

Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 1 (c)

## libc: gestión de memoria dinámica

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.



# Objetivos generales

1. Conocer el **espacio de memoria** de un proceso.
  1. Regiones de memoria, preparación de un ejecutable, etc.
2. **Gestores de memoria:**
  1. *Heap* para usuario, en kernel, memoria virtual, etc.

# A recordar...

---

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:  
las transparencias solo no son suficiente.  
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de prácticas** y las **prácticas** de forma progresiva.

# Ejercicios, cuadernos de prácticas y prácticas

Ejercicios ✓	Cuadernos de prácticas ✗	Prácticas ✗																
<p>© copyright all rights reserved</p> <p>Grado en Ingeniería Informática Diseño de Sistemas Operativos [5] Gestión de memoria</p> <p>ARCOS</p> <p>Grupo: ..... NIA: ..... Nombre y apellidos: .....</p> <p><b>Ejercicio 1</b></p> <p>El gestor de memoria de un sistema operativo debe permitir que sea posible ejecutar concurrentemente 3 procesos, cuyos tamaños en bytes de cada uno de los segmentos que forman parte de sus imágenes de memoria son los siguientes:</p> <table border="1"><thead><tr><th>PROCESO</th><th>CODIGO</th><th>DATOS</th><th>PILA</th></tr></thead><tbody><tr><td>A</td><td>16384</td><td>8700</td><td>8192</td></tr><tr><td>B</td><td>2048</td><td>1000</td><td>1024</td></tr><tr><td>C</td><td>4096</td><td>2272</td><td>2048</td></tr></tbody></table> <p>Además, se sabe que se dispone de una memoria física de 16 Kbytes y que el espacio de direcciones del sistema es de 64 Kbytes.</p> <p>Se pide:</p> <ol style="list-style-type: none"><li>Determinar si son viables tamaños de página de 1024 bytes y 512 bytes. (1 punto)</li><li>En el supuesto de que ambos tamaños de página sean posibles, justificar qué tamaño debería utilizar el sistema de paginación si se tiene en cuenta la fragmentación interna</li></ol>	PROCESO	CODIGO	DATOS	PILA	A	16384	8700	8192	B	2048	1000	1024	C	4096	2272	2048		
PROCESO	CODIGO	DATOS	PILA															
A	16384	8700	8192															
B	2048	1000	1024															
C	4096	2272	2048															

# Lecturas recomendadas

---

## Base



1. Carretero 2007:
  1. Cap.4

## Recomendada



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

# Contenidos

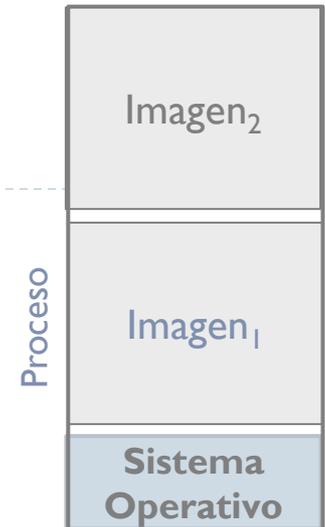
---

## 1. Introducción

- a. Punteros en C
- b. Memoria en un proceso

## 2. Gestor de memoria

## 3. Interfaz en espacio de usuario a llamadas al sistema: gestión de la memoria dinámica en libc



# Contenidos

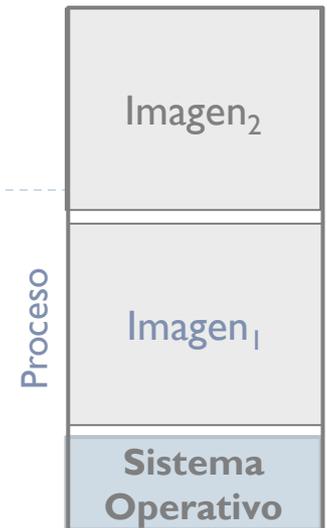
---

## 1. Introducción

- a. Punteros en C
- b. Memoria en un proceso

## 2. Gestor de memoria

## 3. Interfaz en espacio de usuario a llamadas al sistema: gestión de la memoria dinámica en libc



# Introducción

---

- ▶ Modelo



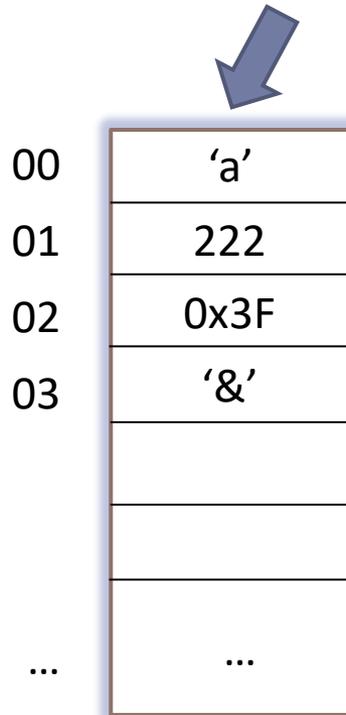
- ▶ Definiciones



# Uso básico de la memoria

## dirección, valor y tamaño

---

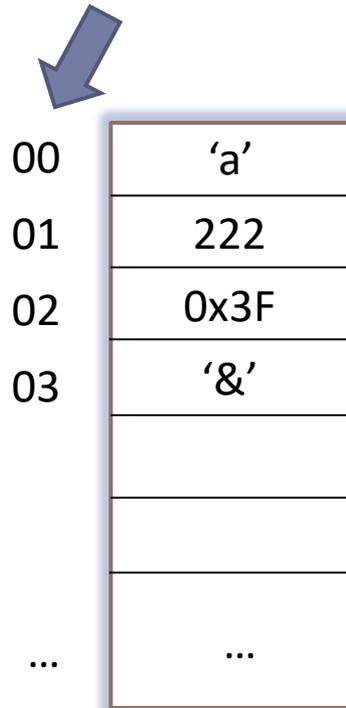


- ▶ **Valor**
  - ▶ Elemento guardado en memoria.

# Uso básico de la memoria

## dirección, valor y tamaño

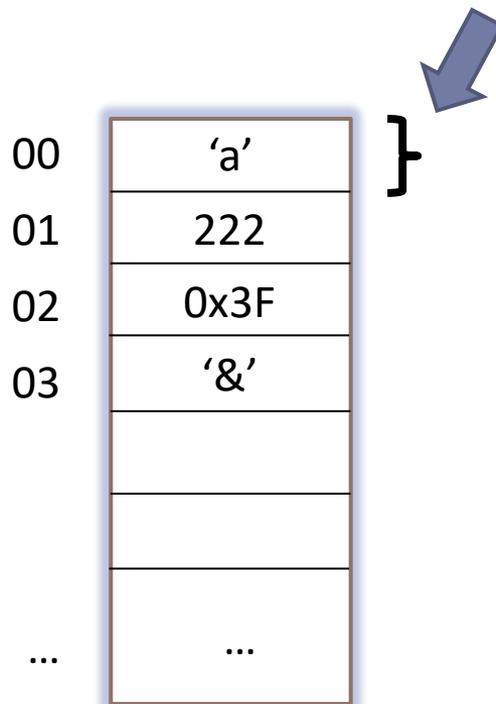
---



- ▶ **Valor**
  - ▶ Elemento guardado en memoria.
- ▶ **Dirección**
  - ▶ Posición de memoria.

