Grupo ARCOS
Universidad Carlos III de Madrid

Lección 5 (b)
La gestión de memoria

Diseño de Sistemas Operativos
Grado en Ingeniería Informática

Objetivos

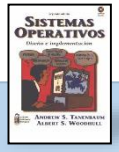
- Conocimiento de los métodos de gestión interna de recursos en un sistema operativo.
- Usar herramientas de monitorización, gestión y ajuste de sistemas operativos.

Lecturas recomendadas

Base

1. Carretero 2007:
   1. Cap.5

Recomendada



1. Tanenbaum 2006(en):
   1. Cap.4
2. Stallings 2005:
   1. Parte tres
3. Silberschatz 2006:
   1. Cap. 4

A recordar…

1. Estudiar la teoría asociada.
   - Estudiar el material asociado a la bibliografía: las transparencias solo no son suficiente.
2. Repasar lo visto en clase.
   - Realizar el cuaderno de prácticas progresivamente.
3. Ejercitar las competencias.
   - Realizar las prácticas progresivamente.
   - Realizar todos los ejercicios posibles.

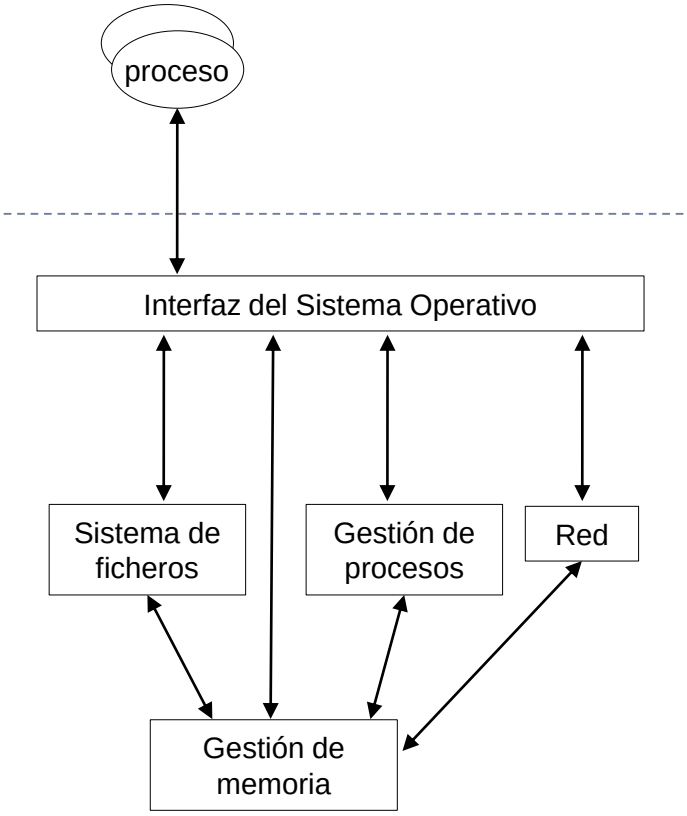
Contenidos

1. Introducción
2. Mecanismos para la gestión de la memoria
3. Políticas y directrices de gestión

Contenidos

1. **Introducción**
2. Mecanismos para la gestión de la memoria
3. Políticas y directrices de gestión

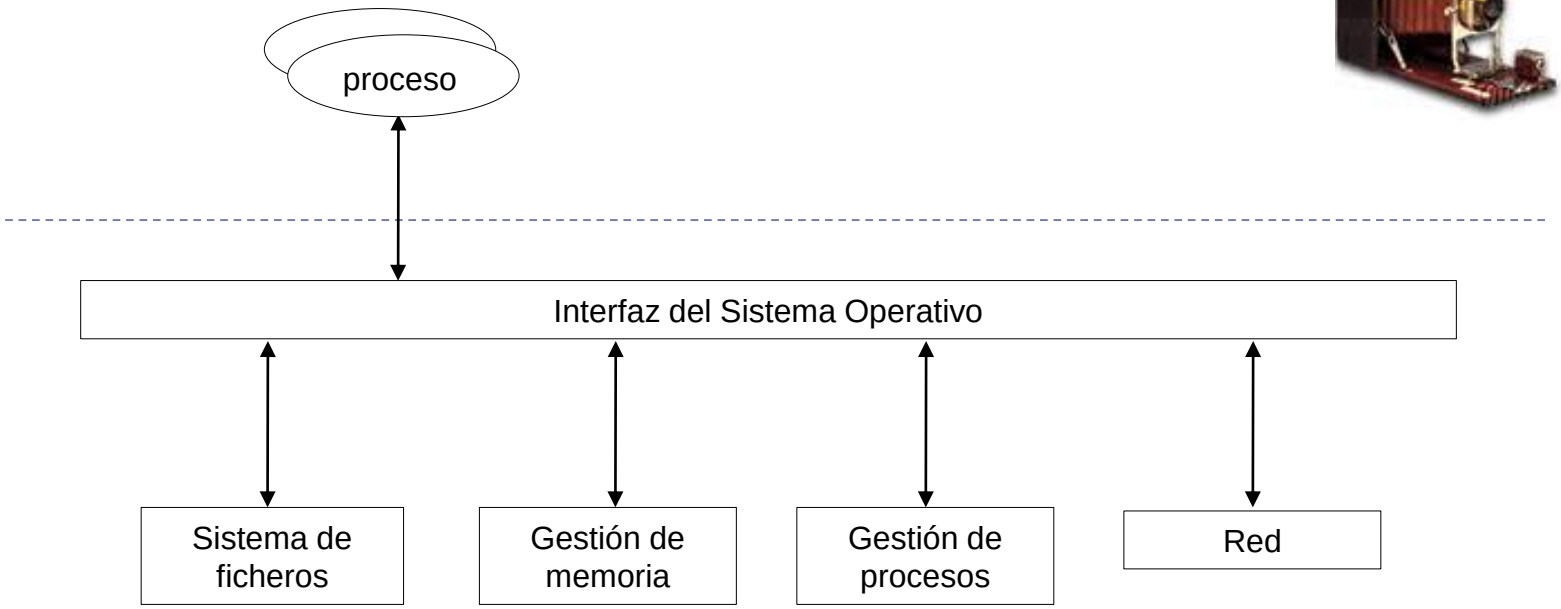
Ámbito de la gestión de memoria



- Encargado de la gestión de la memoria entre procesos y el *kernel*
- El resto del sistema operativo es su mejor cliente:
  - Gestión de procesos
  - Gestión de ficheros
- Pero es un reflejo de las necesidades de los procesos

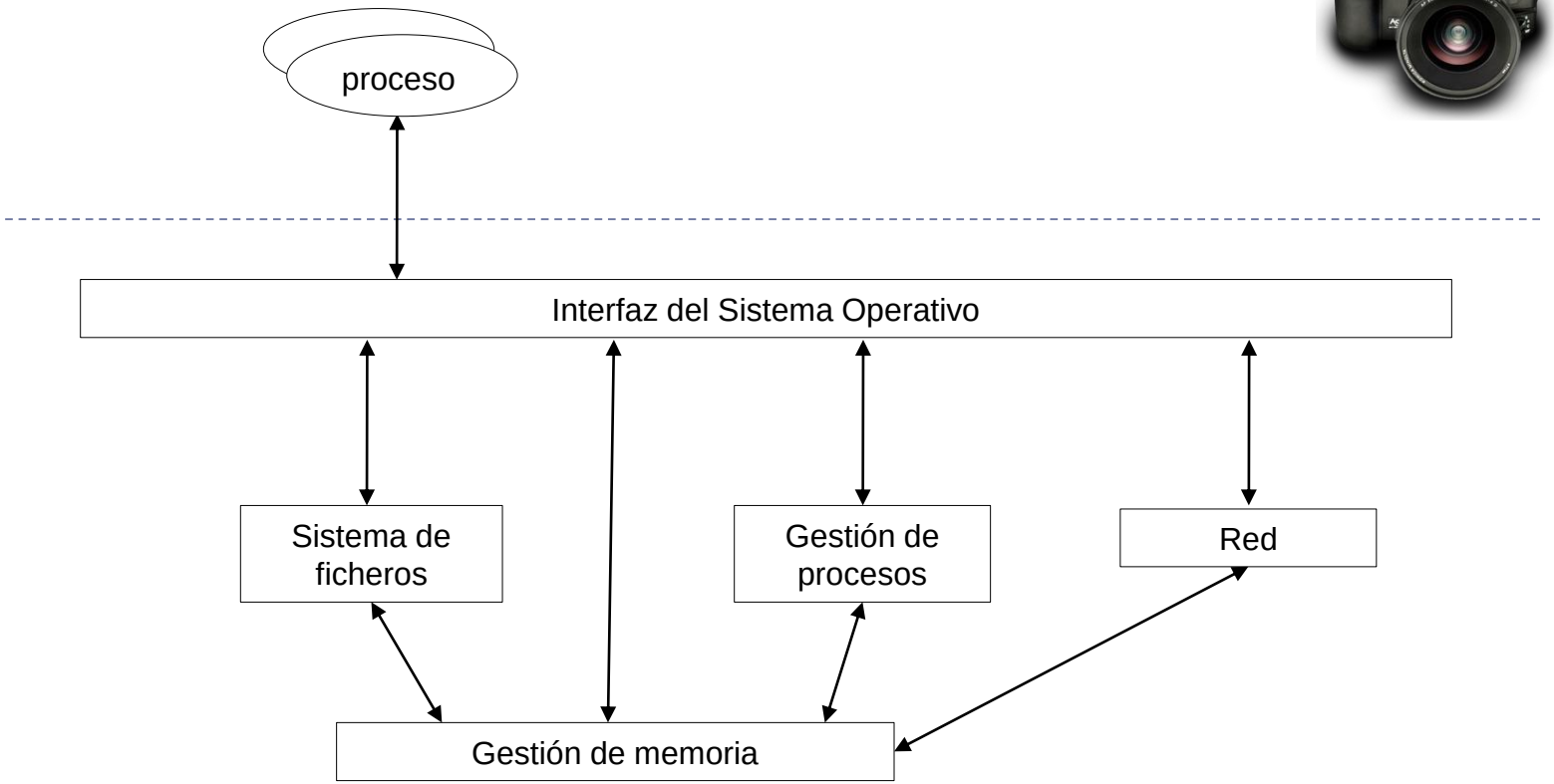
Arquitectura de la gestión de memoria

antigua arquitectura

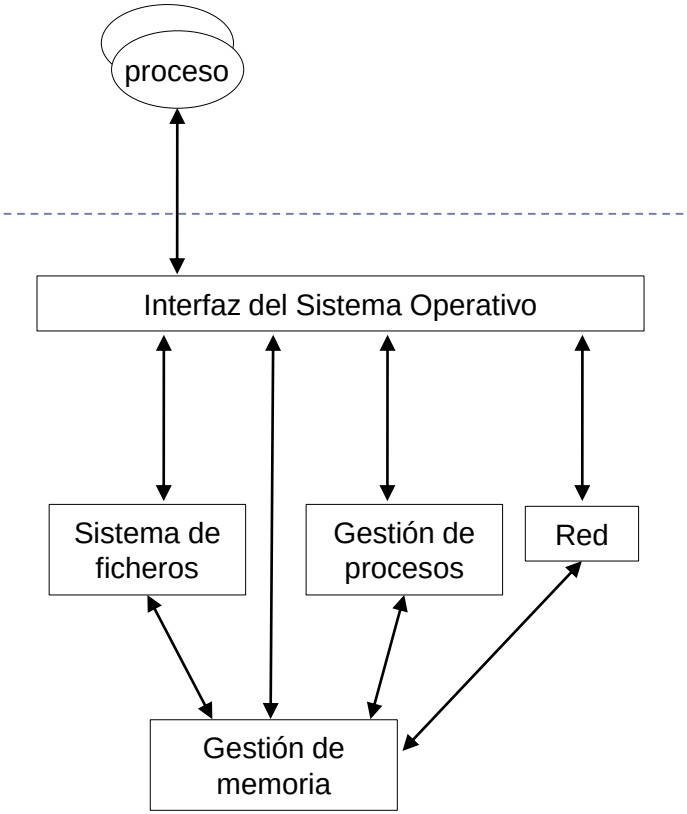


Arquitectura de la gestión de memoria

nueva arquitectura



Objetivos generales de la memoria

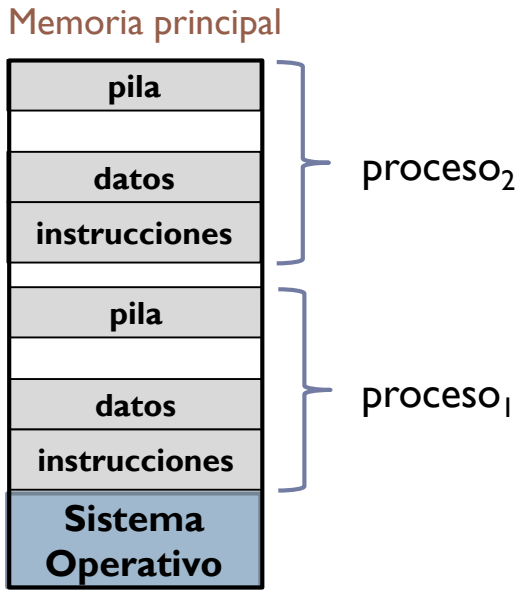


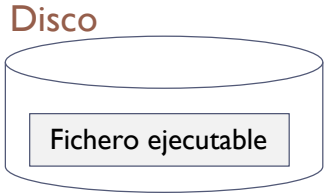
1. Localización de referencias a memoria
   - ha de traducir las referencias a memoria a direcciones físicas
2. Protección de espacios de memoria
   - prohibir referencias entre procesos distintos
3. Compartición de espacios de memoria
   - permitir que varios procesos accedan a un espacio de memoria común
4. Organización lógica (de programas)
   - los programas se dividen en módulos independientes
5. Organización física (de la memoria)
   - rellenar la memoria con múltiples programas y módulos

http://doursat.free.fr/docs/CS446_F05/CS446_F05_3_Memory1.pdf

Objetivos generales de la memoria
1.- Localización de referencias a memoria

- El programador no tiene porque saber dónde se colocará el programa en memoria cuando se ejecute
  - Un programa puede ser ejecutado varias veces (cada una de ellas irá a una parte de memoria diferente)
- Mientras que el programa sea ejecutado también puede mandarse a disco y volver a memoria en una posición diferente
- Por tanto, las referencias lógicas (relativas) de memoria han de traducirse a direcciones físicas (absolutas)

