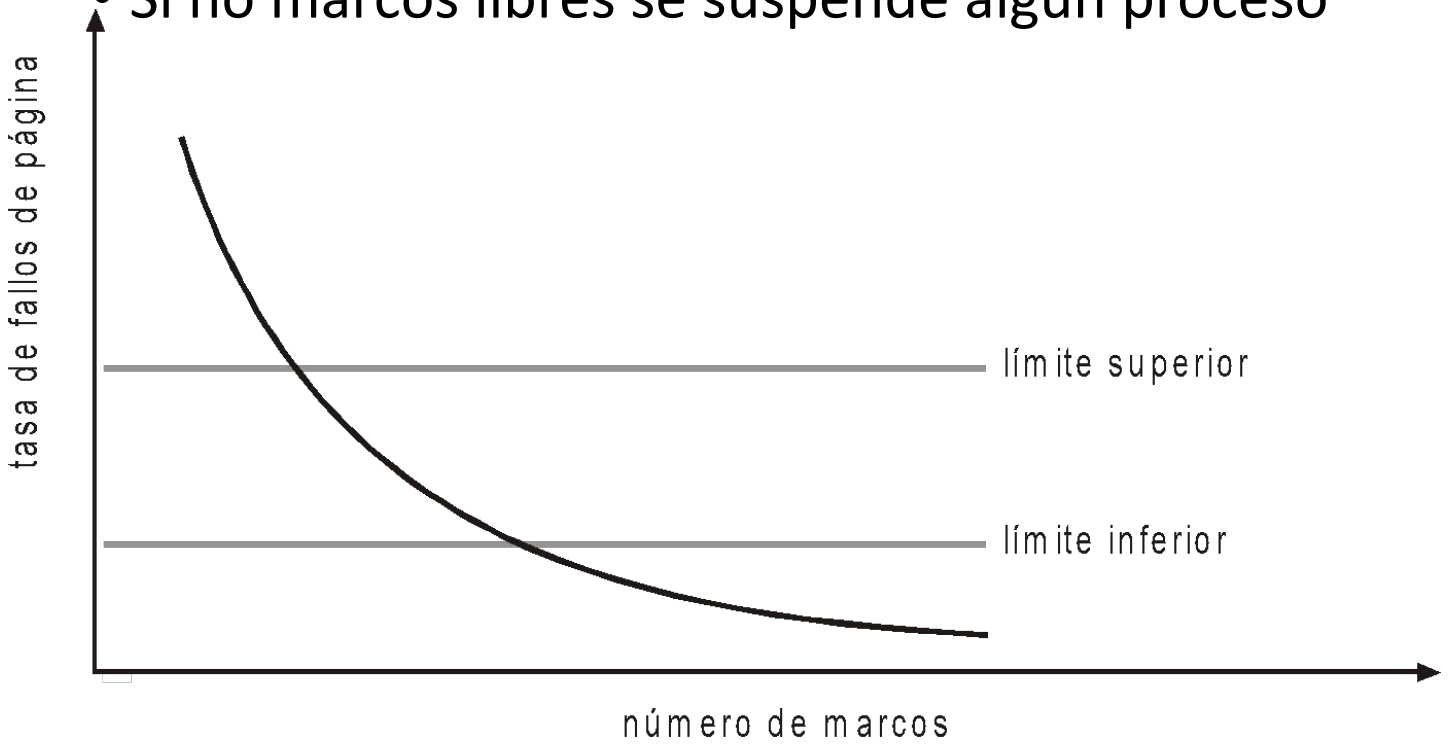


Estrategia del conjunto de trabajo

- Intentar conocer el conjunto de trabajo de cada proceso
  - Páginas usadas por el proceso en las últimas N referencias
- Si conjunto de trabajo decrece se liberan marcos
- Si conjunto de trabajo crece se asignan nuevos marcos
  - Si no hay disponibles: suspender proceso(s)
  - Se reactivan cuando hay marcos suficientes para c. de trabajo
- Asignación dinámica con reemplazo local
- Difícil implementación estricta
  - Precisaría una MMU específica
- Se pueden implementar aproximaciones:
  - Estrategia basada en frecuencia de fallos de página (PFF):
    - Controlar tasa de fallos de página de cada proceso

Estrategia basada en frecuencia de fallos

- Si tasa < límite inferior se liberan marcos aplicando un algoritmo de reemplazo
- Si tasa > límite superior se asignan nuevos marcos
  - Si no marcos libres se suspende algún proceso



Control de carga y reemplazo global

- Algoritmos de reemplazo global no controlan hiperpaginación
  - ¡Incluso el óptimo!
  - Necesitan cooperar con un algoritmo de control de carga
- Ejemplo: UNIX 4.3 BSD
  - Reemplazo global con algoritmo del reloj
    - Variante con dos “manecillas”
  - Uso de *buffering* de páginas
    - “demonio de paginación” controla nº de marcos libres
  - Si número de marcos libres < umbral
    - “demonio de paginación” aplica reemplazo
  - Si se repite con frecuencia la falta de marcos libres:
    - Proceso “swapper” suspende procesos

Contenido

- Objetivos del sistema de gestión de memoria
- Modelo de memoria de un proceso
- Asignación contigua
  - Intercambio (*swapping*)
- Memoria virtual
  - Políticas de intercambio
- Gestión de memoria en Linux

Gestión del espacio de usuario en Linux

- En la mayoría de los sistemas operativos hay dos esquemas de gestión de memoria:
  - Sistema de paginación para la gestión de memoria virtual de los procesos.
    - En una máquina de 32 bits cada proceso tiene 3GB para direcciones virtuales de usuario y 1GB para las de núcleo
    - El número de niveles de la tabla de páginas depende de la arquitectura.
    - Asigna un conjunto de páginas contiguas usando el algoritmo buddy.
    - Algoritmo de reemplazamiento: modificación del algoritmo del reloj con dos manecillas.
    - Si es NUMA se asigna memoria del nodo asociado al procesador que ejecuta el proceso.
    - Gestión del heap: la realiza la biblioteca del lenguaje de programación con soporte del sistema operativo.

Gestión del heap de C en Linux

- La biblioteca de gestión del heap solicita una nueva página al SO cuando necesita más memoria
  - Llamada al sistema brk
- Normalmente no la devuelve cuando le sobra.
- La asignación debe ser muy eficiente porque hay un GRAN NUMERO de peticiones:
  - Algoritmo de múltiples listas con huecos de tamaño variable “buddy perezoso”
    - No se juntan huecos adyacentes.
  - Para sistemas multithread hay versiones con varios heaps
    - Para que el acceso a las estructuras de datos que mantienen el estado del heap no sea un cuello de botella.

Gestión del espacio de kernel en Linux

- En la mayoría de los sistemas operativos hay dos esquemas de gestión de memoria:
  - Sistema de gestión de áreas de memoria del kernel (slab allocator).
    - Para las estructuras de datos y buffers que crea el kernel.
    - Pide un grupo de marcos páginas contiguas de memoria al sistema de paginación y gestiona el espacio.
    - Asignador basado en objetos: tiene caches de objetos usados con frecuencia (el kernel crea y destruye objetos del mismo tipos, así evita destruirlos e inicializarlos)