# Uso básico de la memoria

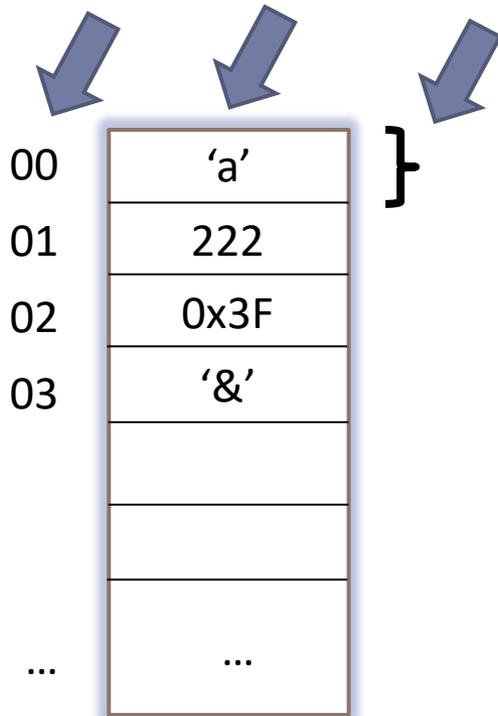
## dirección, valor y tamaño



- ▶ **Valor**
  - ▶ Elemento guardado en memoria.
- ▶ **Dirección**
  - ▶ Posición de memoria.
- ▶ **Tamaño**
  - ▶ Número de bytes necesarios para almacenar el valor.

# Uso básico de la memoria

## dirección, valor y tamaño



### ▶ Valor

- ▶ Elemento guardado en memoria a partir de una dirección, y que ocupa un cierto tamaño para ser almacenada.

### ▶ Dirección

- ▶ Número que identifica la posición de memoria (celda) a partir de la cual se almacena el valor de un cierto tamaño.

### ▶ Tamaño

- ▶ Número de bytes necesarios a partir de la dirección de comienzo para almacenar el valor.

# Valor, dirección, tamaño en C: ejemplo 1

```
#include <stdio.h>

int main ( int argc,
           char *argv[] )
{
    int i = 3;

    printf("%d ", i);
    printf("%d ", &i);
    printf("%d", sizeof(i));

    return 0;
}
```

- ▶ en definición: **int i ;**
  - ▶ Variable i de tipo entero
- ▶ en uso: **<var>**
  - ▶ Valor de...
- ▶ en uso: **& <var>**
  - ▶ Dirección de...
- ▶ en uso: **sizeof(<var>)**
  - ▶ Tamaño en bytes de...

# Valor, dirección, tamaño en C: ejemplo 2

```
#include <stdio.h>

int main ( int argc,
          char *argv[] )
{
    int i = 3;
    int *pi = &i;

    printf("%ld ", pi);
    printf("%d ", *pi);
    printf("%d", sizeof(*pi));
    return 0;
}
```

- ▶ en definición: **int \*pi;**
  - ▶ Puntero a entero
- ▶ en uso: **<var>**
  - ▶ Dirección de...
- ▶ en uso: **\* <exp>**
  - ▶ Valor contenido en...
- ▶ en uso: **sizeof(<tipo>)**
  - ▶ Tamaño en bytes de...

# Uso básico de la memoria interfaz funcional



00	'a'
01	222
02	0x3F
03	'&'
...	...

- ▶ **valor = mem\_read (dirección)**
- ▶ **mem\_write (dirección, valor)**

Antes de acceder a una dirección, tiene que apuntar a una zona de memoria previamente reservada (en tiempo de compilación o en tiempo de ejecución).

# Escritura de variable en C: ejemplo 1

```
#include <stdio.h>

void imprimir ( int val ){
    printf("v:%d\n", val) ;
}

int main ( int argc,
          char *argv[] )
{
    int i = 3;
    imprimir(i) ;
    return 0;
}
```

► en asignación: **int i = 3 ;**

► Asignar valor...

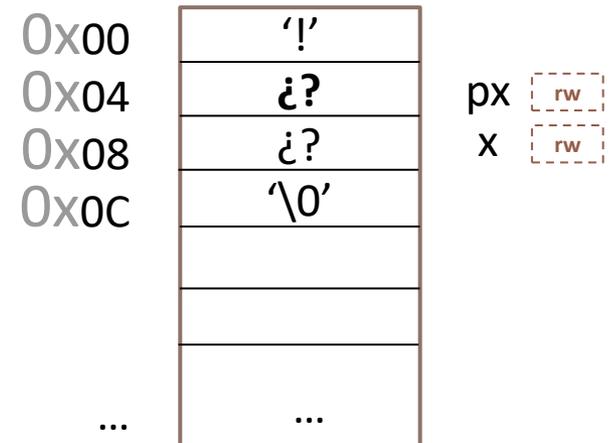
► en paso de parámetros: **<var>**

► Asignar valor de...

# Dirección y contenido en C: ejemplo 2

---

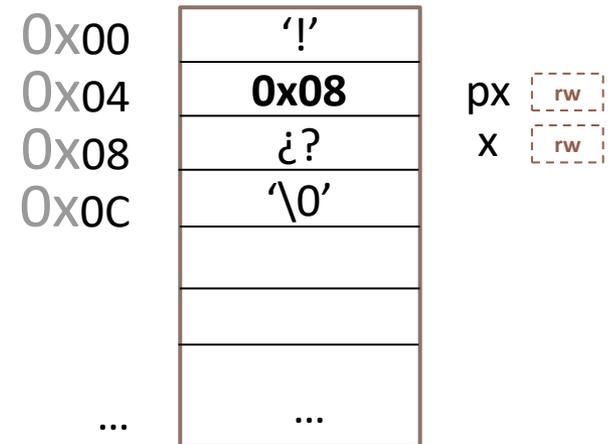
- Dos variables
  - `int *px;`
  - `int x;`



# Dirección y contenido en C: ejemplo 2

---

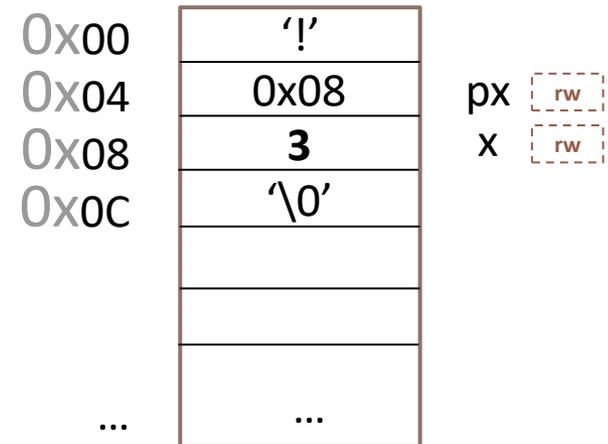
- Dos variables
  - `int *px;`
  - `int x;`
- **`px = &x;`**