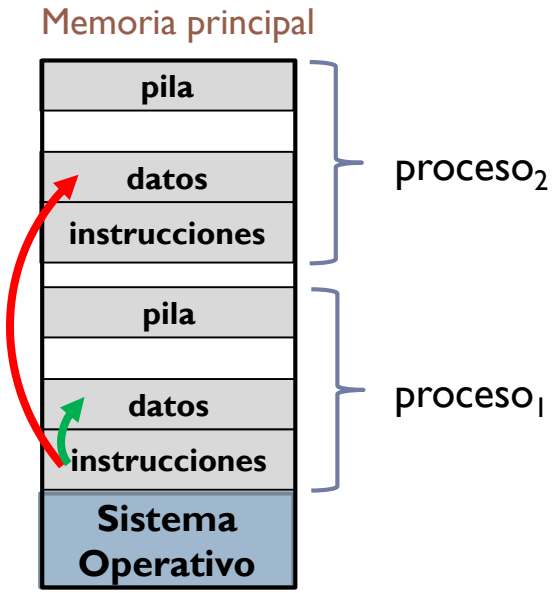




http://doursat.free.fr/docs/CS446_F05/CS446_F05_3_Memory1.pdf

Objetivos generales de la memoria
2.- Protección de espacios de memoria

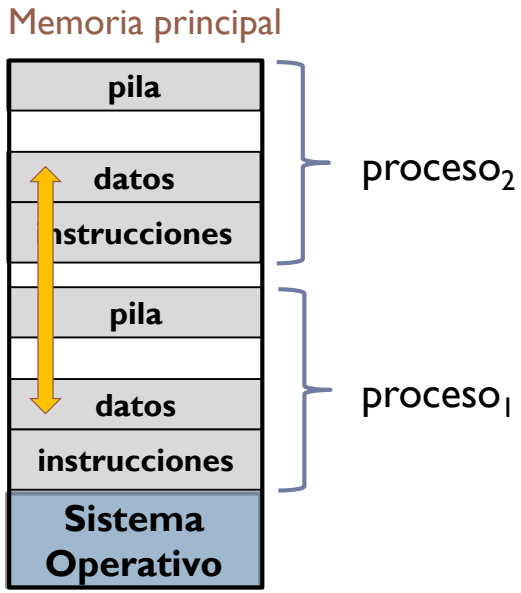
- Los procesos no han de usar posiciones de memoria de otros procesos
  - Excepción: depurador, …
- Las posiciones de memoria tendrían que ser comprobadas en tiempo de ejecución
  - No es posible comprobar los accesos a memoria física en tiempo de compilación
- Las posiciones de memoria han de comprobarse por hardware
  - El sistema operativo no puede anticipar las referencias de memoria (calculadas) que un proceso va a realizar



http://doursat.free.fr/docs/CS446_F05/CS446_F05_3_Memory1.pdf

Objetivos generales de la memoria
3.- Compartición de espacios de memoria

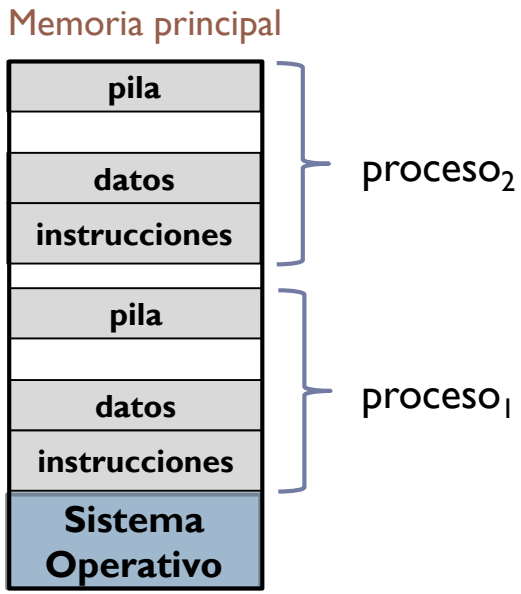
- Opuesto al punto anterior (aparentemente) debe ser posible que varios procesos puedan acceder a la misma porción de memoria:
  - Procesos ejecutando el mismo código podrían compartir la misma copia de código en memoria
  - Procesos que cooperan en la misma tarea pueden necesitar acceder a las mismas estructuras de datos
- Debe ser solicitado y concedido explícitamente
  - Depurador, etc.



http://doursat.free.fr/docs/CS446_F05/CS446_F05_3_Memory1.pdf

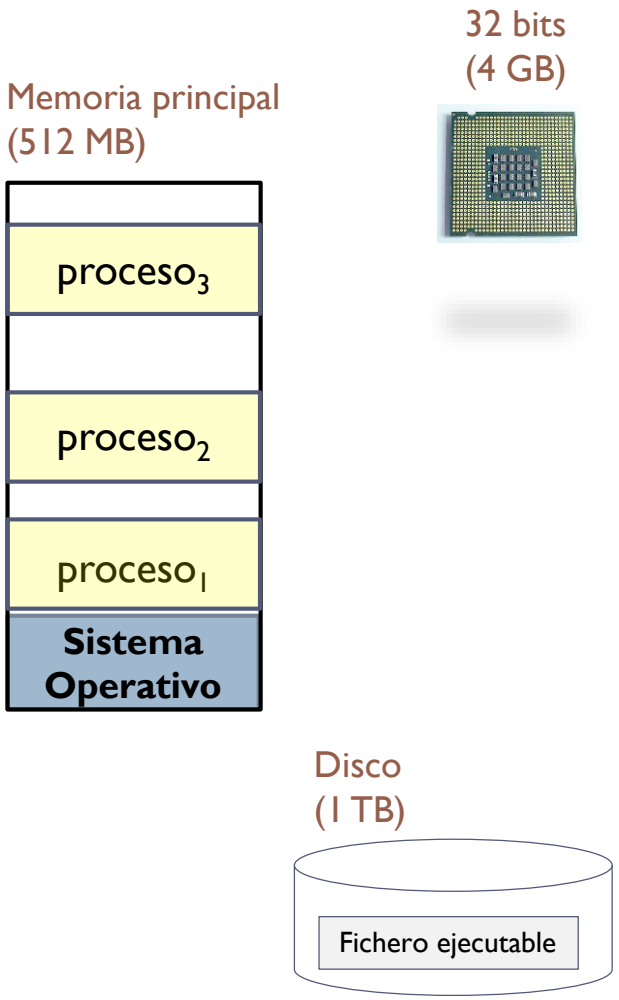
Objetivos generales de la memoria
4.- Organización lógica (de los programas)

- Los datos de un proceso no son homogéneos
  - Ej.: código, variables locales, etc.
  - Cada tipo de información tiene distintas necesidades
    - Lectura, escritura, ejecución, etc.
    - Creación estática o dinámica
- La información de un proceso (su imagen) se divide en diferentes regiones
  - Cada región se adapta a un tipo de datos concreto (código, variable dinámica, etc.)
    - Hay que gestionar las zonas sin asignar (huecos)
- Gestionar la memoria de un proceso es gestionar cada una de sus regiones



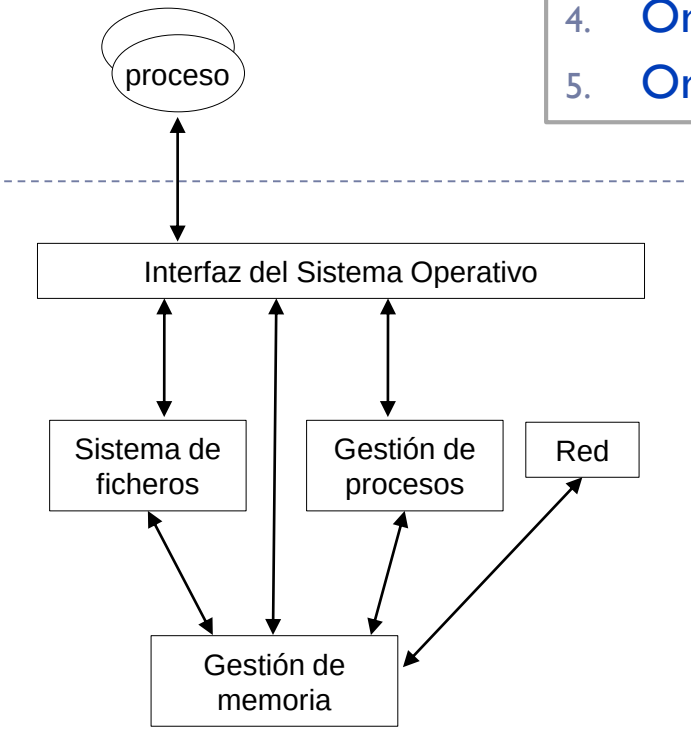
Objetivos generales de la memoria
5.- Organización física (de la memoria)

- Poder ejecutar un proceso cuando su imagen de memoria es más grande que la memoria principal:
  - Se guardan en disco las partes del proceso que no se usen en el momento
- Poder ejecutar un conjunto de procesos cuya ocupación de memoria es mayor que la memoria principal
- Evitar pérdida de memoria por fragmentación:
  - Hay memoria física libre pero está fragmentada en espacios no contiguos que el sistema de gestión no puede aprovechar



Ámbito, arquitectura y objetivos
resumen

1. Localización de referencias a memoria
2. Protección de espacios de memoria
3. Compartición de espacios de memoria
4. Organización lógica (de programas)
5. Organización física (de la memoria)

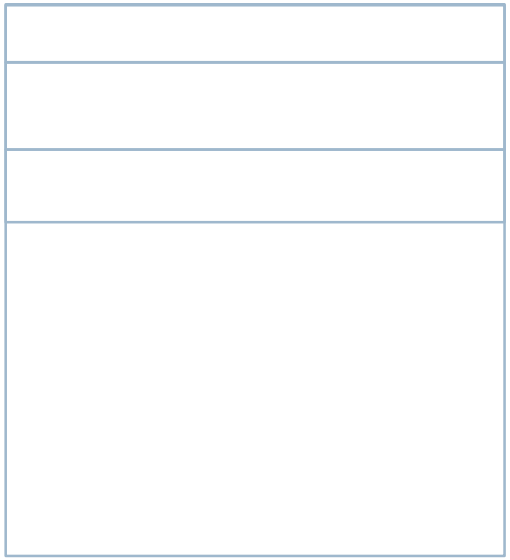


Contenidos

1. Introducción
2. **Mecanismos para la gestión de la memoria**
3. Políticas y directrices de gestión

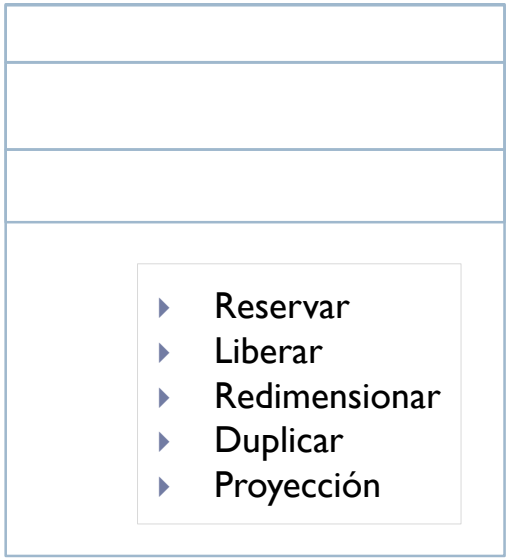
Gestor de memoria (*memory allocator*)

memory allocator = Bloque

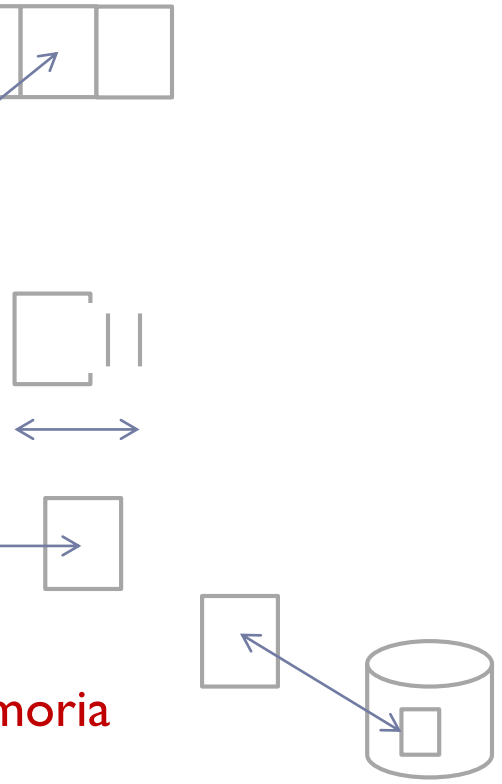


Gestor de memoria (*memory allocator*)

memory allocator = Bloque + Interfaz

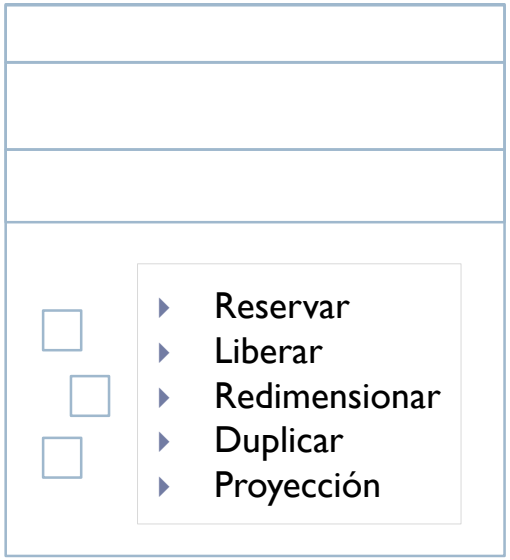


- Reservar
- Liberar
- Cambiar tamaño
- Duplicación
- Proyección en memoria

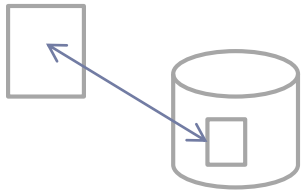


Gestor de memoria (*memory allocator*)

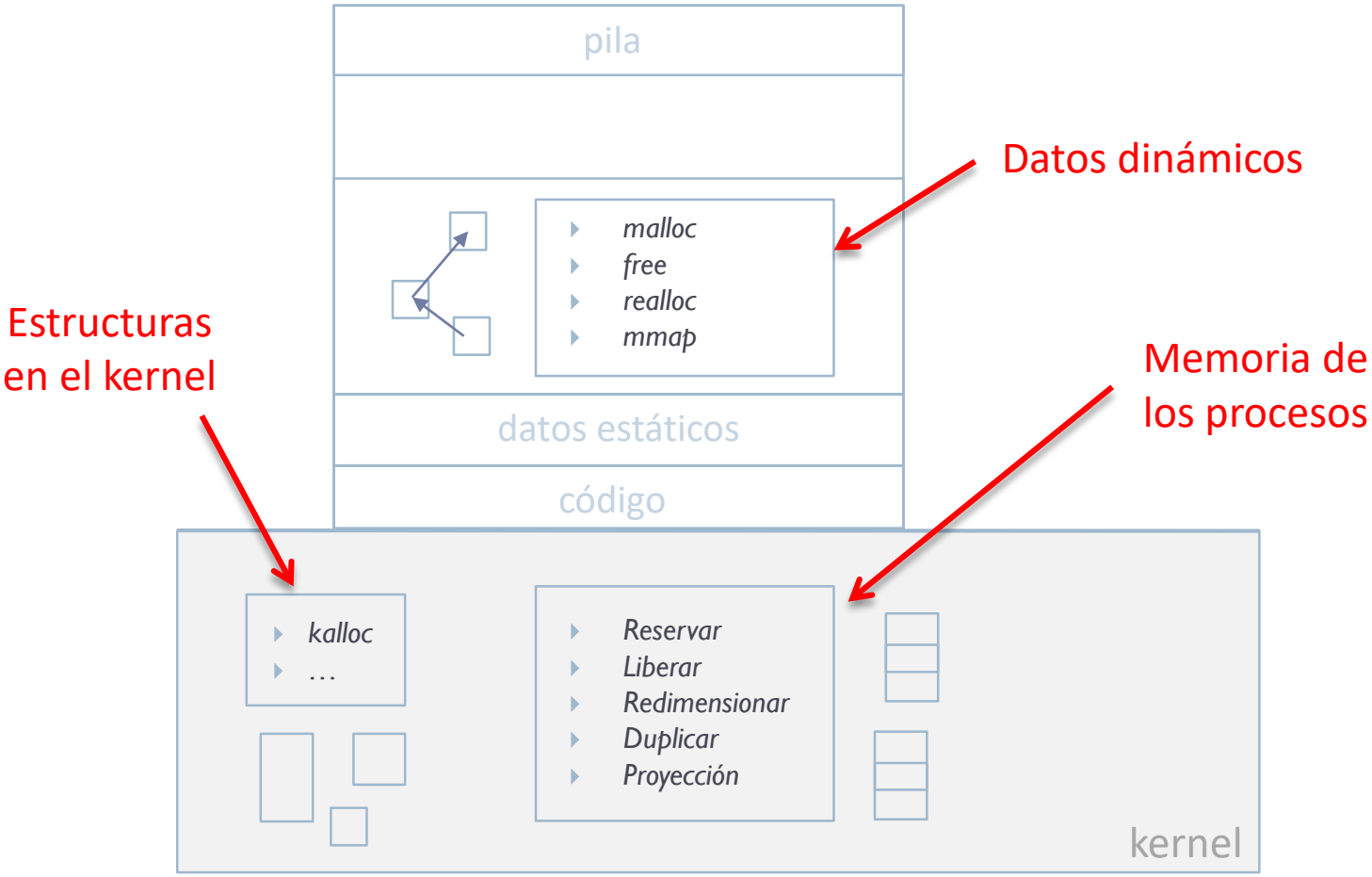
memory allocator = Bloque + Interfaz + Metadatos