# Dirección y contenido en C: ejemplo 2

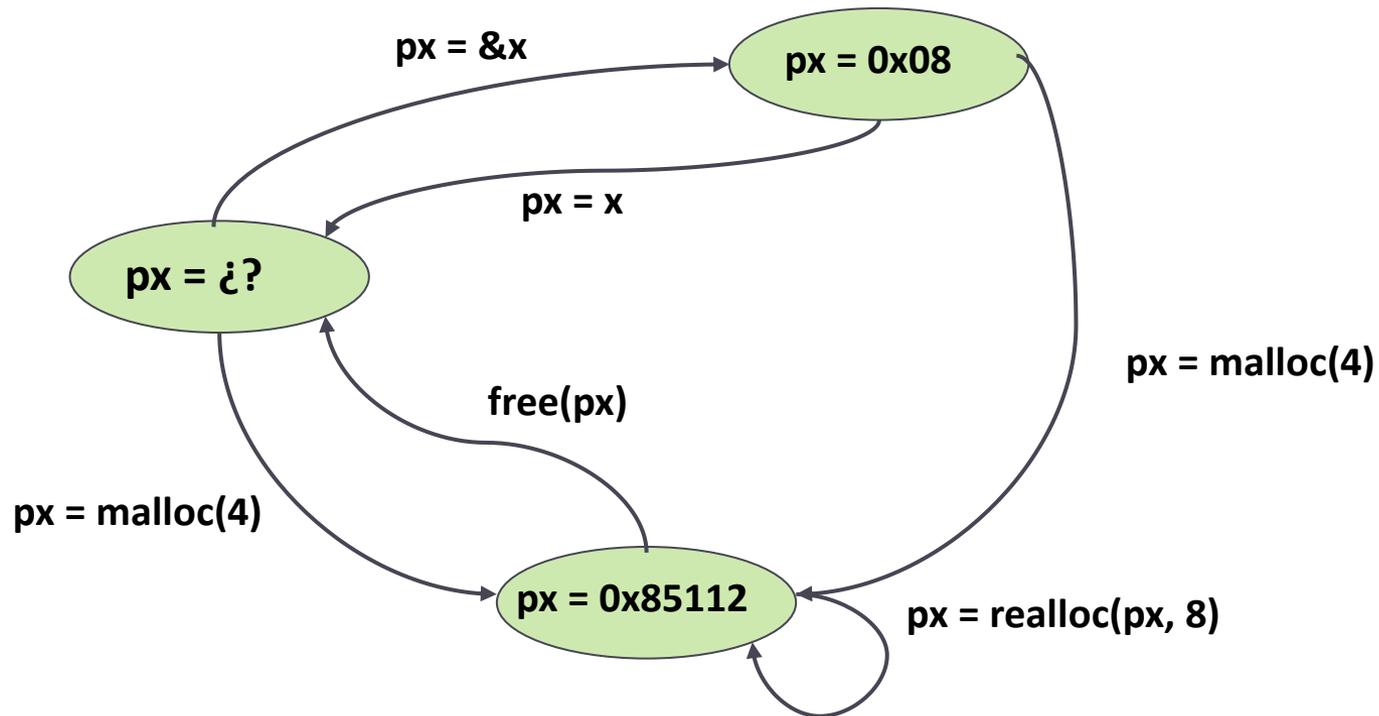
---

- Dos variables
  - `int *px;`
  - `int x;`
- `px = &x;`
- `*px = 3;`



# Por defecto, TÚ tienes que llevar la contabilidad...

---



# Contabilidad errónea -> SIGSEGV

- Dos variables

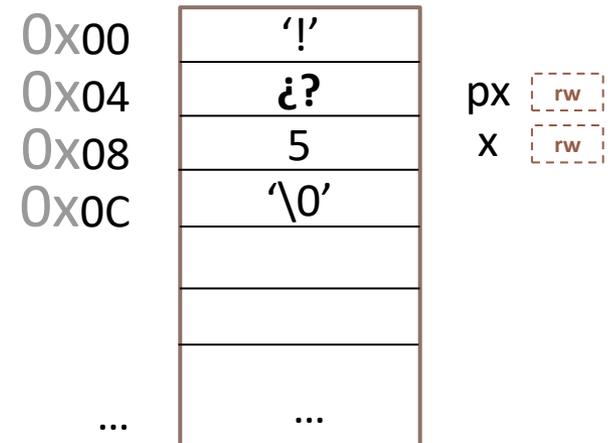
- `int *px;`
- `int x;`

- **\*px = x;**

- **MAL:** Si no se ha reservado memoria.

```
int *px;  
*px = 5;
```

- **BIEN:** Si se ha reservado previamente.



# Contabilidad errónea -> SIGSEGV

---

## ❑ Gestor de memoria apropiado:

- ❑ libc malloc
- ❑ dlmalloc, jemalloc, hoard, etc.
- ❑ Electric Fence:  
[https://elinux.org/Electric\\_Fence](https://elinux.org/Electric_Fence)

## ❑ Herramientas de asistencia apropiadas:

- ❑ valgrind
- ❑ gdb, etc.

**CP1**

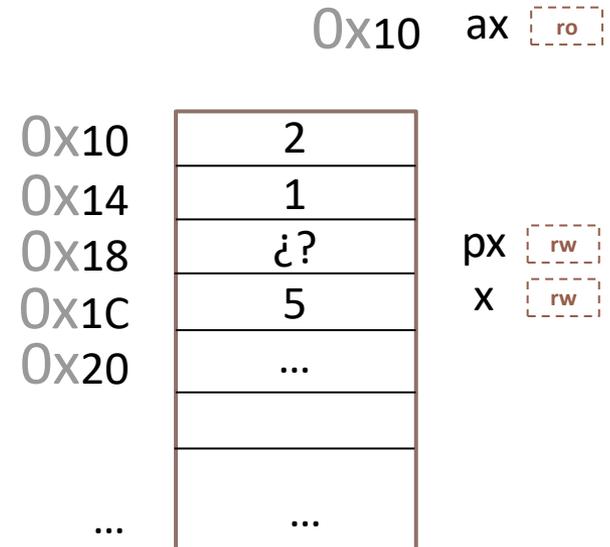
# dirección y contenido en C: array vs pointers

## ▶ Tres variables

▶ `int ax[2]`

▶ `int *px;`

▶ `int x = 5;`



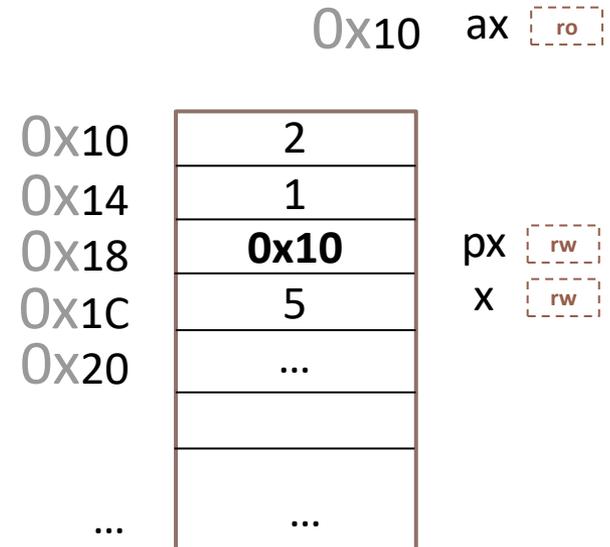
# dirección y contenido en C: array vs pointers

## ▶ Tres variables

```
▶ int ax[2]
```

```
▶ int *px;
```

```
▶ int x = 5;
```



```
▶ px = ax; // ax == &ax
```

```
▶ ax = px; // ERROR
```

```
▶ px[1] = 1; // ax[1] = 1;
```

```
▶ *px = 2; // *ax = 2;
```