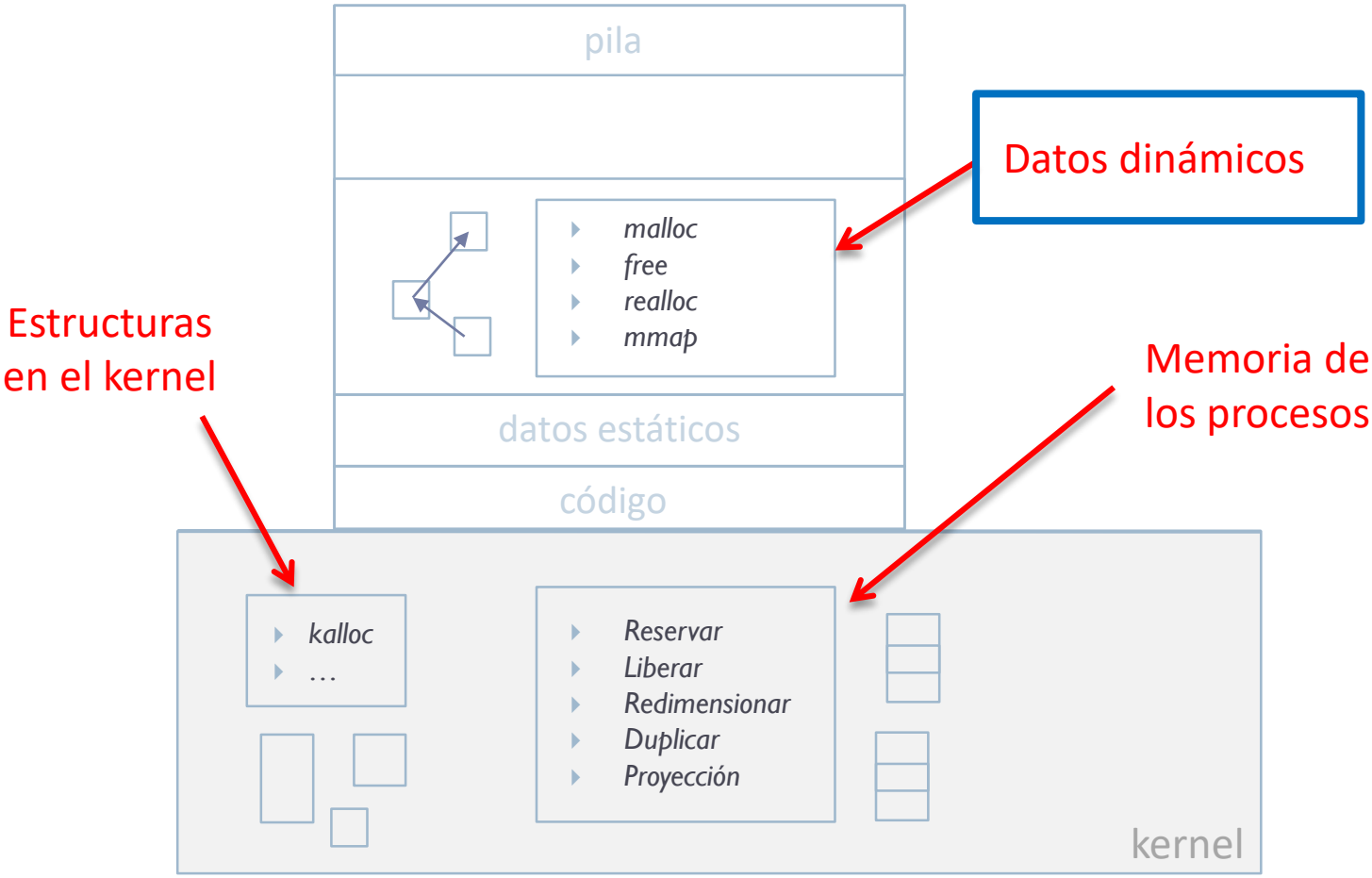
- Reservar
- Liberar
- Cambiar tamaño
- Duplicación
- Proyección en memoria



Gestores a varios niveles



Gestores a varios niveles



Gestión de memoria dinámica

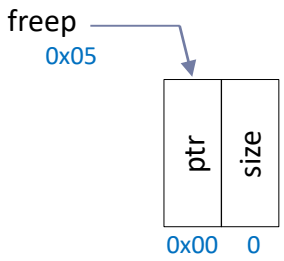
- ¿Por qué es tan 'delicado' el uso de memoria dinámica?

```
acaldero@phoenix:~/infodso/$ ./ptr
Violación de segmento

acaldero@phoenix:~/infodso/$ gdb ptr
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>...
Leyendo sÃmbolos desde /home/acaldero/work/infodso/memoria/ptr...hecho.
(gdb) run
Starting program: /home/acaldero/work/infodso/memoria/ptr

Program received signal SIGSEGV, Segmentation fault.
0xb7f79221 in ?? () from /lib/libc.so.6
```

Ejemplo de *libc storage allocator*
Header

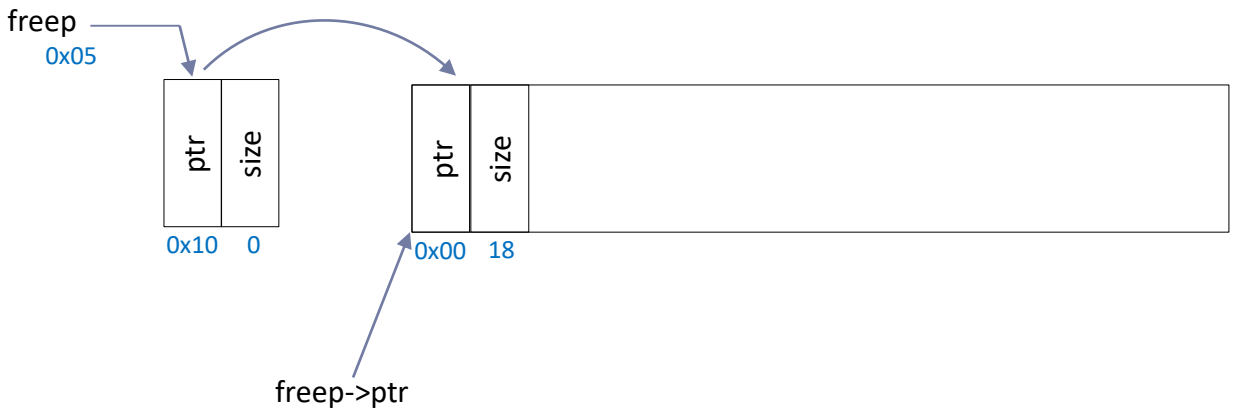


- static Header base :
  - Primer elemento de la lista
  - Con tamaño 0 (cabeceras)

```
typedef long Align;    /* for alignment to long boundary */
union header {         /* block header */
      struct {
          union header *ptr; /* next block if on free list */
          unsigned size;     /* size of this block */
      } s;
      Align x;          /* force alignment of blocks */
  };
  typedef union header Header;
```

Ejemplo de *libc storage allocator*

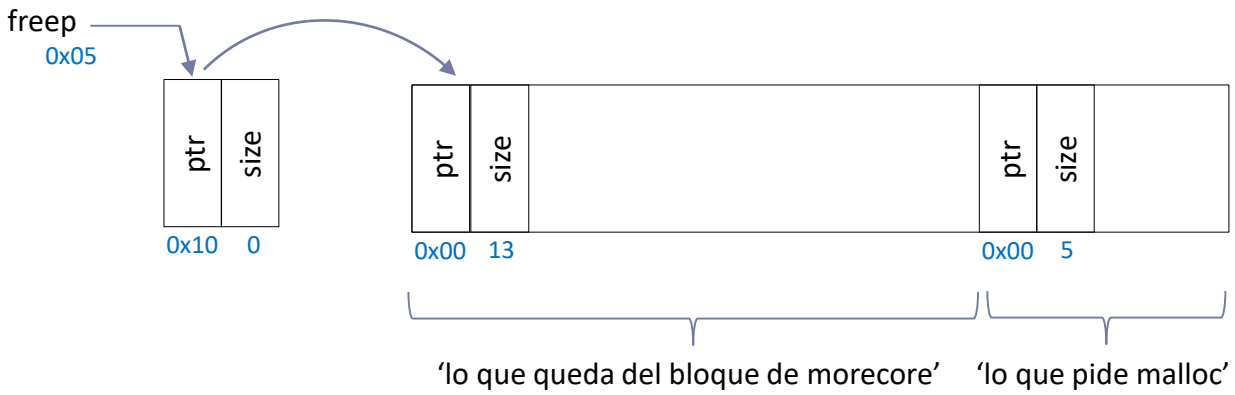
morecore



- morecore (int n_cab)
  - SI (n_cab < min_ncab)
    n_cab = min_ncab; // 144 bytes = 18 cabeceras
  - freep->ptr = sbrk(n_cab*2*sizeof(int))
  - freep->ptr->ptr=null;
  - freep->ptr->size=n_cab;

Ejemplo de *libc storage allocator*

malloc

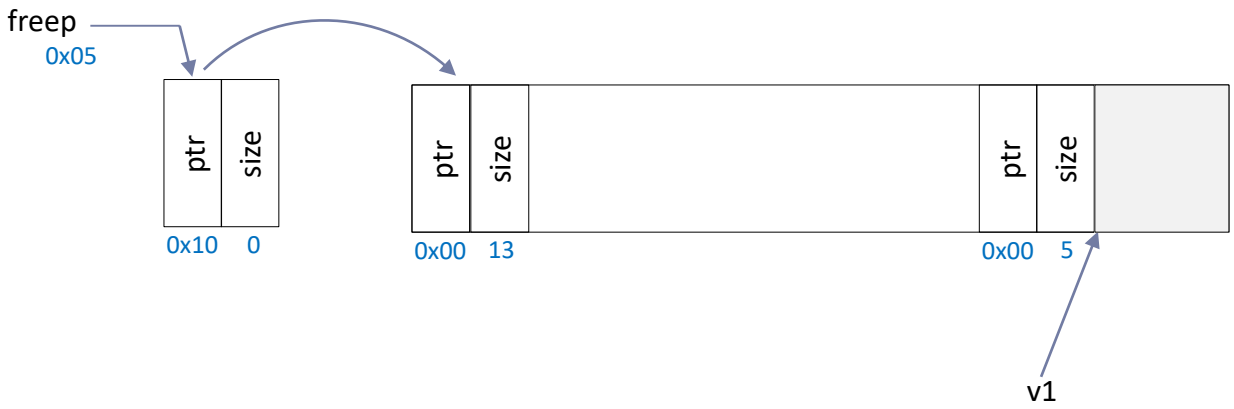


- int *v1;
- char *v2 ;

- v1 = malloc(8*sizeof(int)) ;

Ejemplo de *libc storage allocator*

malloc

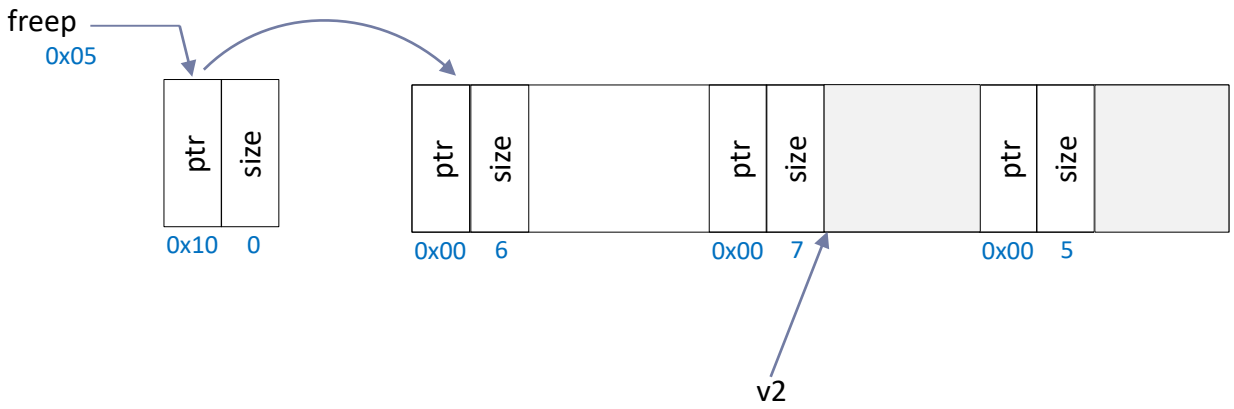


- int *v1;
- char *v2 ;

- v1 = malloc(8*sizeof(int)) ;

Ejemplo de *libc storage allocator*

malloc

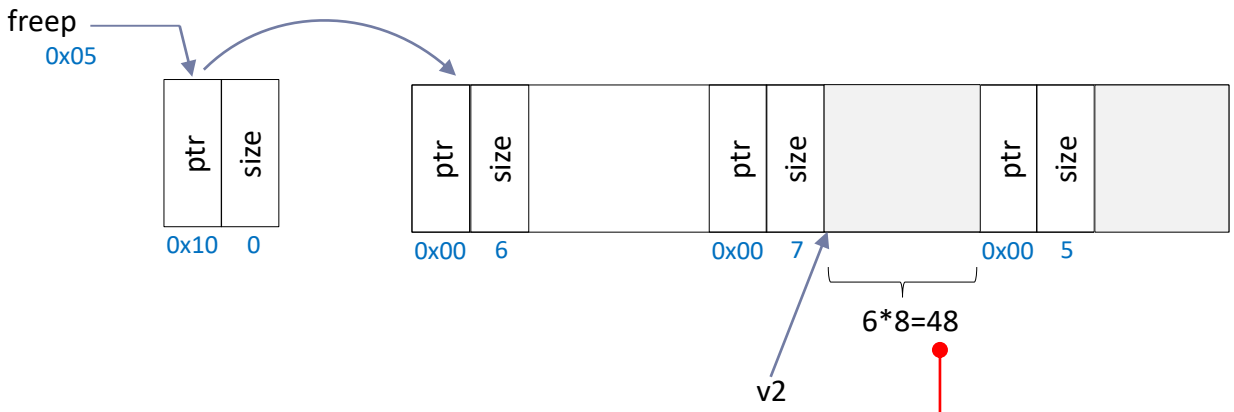


- int *v1;
- char *v2 ;

- v1 = malloc(8*sizeof(int)) ;
- v2 = malloc(41) ;

Ejemplo de *libc storage allocator*

problema de fragmentación interna

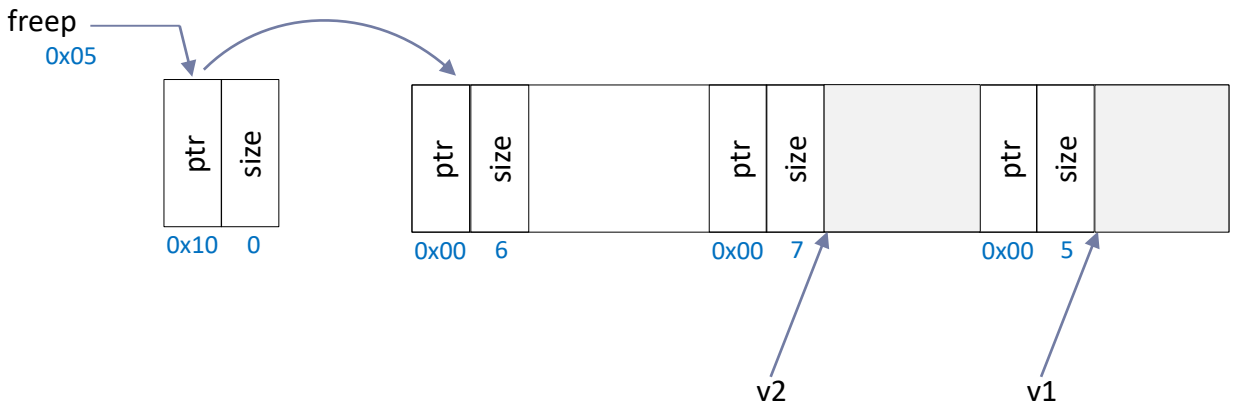


- int *v1;
- char *v2 ;

- v1 = malloc(8*sizeof(int)) ;
- v2 = malloc(41) ;

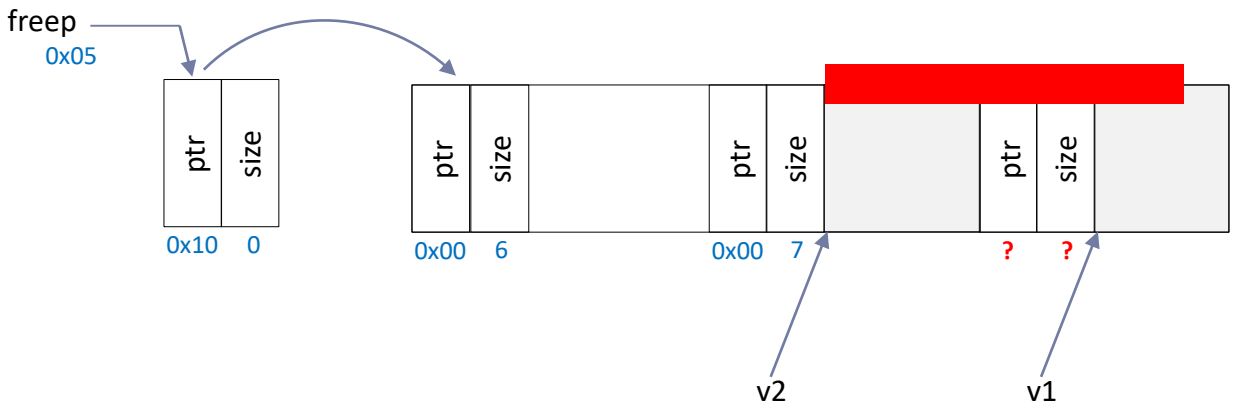
- Unidad de asignación es 8 bytes (1 cabecera de 2 enteros)
- Se redondea a múltiplo de la unidad de asignación

Ejemplo de *libc storage allocator*
problema de sobreescritura



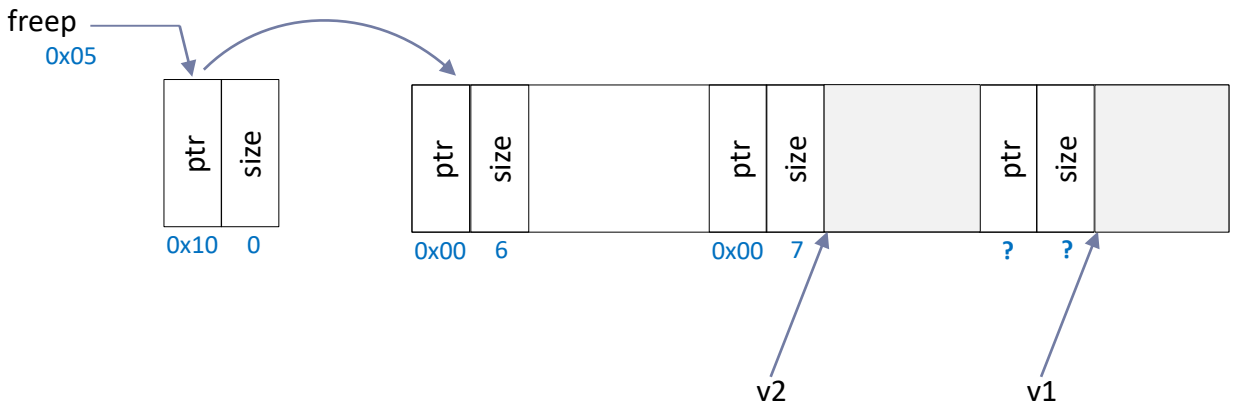
- // se ha reservado solo 41 caracteres para v2
- for (int i=0; i<64; i++)
  - v2[i] = 'x' ;
- free(v1) ;

Ejemplo de *libc storage allocator*
problema de sobreescritura



- // se ha reservado solo 41 caracteres para v2
- for (int i=0; i<64; i++)
  - v2[i] = 'x' ;
- free(v1) ;

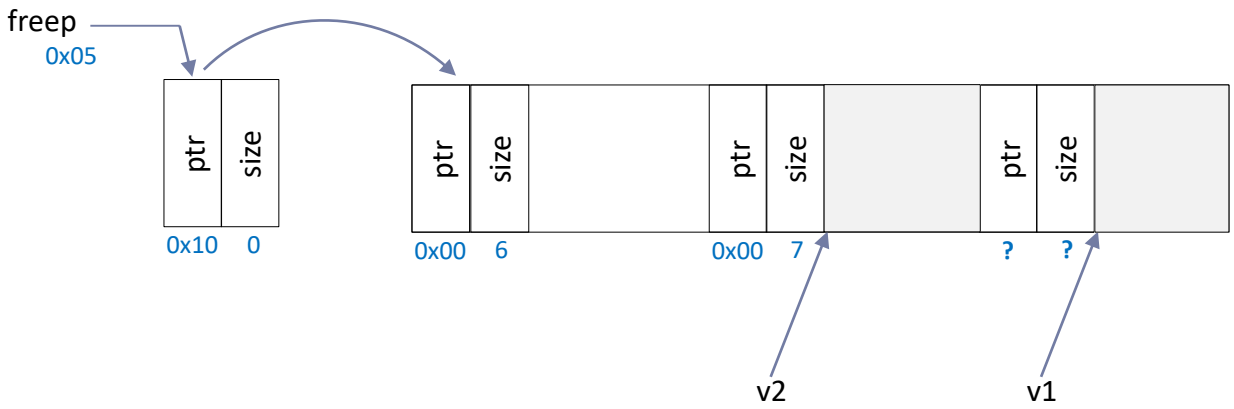
Ejemplo de *libc storage allocator*
problema de sobreescritura



- // se ha reservado solo 41 caracteres para v2
- for (int i=0; i<64; i++)
  - v2[i] = 'x' ;
- free(v1) ; <- incapaz de recuperar la cabecera válida… SIGSEV

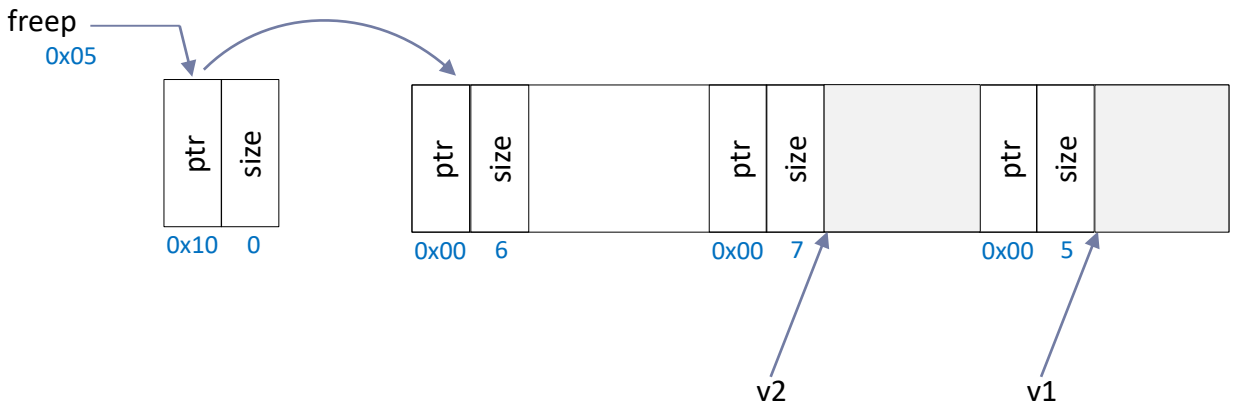
Ejemplo de *libc storage allocator*

otros problemas típicos



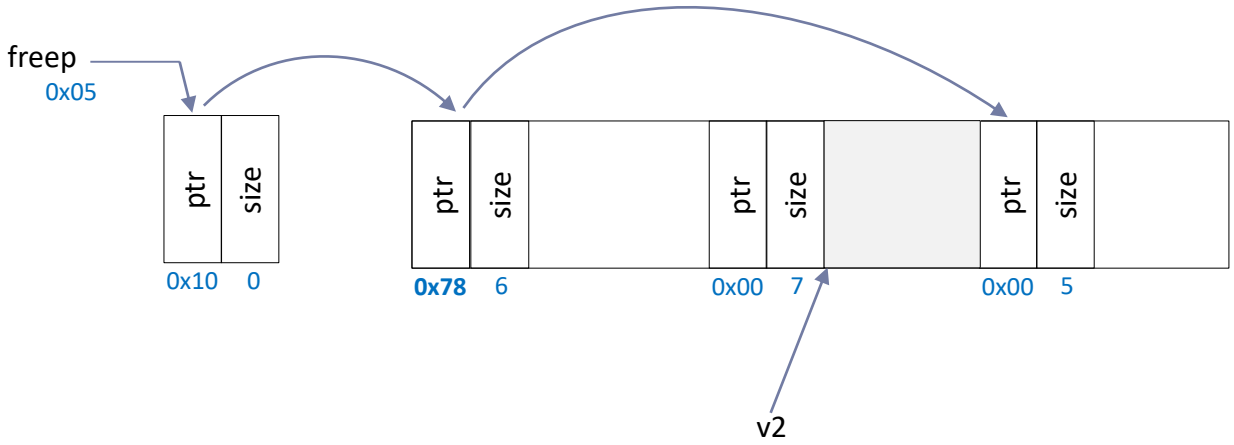
- Liberar una zona de memoria no gestionada:
  - int i; free(&i);
- Liberar dos veces una misma zona de memoria
- Acceder a memoria no pedida anteriormente
  - char *pchar; printf("%s",pchar);

Ejemplo de *libc storage allocator*
free



- free(v1) ;

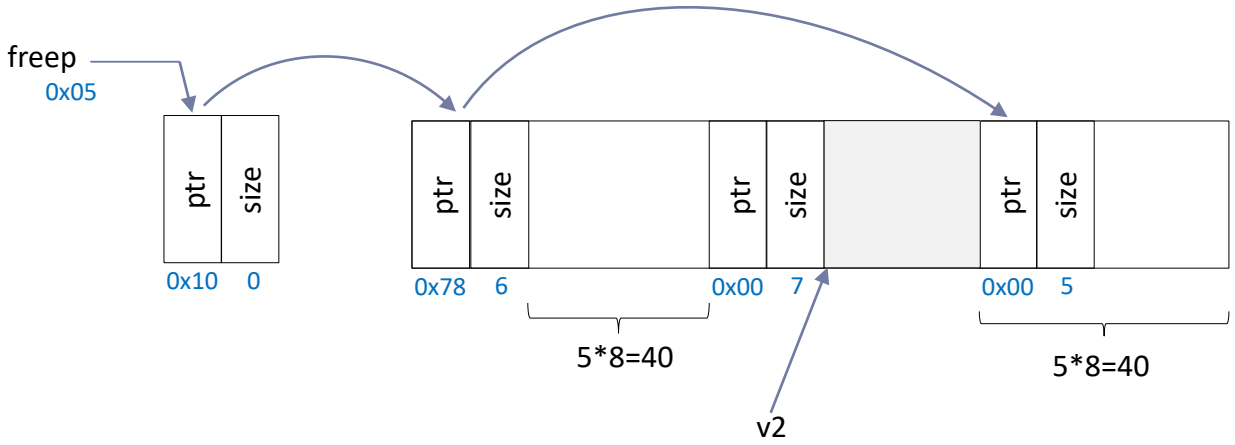
Ejemplo de *libc storage allocator*
free



- free(v1) ;

Ejemplo de *libc storage allocator*

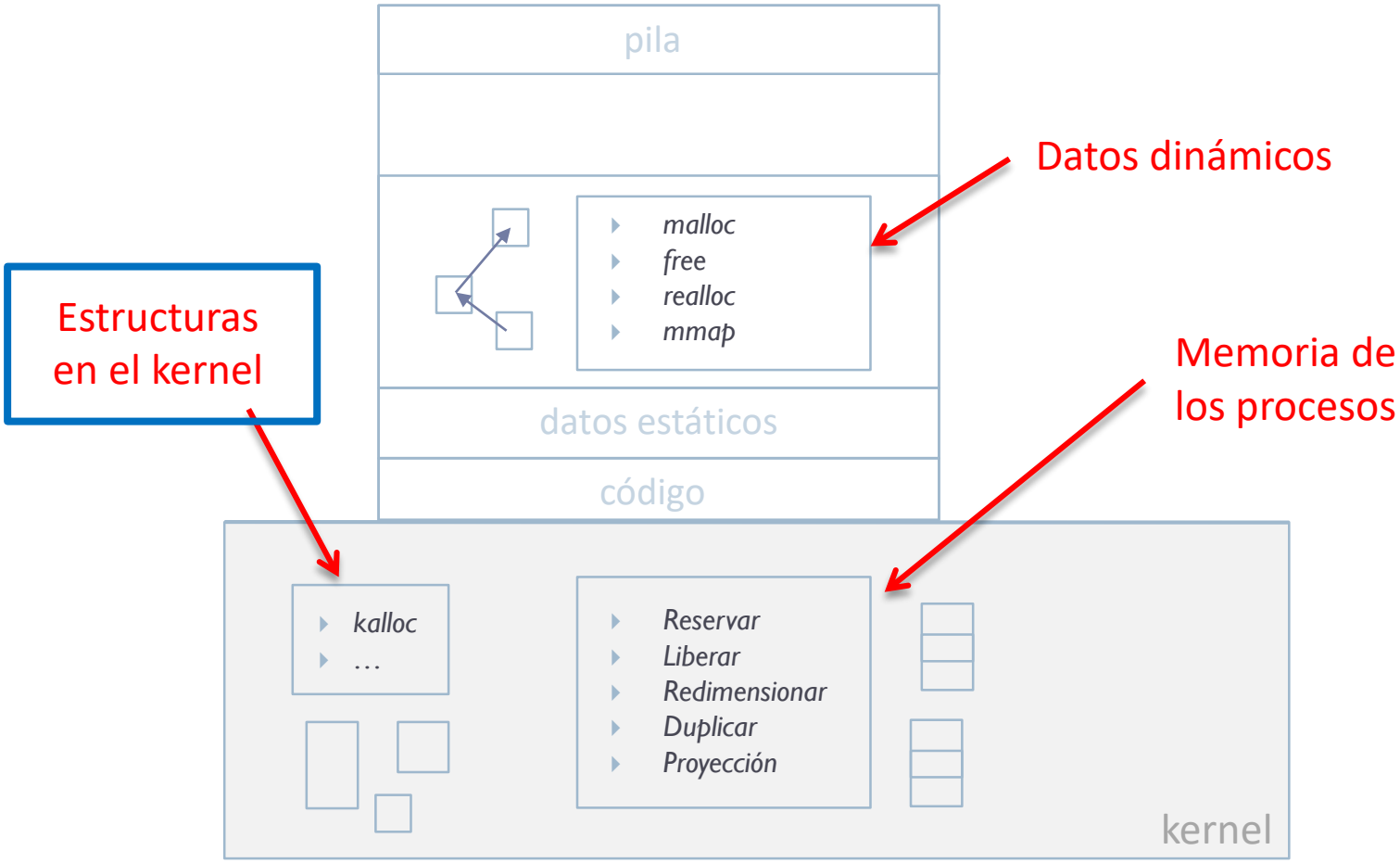
problema de fragmentación externa



- v1 = malloc(20*sizeof(int)) ; // 20*4 = 80 bytes

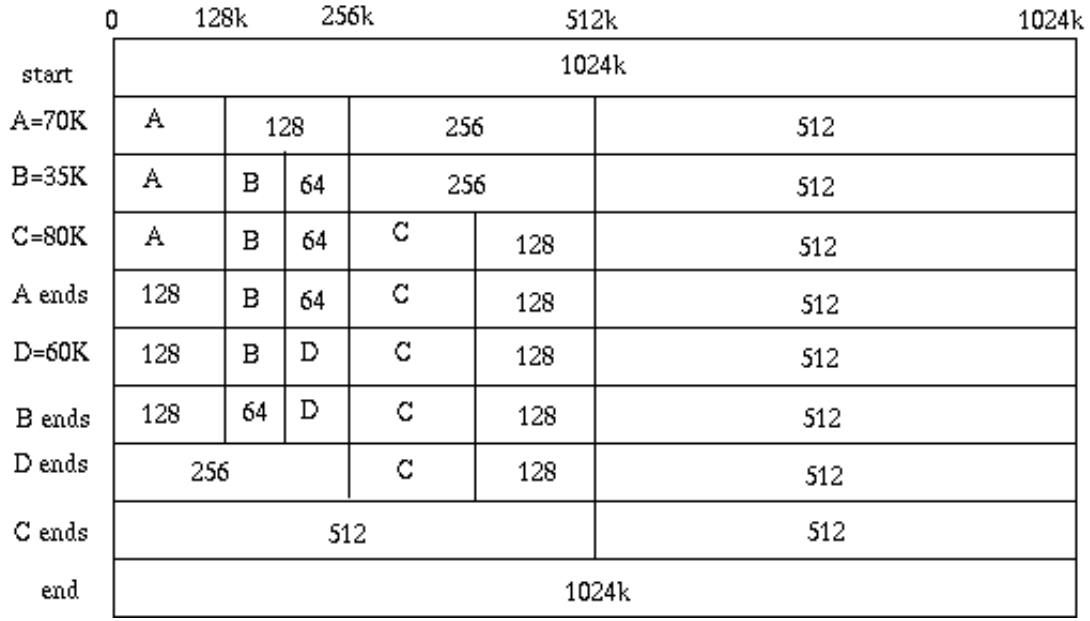
- Con el paso del tiempo, secuencias de llamadas a malloc+free que dejan muchos huecos entre bloques usados
  - Búsquedas lentas en listas enlazadas
  - Espacio libre hay para satisfacer la petición, pero no hueco de ese tamaño