# dirección y contenido en C: array vs pointers

## ▶ Tres variables

▶ `int ax[2]`

▶ `int *px;`

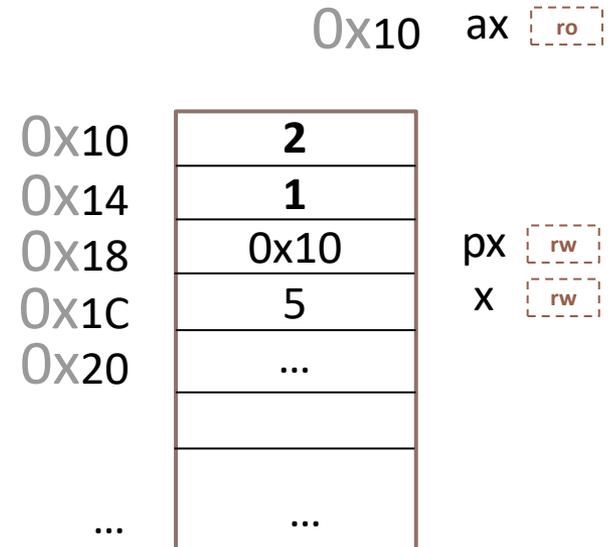
▶ `int x = 5;`

▶ `px = ax; // ax == &ax`

▶ `ax = px; // ERROR`

▶ `px[1] = 1; // ax[1] = 1;`

▶ `*px = 2; // *ax = 2;`



# Escritura de variable en C: ejemplo 1

```
#include <stdio.h>

void imprimir ( int val ) {
    printf("v:%d\n", val) ;
}

int main ( int argc,
           char *argv[] )
{
    int i = 3;
    imprimir(i) ;
    return 0;
}
```

► en asignación: **int i = 3 ;**

► Asignar valor...

► en paso de parámetros: **<var>**

► Asignar valor de...

# Paso de parámetros

---

```
void prueba2 ( int j, char c, float f, int pj )  
{  
    /* ... */  
}
```

# Paso de parámetros

---

```
i = 10 ;  
float PI = 3.14 ;  
  
prueba2 ( i,      'a',      PI,      &i ) ;  
...
```

```
void prueba2 ( int j, char c, float f, int pj )  
{  
    /* ... */  
}
```

# Paso de parámetros

---

```
i = 10 ;  
float PI = 3.14 ;  
  
prueba2 ( i,      'a',      PI,      &i ) ;  
...
```

```
void prueba2 (     )  
{  
    /* ... */  
}
```

**1) Se crea en pila las variables formales (j, c, f, pj)**

# Paso de parámetros

---

```
i = 10 ;  
float PI = 3.14 ;
```

```
prueba2 ( i, 'a', PI, &i ) ;  
...
```

```
void prueba2 ( int j, char c, float f, int pj )  
{  
    /* ... */  
}
```

**2) Se copia el valor de los parámetros reales**

# Paso de parámetros

```
i = 10 ;  
float PI = 3.14 ;
```

```
prueba2 ( i, 'a', PI, &i ) ;  
...
```

10    'a'    3.14    0x

```
void prueba2 ( int j, char c, float f, int pj )  
{  
    /* ... */  
}
```



Siempre se realiza una copia de los parámetros

# Paso de parámetros

---

## Paso de parámetro por **valor**

```
#include <stdio.h>

int main (void)
{
    int i = 10;

    /* ... */
    inc(i) ;
    /* ... */
}
```

```
void inc ( int j )
{
    j = j + 1 ;
}
```

# Paso de parámetros

---

## Paso de parámetro por **valor**

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int i = 10;
```

```
    /* ... */
```

```
    inc(i) ;
```

```
    /* ... */
```

```
}
```

1) se copia  
i en j

10

```
void inc ( int j )
```

```
{
```

```
    j = j + 1 ;
```

```
}
```

# Paso de parámetros

## Paso de parámetro por **valor**

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int i = 10;
```

```
    /* ... */
```

```
    inc(i) ;
```

```
    /* ... */
```

```
}
```

1) se copia  
i en j

10

```
void inc ( int j )
```

```
{
```

```
    j = j + 1 ;
```

```
}
```

2) se modifica  
j (la copia)



# Paso de parámetros

Paso de parámetro  
por **valor**

```
#include <stdio.h>

int main (void)
{
    int i = 10;

    /* ... */
    inc(i) ;
    /* ... */
}
```

1) se copia  
i en j

10

```
void inc ( int j )
{
    j = j + 1 ;
}
```

2) se modifica  
j (la copia)

Paso de parámetro  
por **referencia**



# Paso de parámetros

## Paso de parámetro por **valor**

```
#include <stdio.h>

int main (void)
{
    int i = 10;

    /* ... */
    inc(i) ;
    /* ... */
}
```

1) se copia  
i en j

```
void inc ( int j )
{
    j = j + 1 ;
}
```

2) se modifica  
j (la copia)

## Paso de parámetro por **referencia**

```
#include <stdio.h>

int main (void)
{
    int i = 3;

    /* ... */
    inc(&i) ;
    /* ... */
}
```

```
void inc ( int *j )
{
    *j = *j + 1 ;
}
```

# Paso de parámetros

## Paso de parámetro por **valor**

```
#include <stdio.h>

int main (void)
{
    int i = 10;

    /* ... */
    inc(i);
    /* ... */
}

void inc ( int j )
{
    j = j + 1 ;
}
```

1) se copia i en j

2) se modifica j (la copia)

## Paso de parámetro por **referencia**

```
#include <stdio.h>

int main (void)
{
    int i = 3;

    /* ... */
    inc(&i);
    /* ... */
}

void inc ( int *j )
{
    *j = *j + 1 ;
}
```

1) se copia &i en j

# Paso de parámetros

## Paso de parámetro por **valor**

```
#include <stdio.h>

int main (void)
{
    int i = 10;

    /* ... */
    inc(i);
    /* ... */
}
```

1) se copia  
i en j

```
void inc ( int j )
{
    j = j + 1 ;
}
```

2) se modifica  
j (la copia)

## Paso de parámetro por **referencia**

```
#include <stdio.h>

int main (void)
{
    int i = 3;

    /* ... */
    inc(&i);
    /* ... */
}
```

1) se copia  
&i en j

```
void inc ( int *j )
{
    *j = *j + 1 ;
}
```

2) se modifica  
i a través de \*j

# Paso de parámetros: ejemplo

---

```
#include <stdio.h>

void inc ( int *j )
{
    *j = *j + 1 ;
}

int main (void)
{
    int i = 3;

    inc(&i) ;
    printf("%d\n", i) ;

    return 0;
}
```

- ▶ La función *inc* incrementa el valor pasado por referencia en *j*
- ▶ La función *main* define una variable *i*, incrementa su valor y lo imprime

# Paso de parámetros: ejemplo

```
#include <stdio.h>

void inc ( int *j )
{
    *j = *j + 1 ;
}

int main (void)
{
    int i = 3;

    inc(&i) ;
    printf("%d\n",i) ;

    return 0;
}
```



▶ **gcc -Wall -g -o el el.c**

- ▶ **Wall:**  
mostrar advertencias
- ▶ **g:**  
añadir información de depuración
- ▶ **o:**  
establecer el nombre del ejecutable



▶ **./el**

- ▶ **El directorio actual (.) no está en la variable PATH**

# Paso de parámetros: ejemplo

```
#include <stdio.h>

void inc ( int *j )
{
    *j = *j + 1 ;
}

int main (void)
{
    int i = 3;
    inc( i) ;
    printf("%d\n",i) ;

    return 0;
}
```