Gestores a varios niveles



Gestión de la memoria en el kernel

- Con menor fragmentación externa y menor sobrecarga en la compactación: *buddy memory allocator*



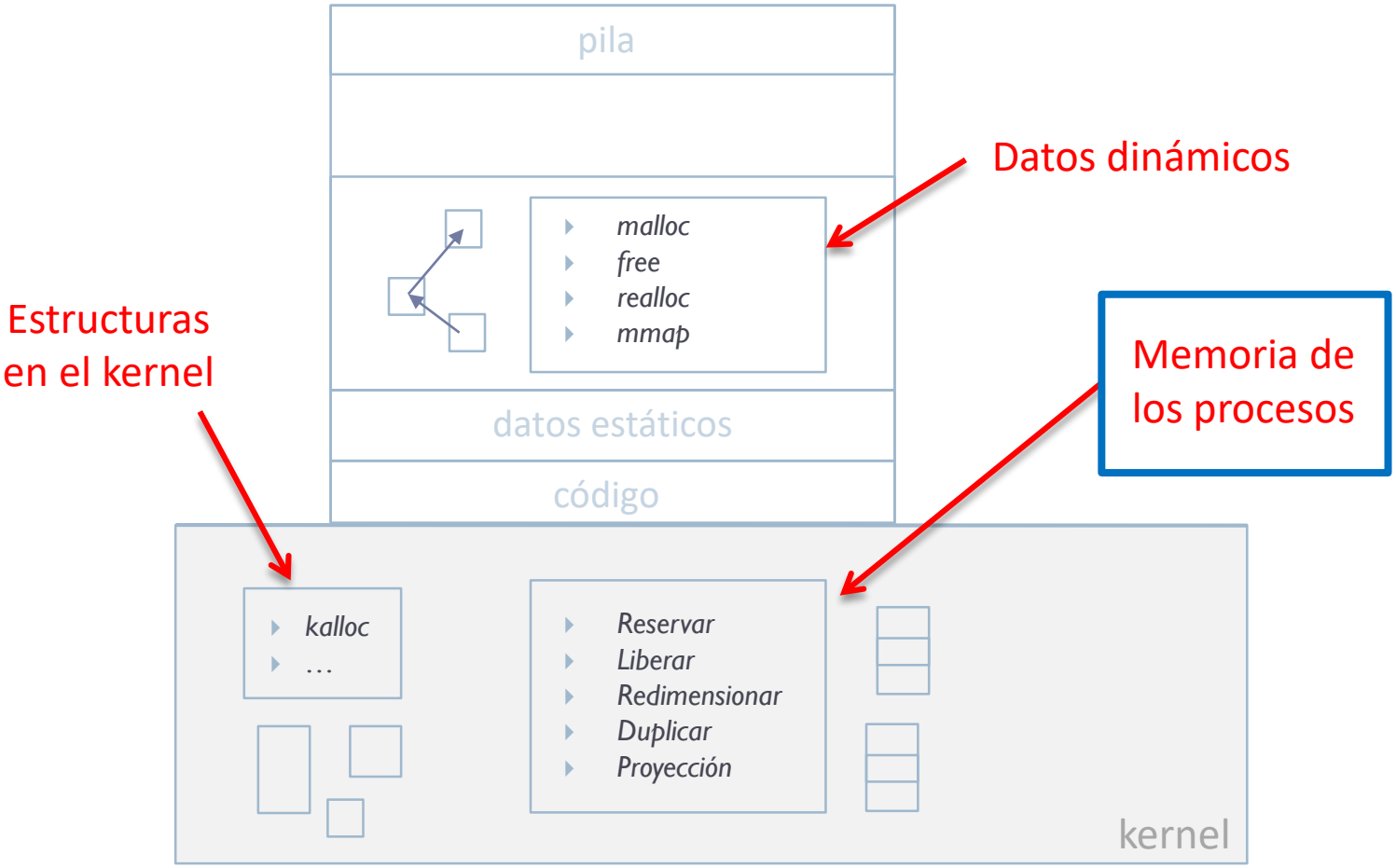
https://secure.wikimedia.org/wikipedia/en/wiki/Buddy_memory_allocation

Gestión de la memoria en el kernel

- En muchos kernels se utiliza el *slab allocation*
  - Ej.: Solaris, FreeBSD, Linux, etc.
- Basado en el *Mach's zone allocator*
- Tiene preasignado porciones de memoria del tamaño de los tipos de datos (objetos) más frecuentemente usados
  - De esta manera se elimina la búsqueda de hueco y la compactación después de la liberación
  - En estas condiciones, más eficiente y elimina fragmentación
- Es posible ver el uso en el kernel de dicho gestor mediante:
  - cat /proc/slabinfo

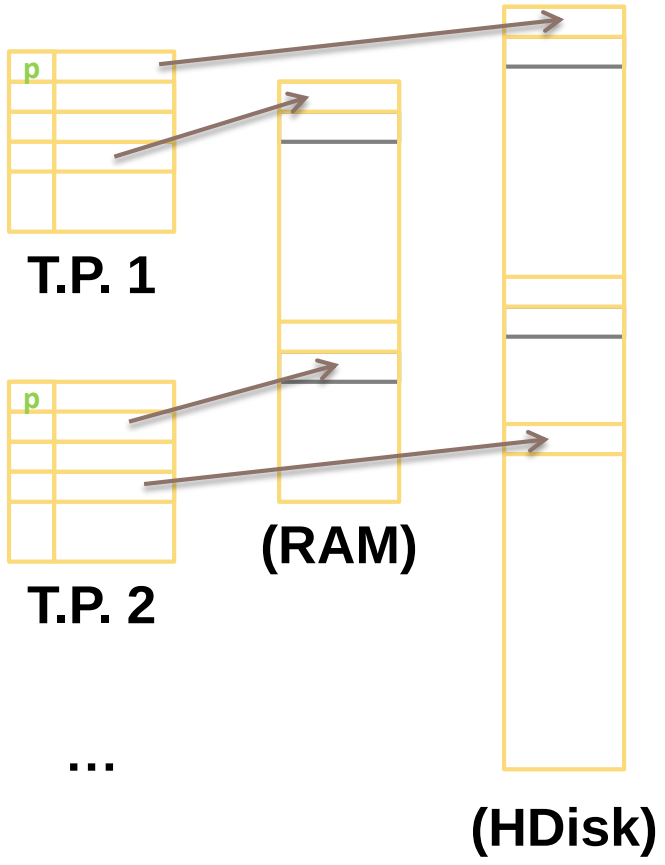
- http://www.ibm.com/developerworks/linux/library/l-linux-slab-allocator/
- http://www.arl.wustl.edu/~fredk/Courses/cse522/fall03/Lectures/kmem.ppt

Gestores a varios niveles

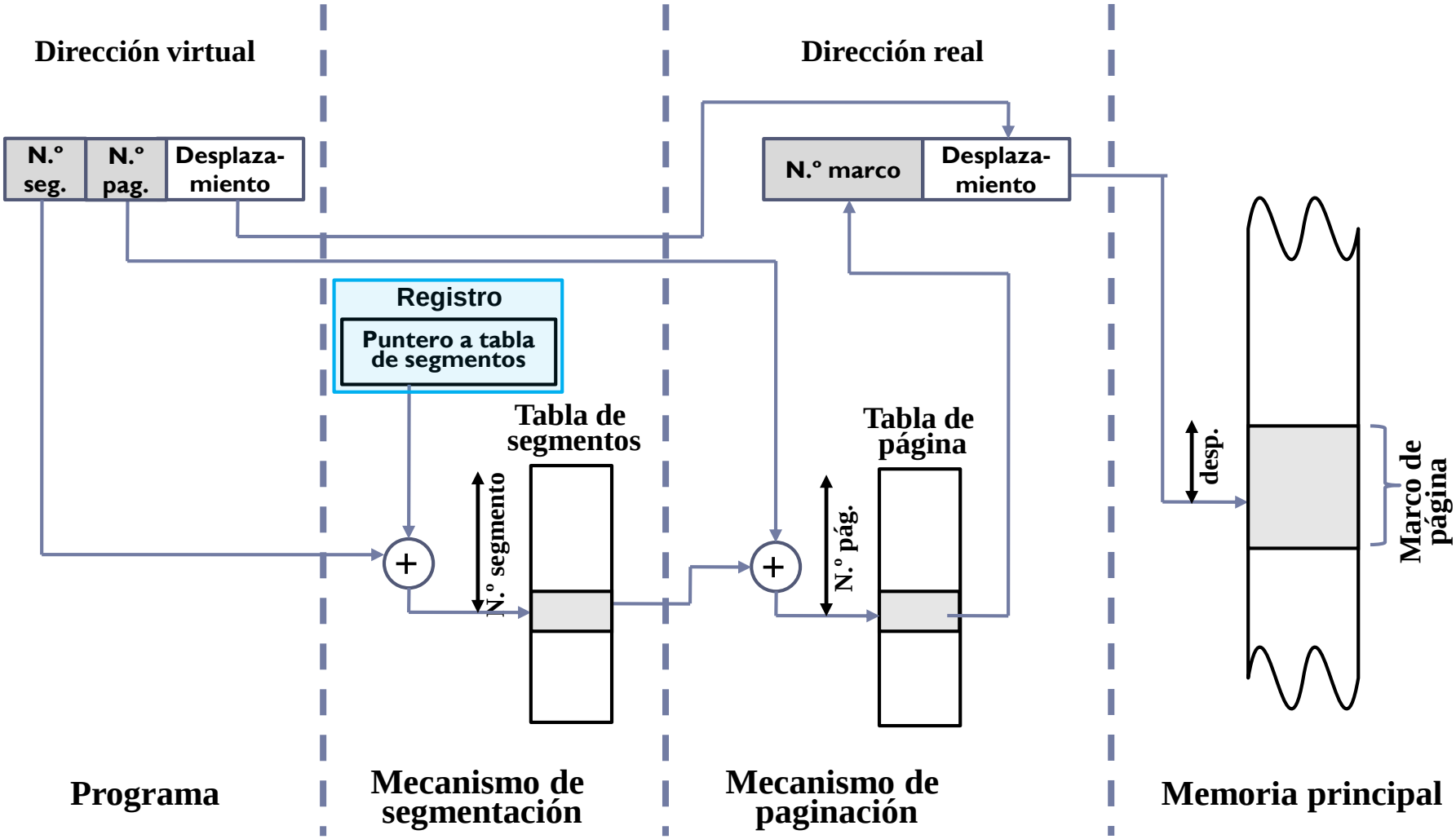


Trabajando con distintos espacios de memoria

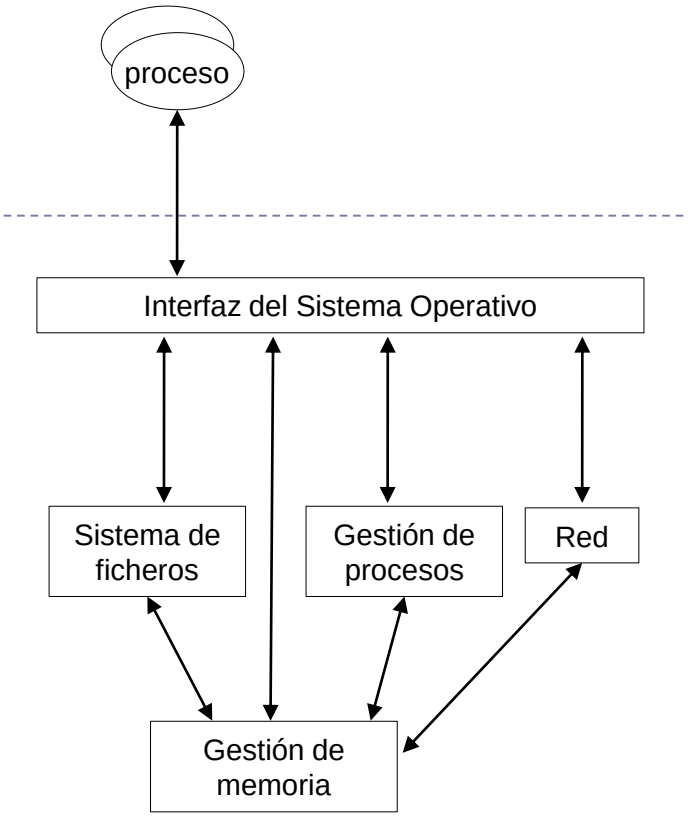
- Tabla de seg. pag. por proceso
- Un registro apunta a la tabla del proceso en ejecución actual



Trabajando con distintos espacios de memoria
segmentación paginada



Objetivos generales de la memoria
usando memoria virtual



1. Localización de referencias a memoria
   - ✓ La MMU se encarga de traducir un espacio virtual al real
2. Protección de espacios de memoria
   - ✓ Cada proceso ve su propio espacio
3. Compartición de espacios de memoria
   - ✓ Entradas de tabla de página que apuntan a los mismos marcos de páginas
4. Organización lógica (de programas)
   - ✓ Uso de segmentación
5. Organización física (de la memoria)
   - ✓ Segmentación paginada ofrecen mayor flexibilidad en la organización

Operaciones sobre regiones
creación de región

- Nueva región no se le asigna m. principal (carga por demanda)
  - Págs. de región se marcan como inválidas
- Dependiendo del tipo de soporte:
  - Soporte en archivo
    - Páginas se marcan como *Cargar de archivo (CA)*
    - Se almacena dirección del bloque del archivo correspondiente
  - Sin soporte
    - Por seguridad, las páginas se marcan como *Rellenar con ceros (RC)*
    - Fallo de página no implica lectura de dispositivo
- Una vez creada región, cuando se expulsa página modificada
  - Si la región es privada se escribe página en *swap*
  - Si la región es compartida se escribe página en soporte
- Pila es "especial": debe contener argumentos del programa
  - Típicamente se copian los argumentos en bloque(s) de *swap*

Operaciones sobre regiones
liberación de región

- Actualizar tabla de regiones para eliminar región
- Marcar como inválidas páginas asociadas
- Si región privada, se libera espacio de *swap*

- La liberación puede deberse a:
  - Solicitud explícita (Ej.: desproyección de región)
  - Finalización del proceso (EXIT en POSIX)
  - EXEC de POSIX libera mapa actual del proceso antes de construir nuevo mapa vinculado al ejecutable

Operaciones sobre regiones
cambio de tamaño de una región

- Si disminuye, similar a liberación pero sólo parte afectada
- Si aumenta tamaño:
  - Comprobar no solapamiento
  - Si preasignación: reserva espacio en *swap* para nuevas páginas
- Casos especiales:
  - Expansión del *heap* y archivos proyectados
    - Solicitada por programa mediante servicio de S.O.
  - Expansión de pila más compleja: es "automática"
    - Programa disminuye valor de SP y accede a zona expandida
      - Fallo de página
    - Tratamiento de fallo de página:
      - Si dirección realmente inválida
        - Si dirección < SP –› Aborta proceso o le manda señal
        - Si no –› Expansión de pila

Operaciones sobre regiones
duplicación de una región

- Necesario para FORK de UNIX
  - Operación costosa e ineficiente: se debe copiar contenido
- Optimización: *copy-on-write* (COW)
  - Se comparten páginas de regiones duplicadas pero:
    - se marcan de sólo lectura y con bit de COW
    - primera escritura –› Fallo de protección –› copia privada
  - Puede haber varios procesos con misma región duplicada
    - Existe un contador de uso por página
    - Cada vez que se crea copia privada se decrementa contador
    - Si llega a 1, se desactiva COW ya que no hay duplicados
  - El FORK no duplicar espacio de memoria, sólo duplica la TP

Operaciones sobre regiones
archivos proyectados en memoria

- Programa solicita proyección de archivo (o parte) en su mapa
  - Puede especificar protección y si privada o compartida
- S.O. rellena entradas correspondientes con:
  - No residente, CA
  - Privada/Compartida y Protección según especifica la llamada
  - Entradas de TP referencian a un archivo de usuario

- Se usa como:
  - Forma alternativa de acceso a archivos frente a *read/write*
  - Carga de bibliotecas dinámicas
  - Globalmente: generalización de memoria virtual

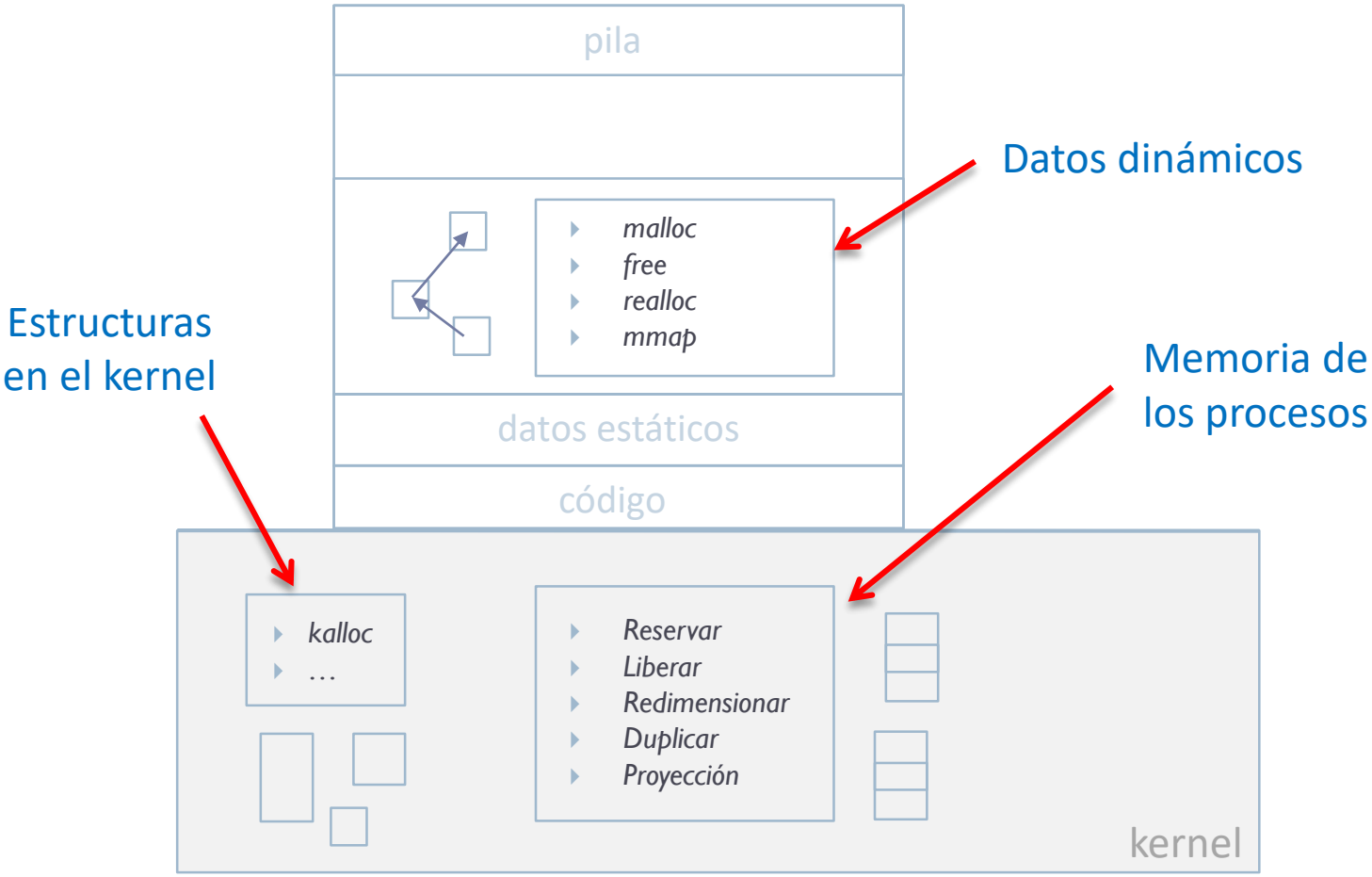
Operaciones sobre regiones
deduplicación de una región

- *copy-on-write* (COW) busca compartir páginas entre proceso padre e hijo para evitar páginas con el mismo contenido.
  - Cuando se modifica es cuando se hace una copia y se modifica esa copia del contenido.
- La deduplicación (KSM) busca compartir páginas entre procesos sin relación de parentesco para evitar páginas con el mismo contenido.
  - Se detecta dos páginas con el mismo contenido y se manipulan las tablas de página/segmentos para compartirlas.
  - Cuando se modifica es cuando se hace una copia y se modifica esa copia del contenido.

http://en.wikipedia.org/wiki/Kernel_same-page_merging

Gestores a varios niveles

resumen

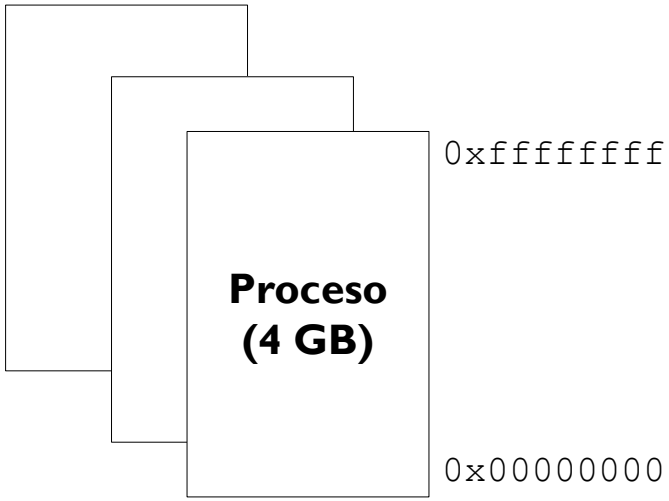


Contenidos

1. Introducción
2. Mecanismos para la gestión de la memoria
3. **Políticas y directrices de gestión**
   a. **Kernel/Procesos**
   b. Parámetros
   c. Aspectos

Espacio de memoria: proceso + kernel

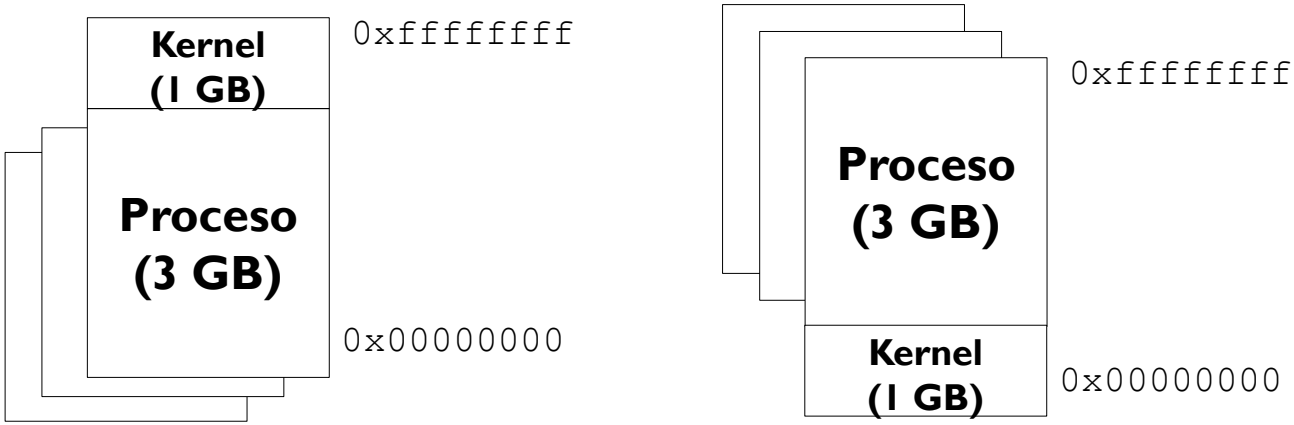
- Cada proceso ve un espacio de direcciones lineal y plano
  - Cada proceso podría acceder a todo el espacio de direcciones posible



http://www.cs.rutgers.edu/~pxk/416/notes/content/09-memory_management-slides-6.pdf

Espacio de memoria: proceso + kernel

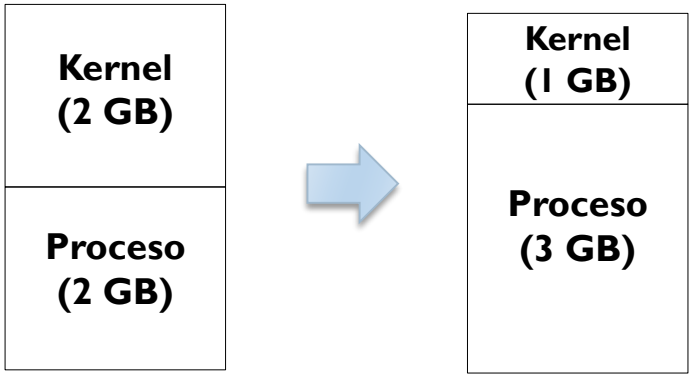
- El espacio usado por el kernel es compartido por todos los procesos
  - No cambia en los cambios de contexto
- El espacio del kernel está protegido (lectura, escritura y ejecución)
  - La mayoría de llamadas al sistema más rápidas (evita cambio de modo u→k y k → u)

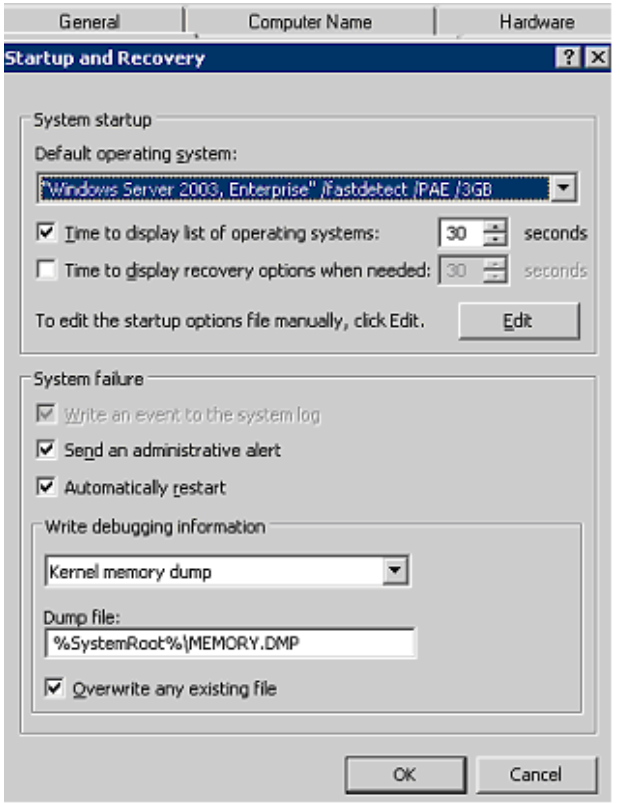


http://www.cs.rutgers.edu/~pxk/416/notes/content/09-memory_management-slides-6.pdf

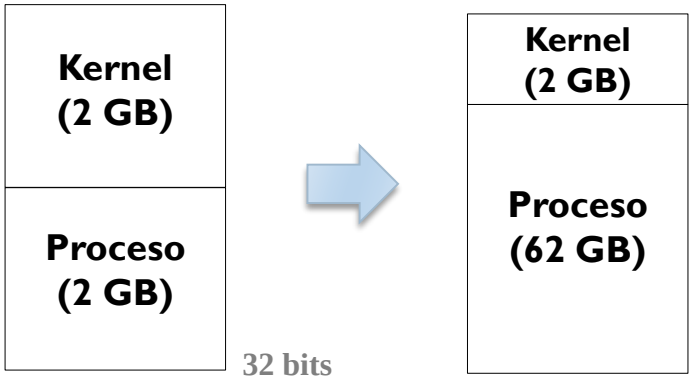
Espacio de memoria: proceso + kernel
Windows