▶ **gcc -Wall -g -o e2 e2.c**

- ▶ **Wall:**  
mostrar advertencias
- ▶ **g:**  
añadir información de depuración
- ▶ **o:**  
establecer el nombre del ejecutable



▶ **./e2**

- ▶ **El directorio actual (.) no está en la variable PATH**

# Paso de parámetros: ejemplo

---



```
acaldero@phoenix:/tmp$ ./e13
Violación de segmento
acaldero@phoenix:/tmp$ gdb e13
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2
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 símbolos desde /tmp/e13...hecho.
(gdb) run
Starting program: /tmp/e13

Program received signal SIGSEGV, Segmentation fault.
0x080483ca in inc (j=0x3) at e13.c:5
5          *j = *j + 1 ;
(gdb) █
```

# Contenidos

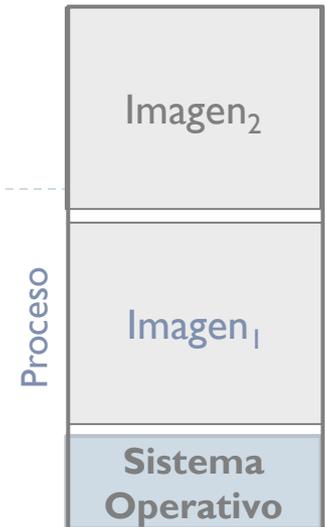
---

## 1. Introducción

- a. Punteros en C
- b. **Memoria en un proceso**

## 2. Gestor de memoria

## 3. Interfaz en espacio de usuario a llamadas al sistema: gestión de la memoria dinámica en libc



# Introducción

---

- ▶ Modelo

Dirección

Tamaño

Valor

- ▶ Definiciones

Programa

Imagen de proceso

Proceso

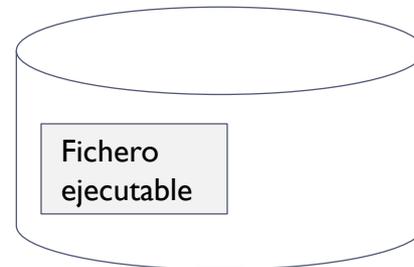


# Programa y proceso

---

- ▶ **Programa:** conjunto de datos e instrucciones ordenadas que permiten realizar una tarea o trabajo específico.

Disco

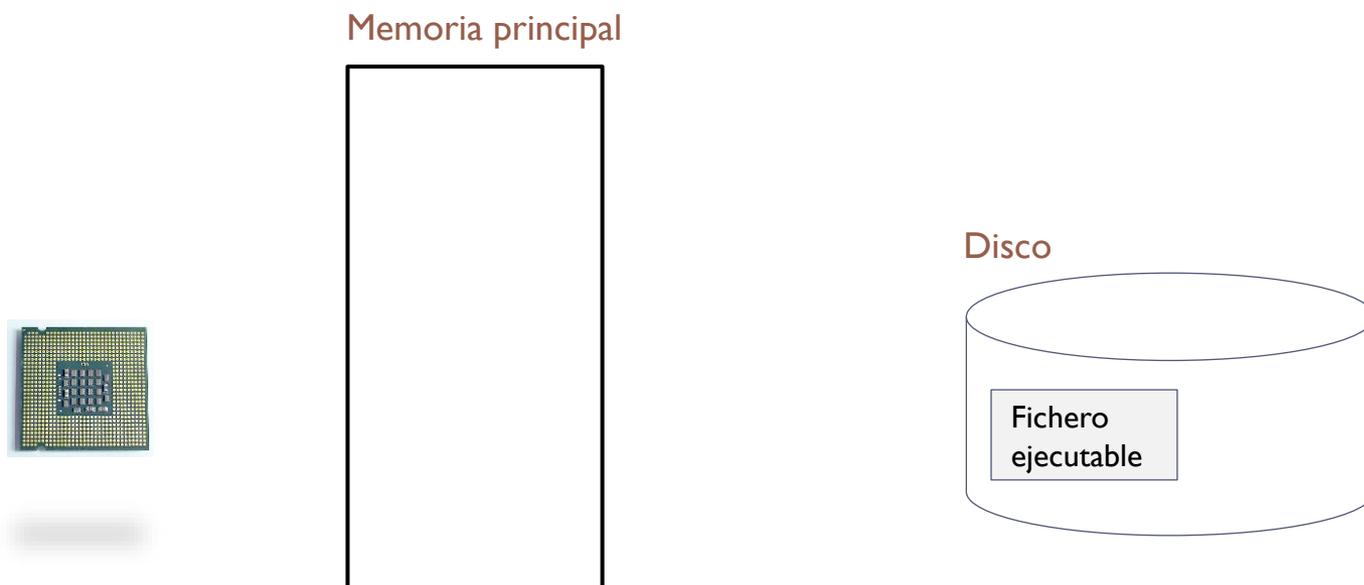




# Programa y proceso

---

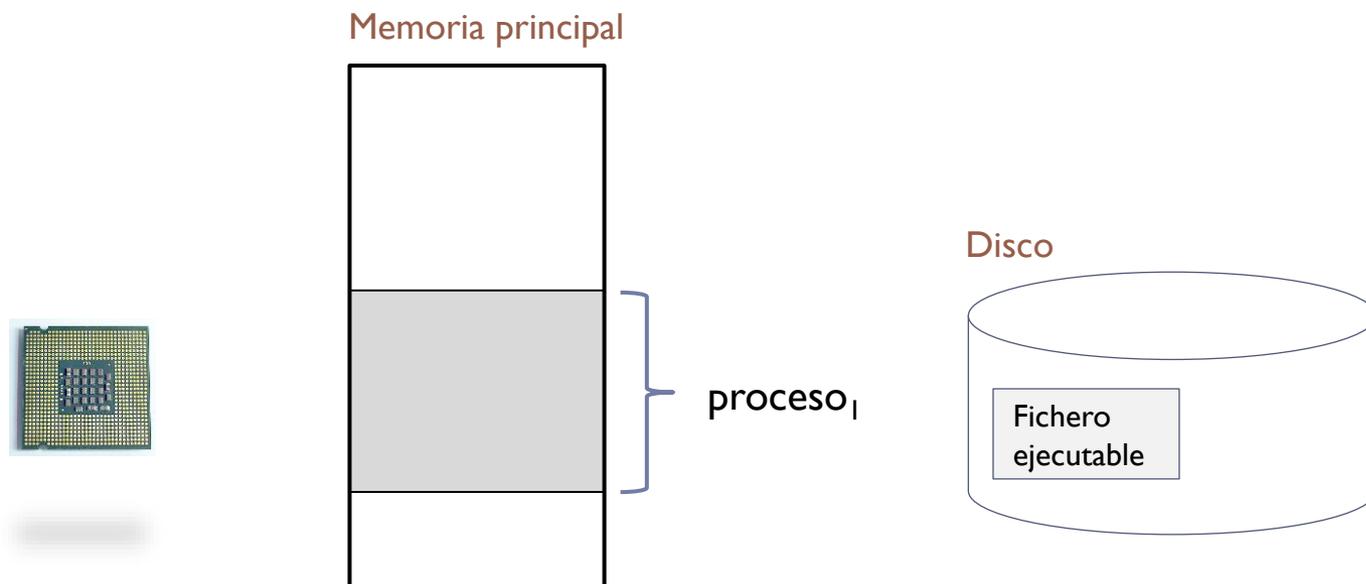
- ▶ **Programa:** conjunto de datos e instrucciones ordenadas que permiten realizar una tarea o trabajo específico.
  - ▶ Para su ejecución, ha de estar en memoria.