- División configurable: /3GB





- Espacio extensible: /PAE



32 bits

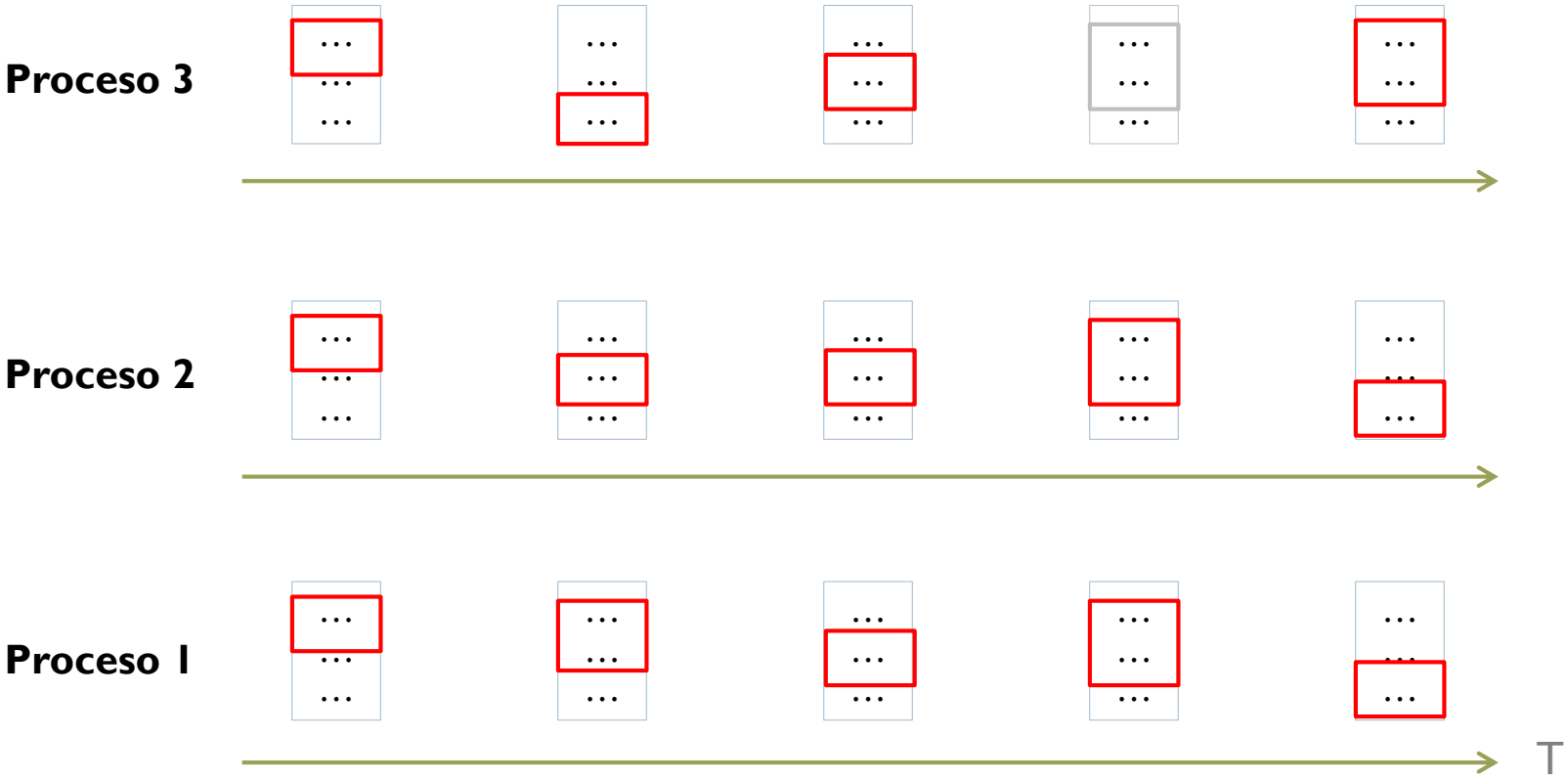
36 bits

https://msdn.microsoft.com/es-es/library/windows/desktop/aa366796(v=vs.85).aspx

Contenidos

1. Introducción
2. Mecanismos para la gestión de la memoria
3. **Políticas y directrices de gestión**
   a. Kernel/Procesos
   b. **Parámetros**
      - **Tamaño de página**
      - **Conjunto residente**
      - **Grado de multiprogramación**
   c. Aspectos

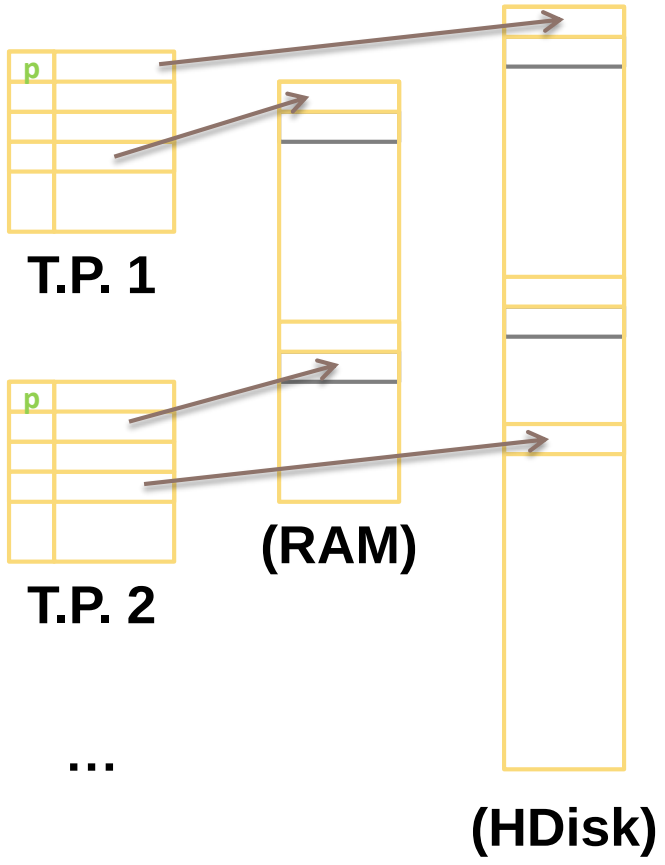
Trabajando con distintos espacios de memoria



Principales parámetros (1/4)

- Tabla de seg. pag. por proceso
- Un registro apunta a la tabla del proceso en ejecución actual

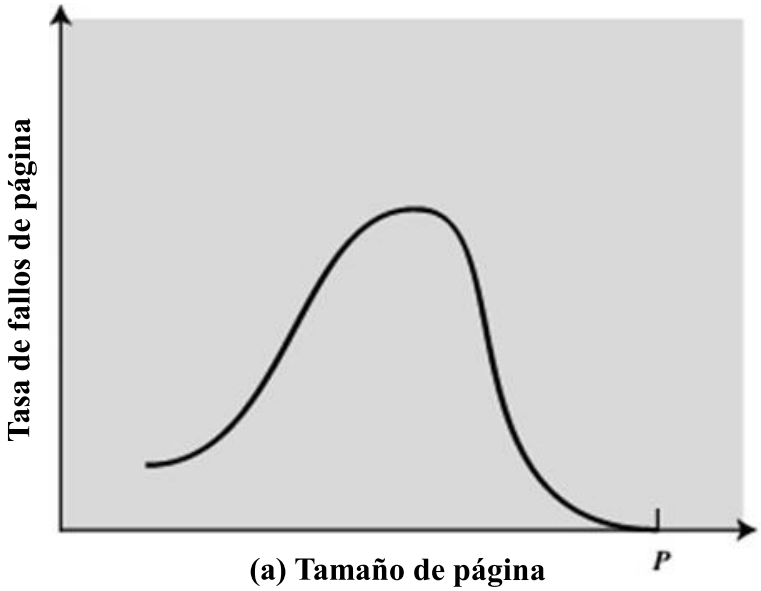
- **Grado de multiprogramación:** número de procesos en memoria en un instante dado
- **Conjunto residente:** número de páginas de un proceso en M.P. en un instante dado
- **Tamaño de página:** tamaño de página para todo el sistema en bytes



Principales parámetros (2/4)

- Hay que equilibrar:
  - El número de procesos en memoria (**grado de multiprogramación**)
  - El número de páginas que tiene cada proceso en M.P. (**conjunto residente**) con el mínimo que necesita para trabajar (**conjunto de trabajo**)
  - El tamaño de la página.
    - Tamaño de T.P., transferencia con M.S., número de fallos, etc.

Comportamiento típico de la paginación en un programa.



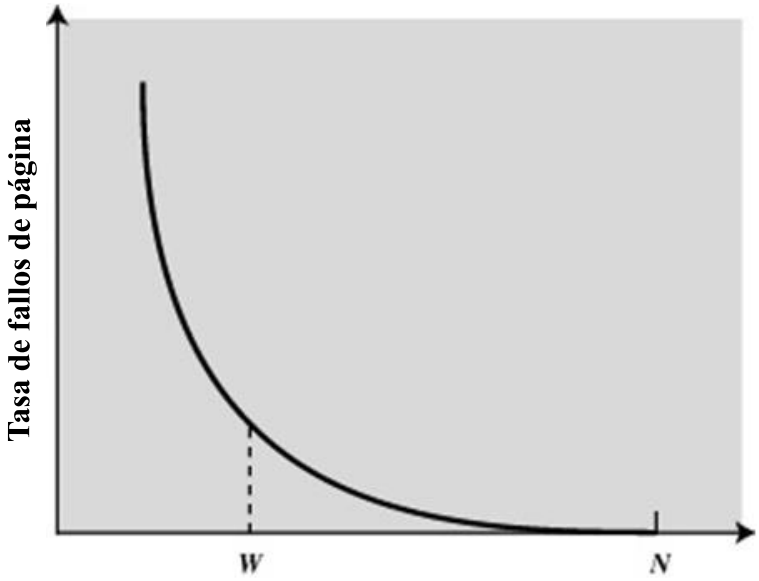
(a) Tamaño de página

P **= Tamaño del proceso completo**
W **= Tamaño del conjunto de trabajo**
N **= Número total de páginas del proceso**

Principales parámetros (3/4)

- Hay que equilibrar:
  - El número de procesos en memoria (**grado de multiprogramación**)
  - El número de páginas que tiene cada proceso en M.P. (**conjunto residente**) con el mínimo que necesita para trabajar (**conjunto de trabajo**)
  - El tamaño de la página.
    - Tamaño de T.P., transferencia con M.S., número de fallos, etc.

Comportamiento típico de la paginación en un programa.



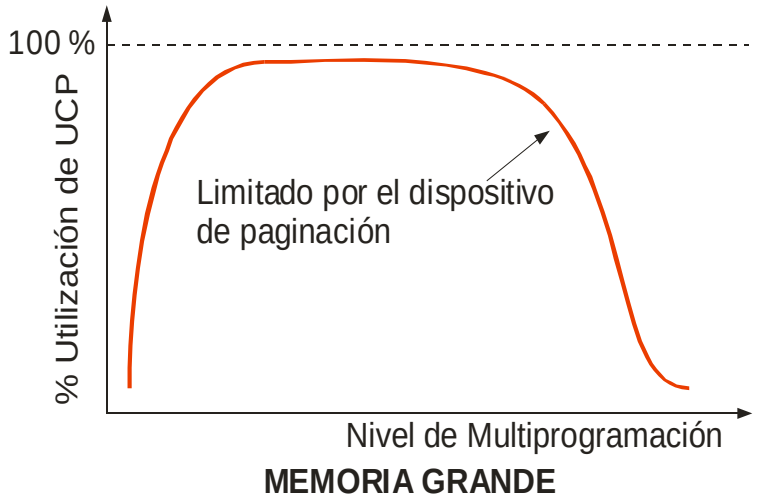
(b) Número de marcos de página

P **= Tamaño del proceso completo**
W **= Tamaño del conjunto de trabajo**
N **= Número total de páginas del proceso**

Principales parámetros (4/4)

- Hay que equilibrar:
  - El número de procesos en memoria (**grado de multiprogramación**)
    - **Swaping**:
      - Trasiego de información entre M. Principal y M. secundaria.
    - **Hiperpaginación**:
      - Se produce cuando el número de fallos es muy elevado
      - El sistema está más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario.
  - ...

Comportamiento típico de la paginación con varios programas.



Soluciones a la hiperpaginación (1/2)

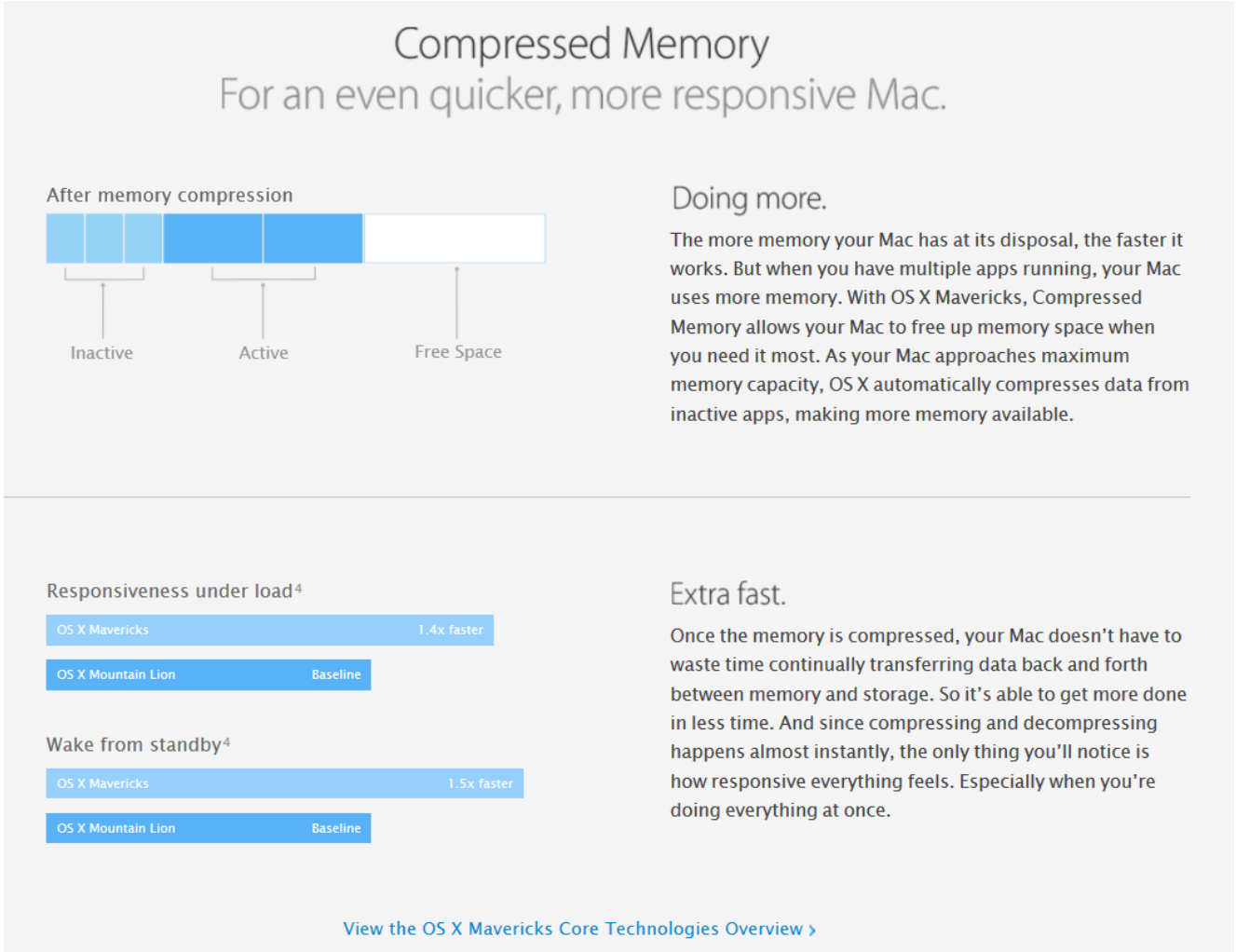
- Soluciones con reemplazo local
  - Estrategia del conjunto de trabajo
    - Intentar conocer el conjunto de trabajo de cada proceso
    - Si conjunto de trabajo decrece se liberan marcos
    - Si conjunto de trabajo crece se asignan nuevos marcos
      - si no hay disponibles: suspender proceso(s)
      - se reactivan cuando hay marcos suficientes para el conjunto de trabajo
  - Estrategia basada en frecuencia de fallos
    - Si tasa < límite inferior se liberan marcos
    - Si tasa > límite superior se asignan nuevos marcos
      - Si no marcos libres se suspende algún proceso

Soluciones a la hiperpaginación (2/2)

- Soluciones con reemplazo global
  - No existe soluciones adecuadas
  - BSD: Uso de *buffering* activando el demonio con un umbral.
    - Si se activa frecuentemente -> suspender algún proceso.
    - Idea general: mantener una reserva de marcos libres
    - 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
    - Si se referencia una página de las listas se usa sin más.