# Programa y proceso

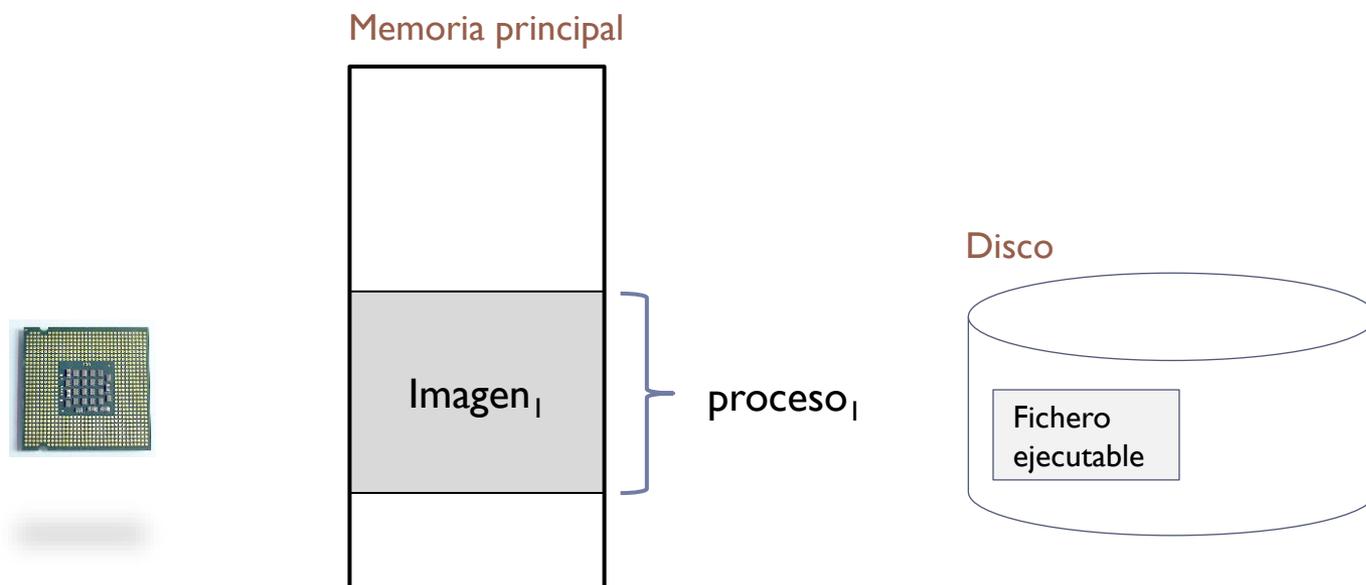
- ▶ **Proceso:** programa en ejecución.





# Imagen de un proceso

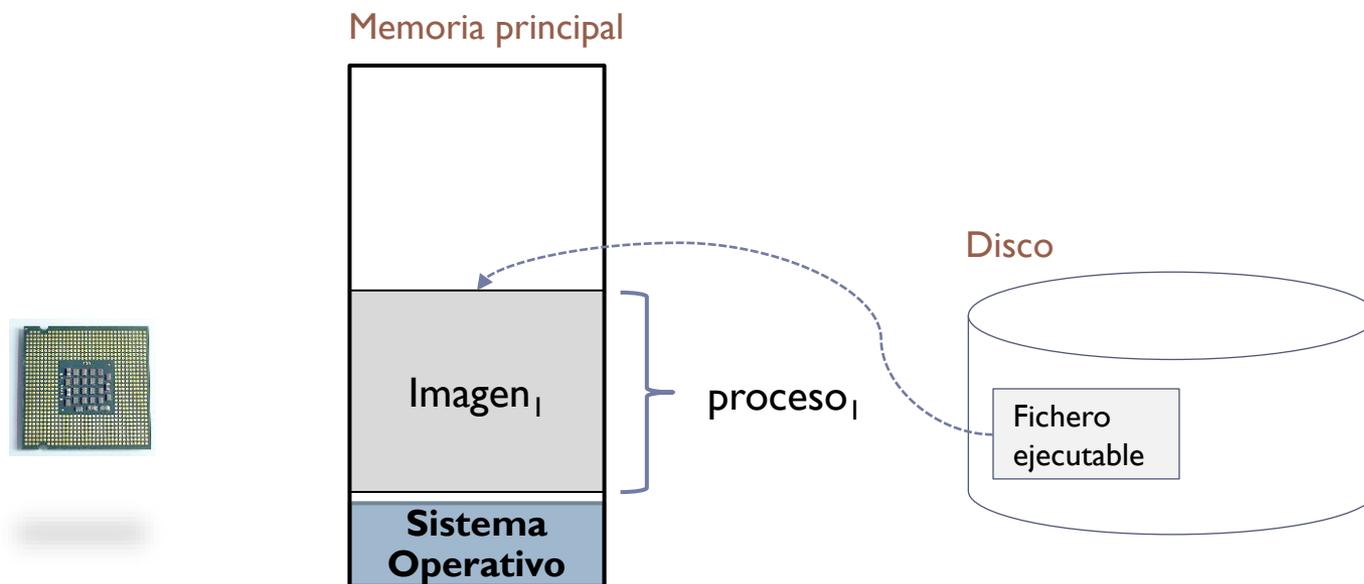
- ▶ **Imagen de memoria:** conjunto de direcciones de memoria asignadas al programa que se está ejecutando (y contenido).





# El sistema operativo...

- ▶ El sistema operativo se encargará en la gestión de la memoria de:
  - ▶ **Crear** un proceso a partir de la información del ejecutable.
  - ▶ El sistema operativo es un proceso que asiste a otros procesos y al hardware según lo indicado por el usuario



# Introducción

## resumen

---

- ▶ Modelo

Dirección

Tamaño

Valor

- ▶ Definiciones

Programa

Imagen de proceso

Proceso

# Introducción

## resumen

---

- ▶ Modelo

Dirección

Tamaño

Valor

- ▶ Definiciones

Programa

Imagen de proceso

Proceso

Regiones principales  
de memoria

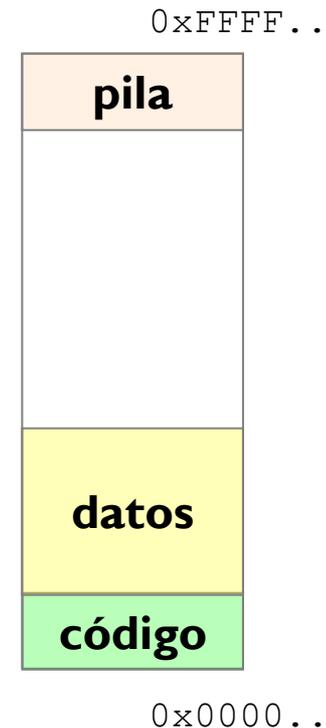


# Organización lógica (de los programas)

## modelo de memoria de un proceso

---

- ▶ Un proceso está formado por una serie de regiones.
- ▶ Una **región** es una **zona contigua** del espacio de direcciones de un **proceso con las mismas propiedades**.
- ▶ Principales propiedades:
  - ▶ Permisos: lectura, escritura y ejecución.
  - ▶ Compartición entre hilos: *private* o *shared*
  - ▶ Tamaño (fijo/variable)
  - ▶ Valor inicial (con/sin soporte)
  - ▶ Creación estática o dinámica
  - ▶ Sentido de crecimiento



# Principales regiones de un proceso

## código (*text*)

### Código

- Estático
- Se conoce en tiempo de compilación
- Secuencia de instrucciones a ser ejecutadas

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

datos estáticos  
**sin** valor inicial

datos estáticos  
**con** valor inicial

**código**

0x0000..



# Principales regiones de un proceso datos (*data*)

## Variable globales

- Estáticas
- Se crean al iniciar el programa
- Existen durante ejecución
- Dirección fija en memoria y ejecutable

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

datos estáticos  
**sin** valor inicial

datos estáticos  
**con** valor inicial

**código**

0x0000..



# Principales regiones de un proceso

## pila (*stack*)

### Variable locales y parámetros

- Dinámicas
- Se crean al invocar la función
- Se destruyen al retornar
- Recursividad: varias instancias de una variable

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

datos estáticos  
**sin** valor inicial

datos estáticos  
**con** valor inicial

**código**

0x0000..



# Principales regiones de un proceso

## pila (*stack*)

```
int a;
int b = 5;

void f(int c) {
    int d;
    static e = 2;

    b = d + 5;
    .....
    return;
}

main (int argc, char **argv) {
    char *p;
    p = (char *) malloc (1024)
    f(b)
    ..... pthread_create(f...)
    free (p)
    ....
    exit (0)
}
```



# Principales regiones de un proceso

## datos dinámicos (*heap*)

### Variable dinámicas

- Variables locales o globales sin espacio asignado en tiempo de compilación
- Se reserva (y libera) espacio en tiempo de ejecución

```
int a;
int b = 5;

void f(int c) {
    int d;
    static e = 2;

    b = d + 5;
    .....
    return;
}

main (int argc, char **argv) {
    char *p;
    p = (char *) malloc (1024)
    f(b)
    .....
    free (p)
    ....
    exit (0)
}
```

0xFFFF..

**pila**

**datos  
dinámicos**

**datos  
estáticos**

**código**

0x0000..