Soluciones a la hiperpaginación…

MacOS

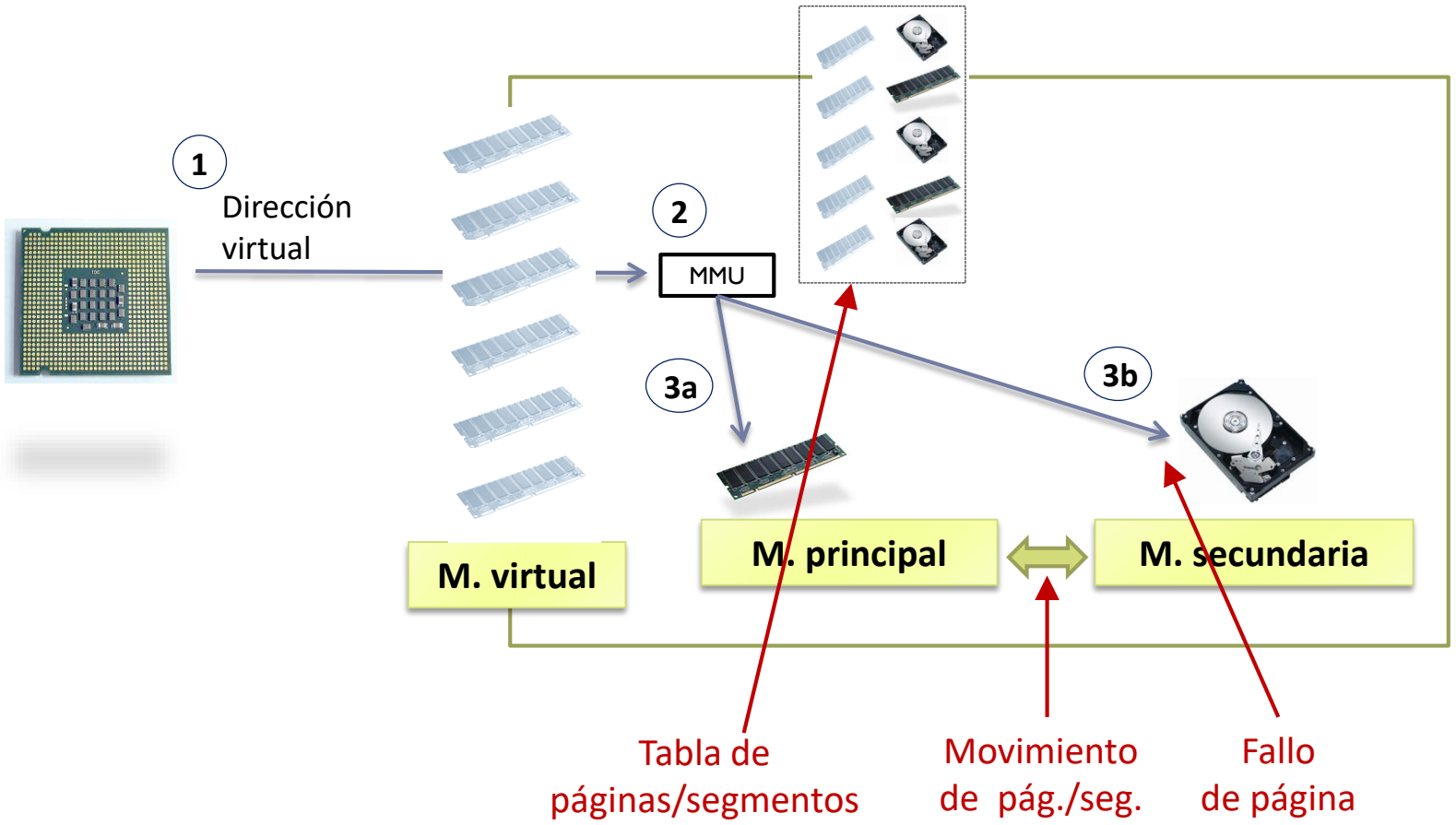


http://www.zdnet.com/mavericks-memory-compression-review-7000022515/

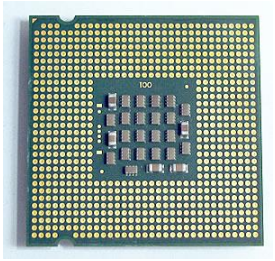
Contenidos

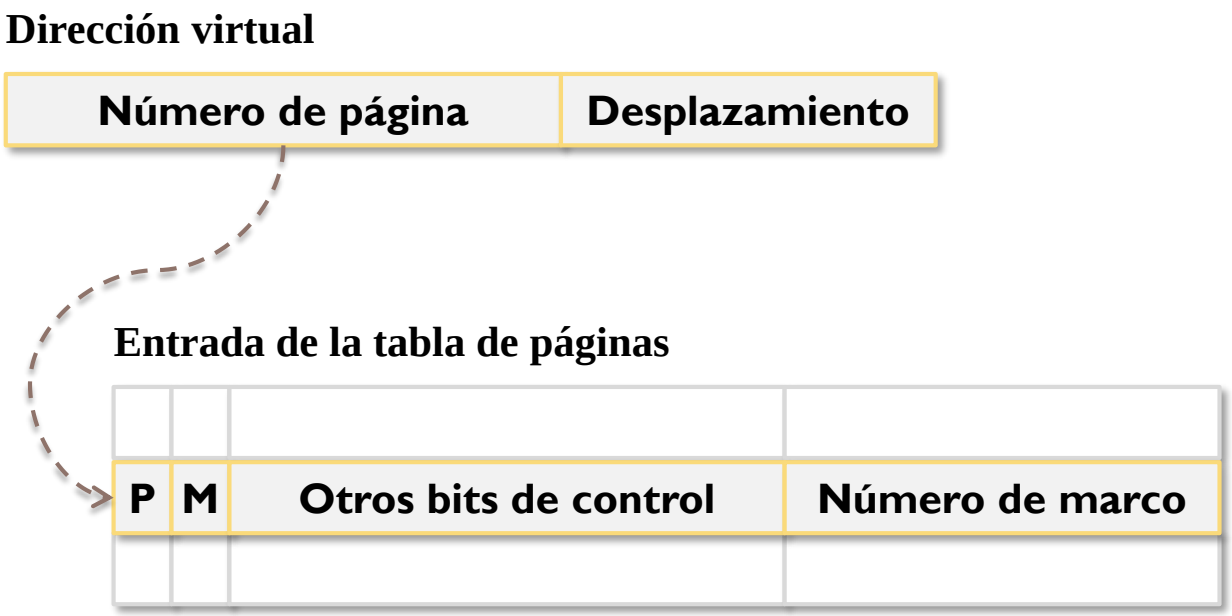
1. Introducción
2. Mecanismos para la gestión de la memoria
3. **Políticas y directrices de gestión**
   a. Kernel/Procesos
   b. Parámetros
   c. **Aspectos**
      - **Tabla de páginas/segmentos**
      - **Movimiento de páginas/segmentos**

Sistemas con memoria virtual



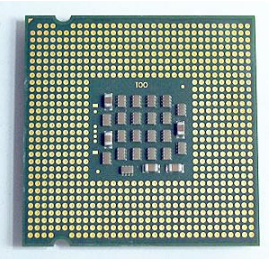
Entradas de la tabla de páginas
formato típico





- Bit P: indica si está presente la página en M.P.
- Bit M: indica si ha sido modificada la página en M.P.
- Otros bits: protección (lectura, escritura, ejecución, etc.), gestión (cow, etc.)

Entradas de la tabla de segmentos
formato típico



Dirección virtual

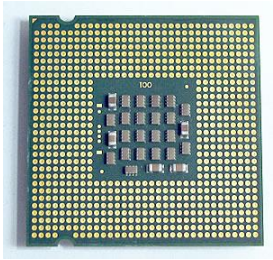
Número de segmento	Desplazamiento

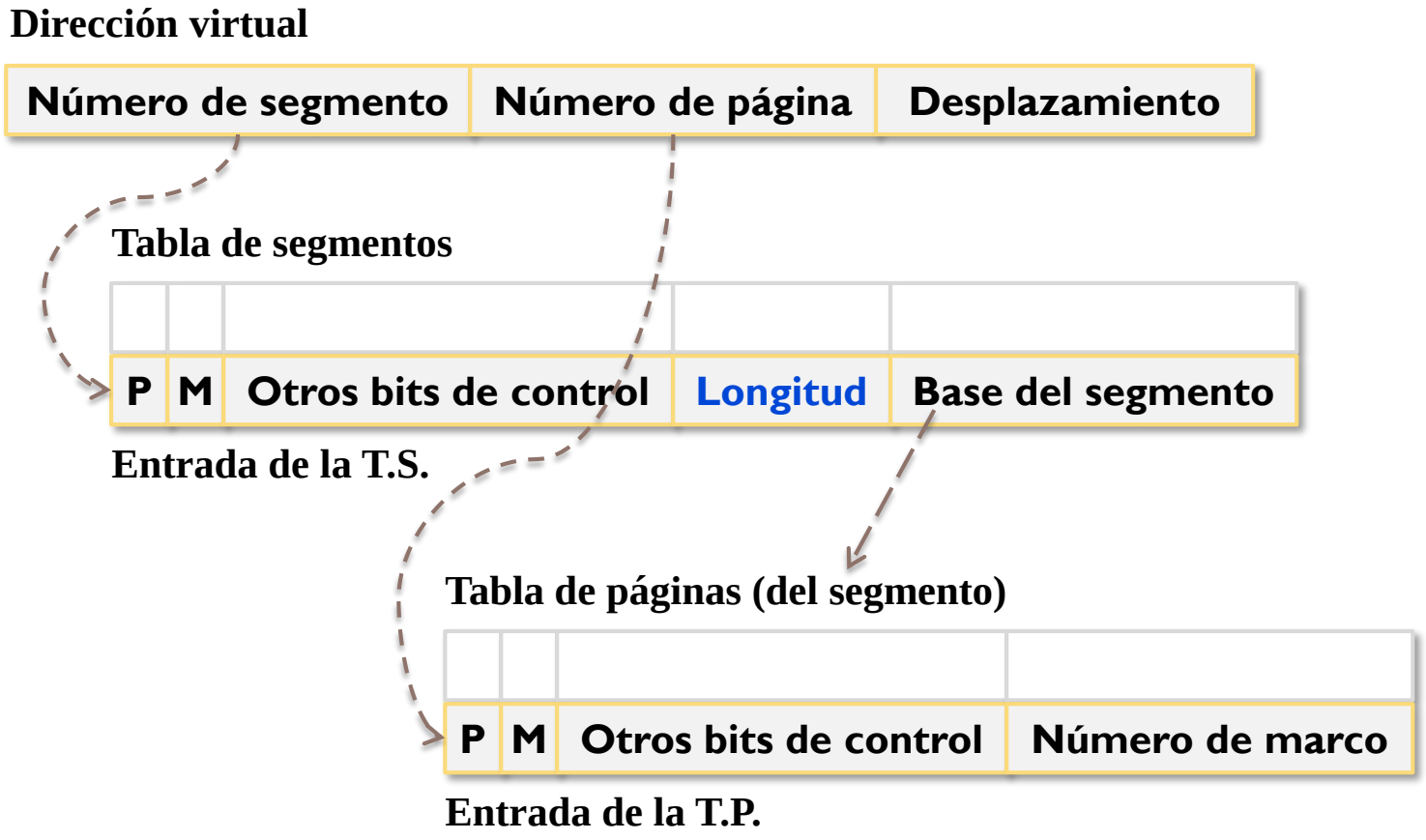
Entrada de la tabla de segmentos

P	M	Otros bits de control	Longitud	Base del segmento

- Bit P: presente
- Bit M: modificado
- Otros bits de control: R,W,X,COW,…

Entradas de la tablas
formato típico





Gestión de la tabla de páginas

- Inicialmente:
  - La crea el sistema operativo cuando se va a ejecutar el programa.
- Uso:
  - La consulta la MMU en la traducción.
- Actualización:
  - La modifica el sistema operativo en los fallos de página.

Movimiento de los segmentos

- Inicialmente:
  - **Se configuran desde el ejecutable de una aplicación**
  - **Código se carga, pila se inicializa, etc.**
- De M. secundaria a M. principal (por demanda):
  - Acceso a segmento no residente: Fallo de segmento
  - S.O. lee el segmento de M. secundaria y la lleva a MP
- De M. principal a M. secundaria (por expulsión):
  - No hay espacio en M. principal para traer un segmento
  - Se expulsa (reemplaza) un segmento residente
  - S.O. escribe segmento expulsado a M. secundaria (si bit M=1)

Movimiento de las páginas

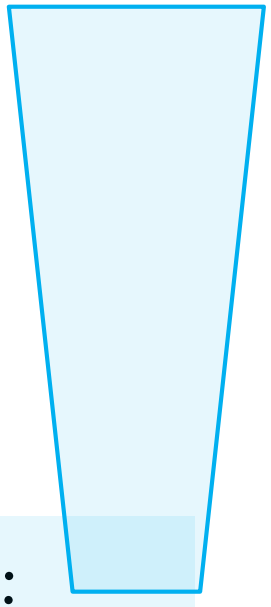
- Inicialmente:
  - Página no residente se marca ausente
  - Se guarda dirección del bloque de *swap* que la contiene
- De M. secundaria a M. principal (por demanda):
  - Acceso a pág. no residente: Fallo de página
  - S.O. lee página de M. secundaria y la lleva a M. principal
- De M. principal a M. secundaria (por expulsión):
  - No hay espacio en M. principal para traer página
  - Se expulsa (reemplaza) una página residente
  - S.O. escribe página expulsada a M. secundaria (si bit M=1)

Tratamiento (general) del fallo de página

- Si dirección inválida –› aborta proceso o le manda señal
- Si no hay ningún marco libre (consulta T. marcos)
  - Selección de víctima (alg. de reemplazo): página P marco M
    - Marca P como inválida
  - Si P modificada (bit *Mod* de P activo)
    - Inicia escritura P en memoria 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)

Movimiento de las páginas

- Inicialmente:
  - Página no residente se marca ausente
  - Se guarda dirección del bloque de *swap* que la contiene
- De M. secundaria a M. principal (por demanda):
  - Acceso a pág. no residente: Fallo de página
  - S.O. lee página de M. secundaria y la lleva a M. principal
- De M. principal a M. secundaria (por expulsión):
  - No hay espacio en M. principal para traer página
  - Se expulsa (reemplaza) una página residente
  - S.O. escribe página expulsada a M. secundaria (si bit M=1)



Se necesita soporte en hardware

Políticas de administración

- Política de reemplazo:
  - Reemplazo local: dentro del proceso
  - Reemplazo global
- Algoritmos de reemplazo: validos para local y global
  - Óptimo
  - FIFO
  - Reloj (o segunda oportunidad)
  - LRU
- Política de asignación de marcos a los procesos:
  - Asignación fija (siempre con reemplazo local):
    - conjunto residente del proceso es constante
  - Asignación dinámica (reemplazo local o global):
    - conjunto residente del proceso es variable

Algoritmos de **no** reemplazo

- Bloqueo de marcos:
  - Cuando un marco está bloqueado, la página cargada en ese marco no puede ser reemplazada.
- Ejemplos de cuándo se bloquea un marco:
  - La mayoría del núcleo del sistema operativo.
  - Estructuras de control.
  - Buffers de E/S (Ej.: los usados con DMA).
- **El bloqueo se consigue asociando un bit de bloqueo a cada marco.**

Algoritmos de reemplazo

- Qué página se va a reemplazar.
- La página que se va a reemplazar tiene que ser la que tenga una menor posibilidad de ser referenciada en un futuro cercano.
- La mayoría de las políticas intentan predecir el comportamiento futuro en función del comportamiento pasado.
- Ejemplo de políticas: **LRU, FIFO, etc.**

Algoritmos básicos de reemplazo

- Política **óptima**:
  - Selecciona para reemplazar la página que tiene que esperar una mayor cantidad de tiempo hasta que se produzca la referencia siguiente.
  - Es imposible de implementar porque requiere que el sistema operativo tenga un conocimiento exacto de los sucesos futuros.

Algoritmos básicos de reemplazo

- Política 'la usada menos recientemente' (**LRU**):
  - Reemplaza la página de memoria que no ha sido referenciada desde hace más tiempo.
  - Debido al principio de cercanía, ésta sería la página con menor probabilidad de ser referenciada en un futuro cercano.
  - **Una solución sería etiquetar cada página con el momento de su última referencia.**

Algoritmos básicos de reemplazo

- Política 'primera en entrar, primera en salir' (**FIFO**):
  - Reemplazar la página en memoria que primero se cargo (que ha estado más tiempo en memoria)
  - Estas páginas pueden necesitarse de nuevo y en un plazo de tiempo corto.
  - Es una de las políticas de reemplazo más sencillas de implementar:
    - **Trata los marcos asignados a un proceso como un buffer circular.**
    - **Las páginas se suprimen de la memoria según la técnica de turno rotatorio (*round-robin*).**

Grupo ARCOS
Universidad Carlos III de Madrid

Lección 5 (b)
La gestión de memoria

Diseño de Sistemas Operativos
Grado en Ingeniería Informática