# Introducción

## resumen

---

- ▶ Modelo

Dirección

Tamaño

Valor

- ▶ Definiciones

Programa

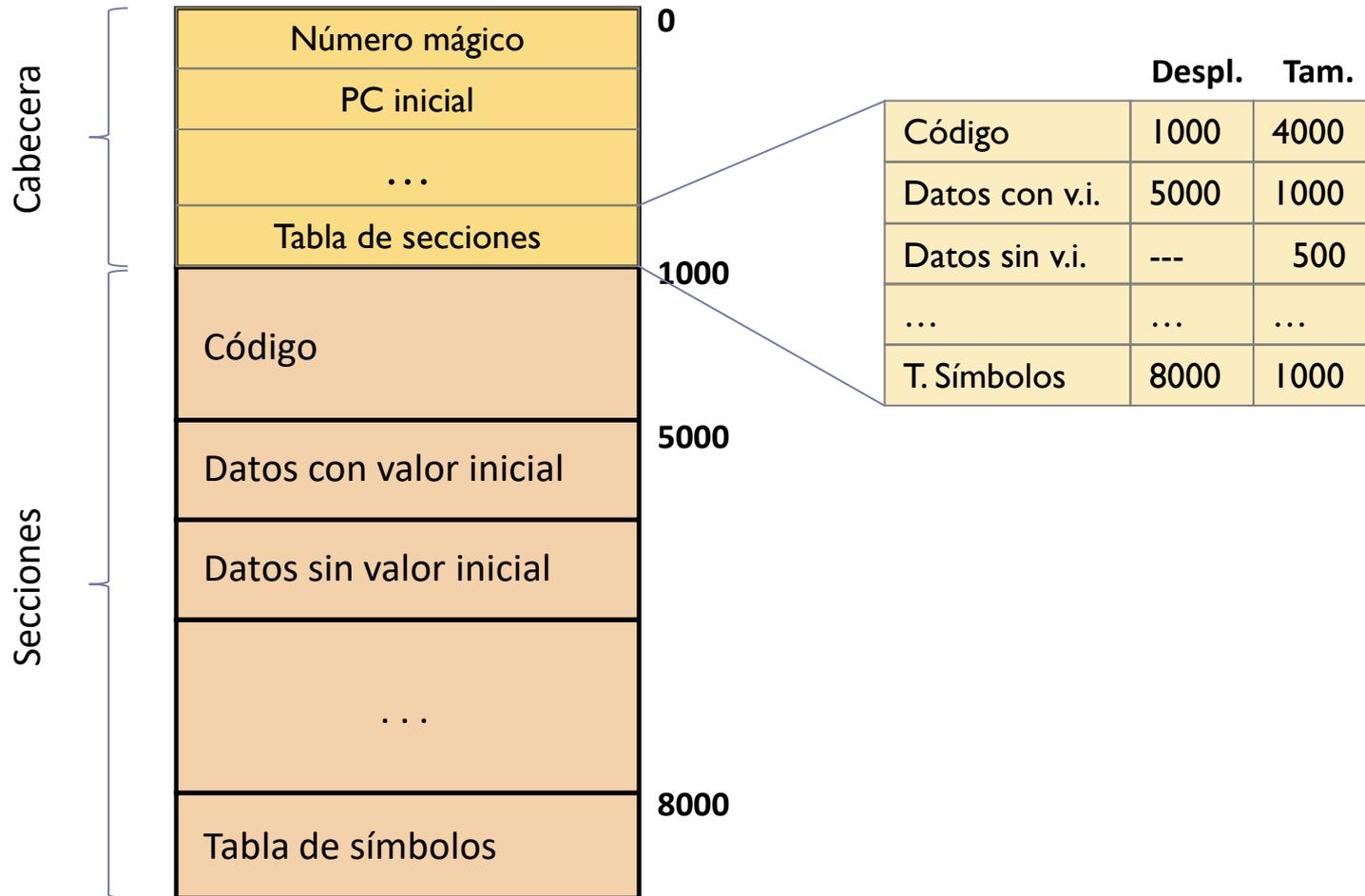
Imagen de proceso

Proceso

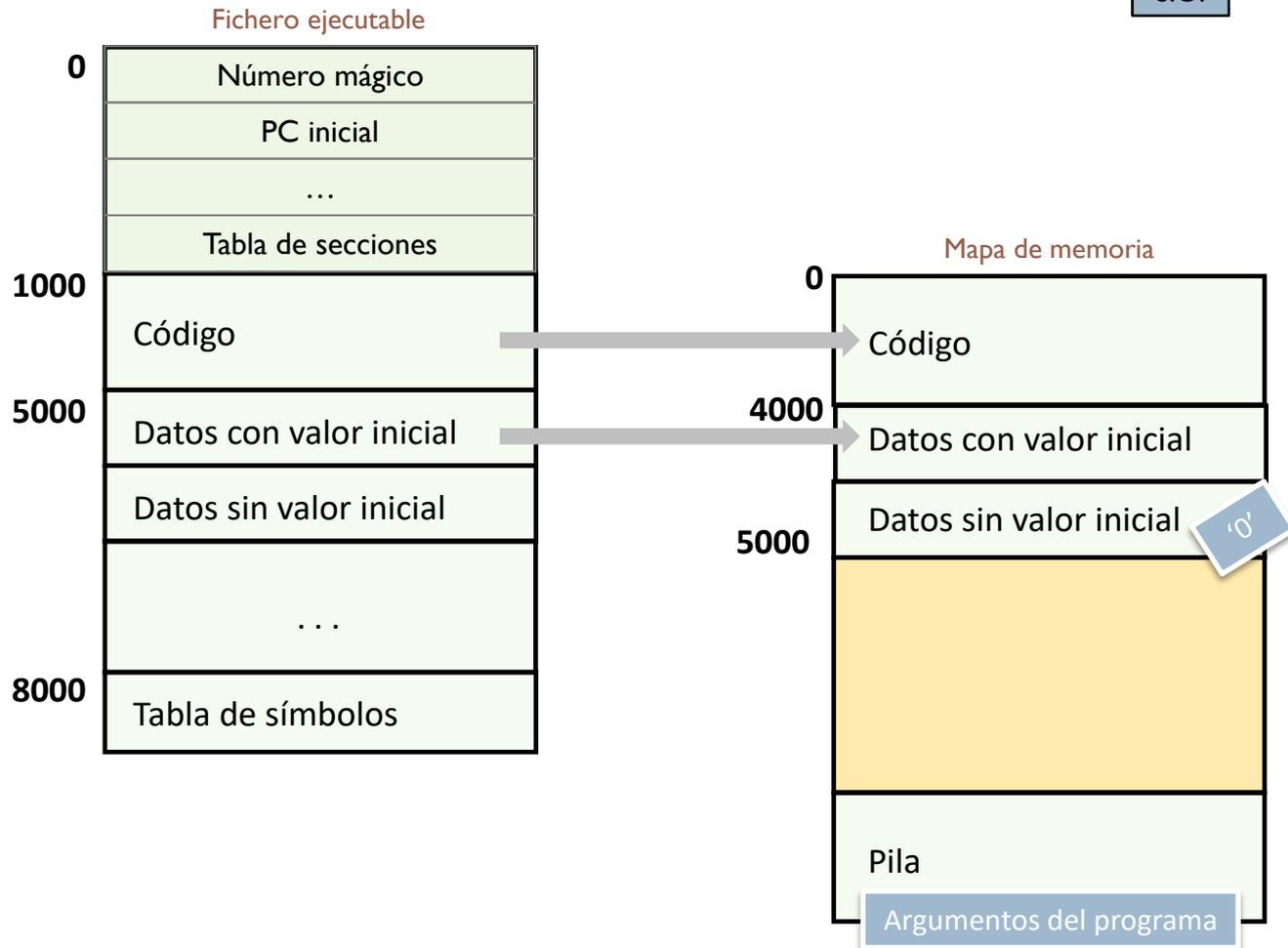
Regiones principales  
de memoria



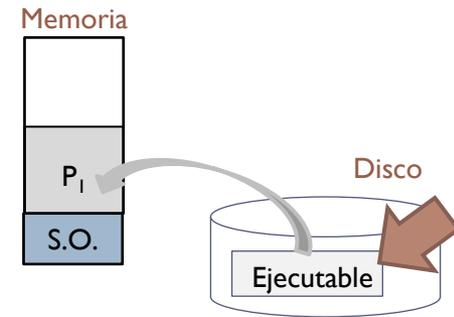
# Ejemplo de formato de ejecutable



# Crear mapa desde ejecutable



# Inspeccionar un ejecutable



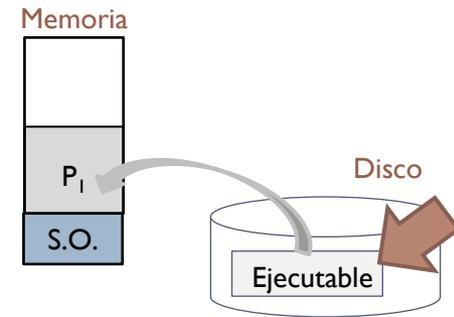
## ► Dependencias de un ejecutable (lib. dinámicas):

```
acaldero@phoenix:~/infodso/$ ldd main.exe
linux-gate.so.1 => (0xb7797000)
libdinamica.so.1 => not found
libc.so.6 => /lib/libc.so.6 (0xb761c000)
/lib/ld-linux.so.2 (0xb7798000)
```

## ► Símbolos de un ejecutable:

```
acaldero@phoenix:~/infodso/$ nm main.exe
08049f20 d __DYNAMIC
08049ff4 d __GLOBAL_OFFSET_TABLE__
0804856c R __IO_stdin_used
          w __Jv_RegisterClasses
08049f10 d __CTOR_END__
08049f0c d __CTOR_LIST__
...
```

# Inspeccionar un ejecutable



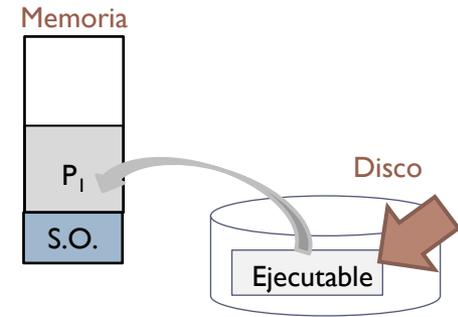
## ► Detalles de las secciones de un ejecutable:

```
acaldero@phoenix:~/infodso/$ objdump -x main.exe
...

Program Header:
...
DYNAMIC off      0x00000f20 vaddr 0x08049f20 paddr 0x08049f20 align 2**2
          filesz 0x000000d0 memsz 0x000000d0 flags rw-
...
STACK off       0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2
          filesz 0x00000000 memsz 0x00000000 flags rw-
...

Dynamic Section:
NEEDED          libdinamica.so
NEEDED          libc.so.6
INIT           0x08048368
...
```

# Inspeccionar un ejecutable



## (continuación)

Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.interp	00000013	08048134	08048134	00000134	2**0
			CONTENTS, ALLOC, LOAD, READONLY, DATA			
...						
12	.text	0000016c	080483e0	080483e0	000003e0	2**4
			CONTENTS, ALLOC, LOAD, READONLY, CODE			
...						
23	.bss	00000008	0804a014	0804a014	00001014	2**2
			ALLOC			
...						

SYMBOL TABLE:

08048134	l	d	.interp	00000000	.interp
08048148	l	d	.note.ABI-tag	00000000	.note.ABI-tag
08048168	l	d	.note.gnu.build-id	00000000	.note.gnu.build-id
...					
0804851a	g	F	.text	00000000	.hidden __i686.get_pc_thunk.bx
08048494	g	F	.text	00000014	main
08048368	g	F	.init	00000000	_init

# Contenidos

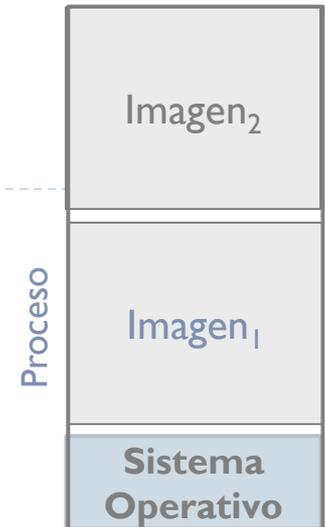
---

## 1. Introducción

- a. Punteros en C
- b. Memoria en un proceso

## 2. **Gestor de memoria**

- ## 3. Interfaz en espacio de usuario a llamadas al sistema: gestión de la memoria dinámica en libc



# Gestor de memoria (*memory allocator*)

---

*memory allocator* = Bloque



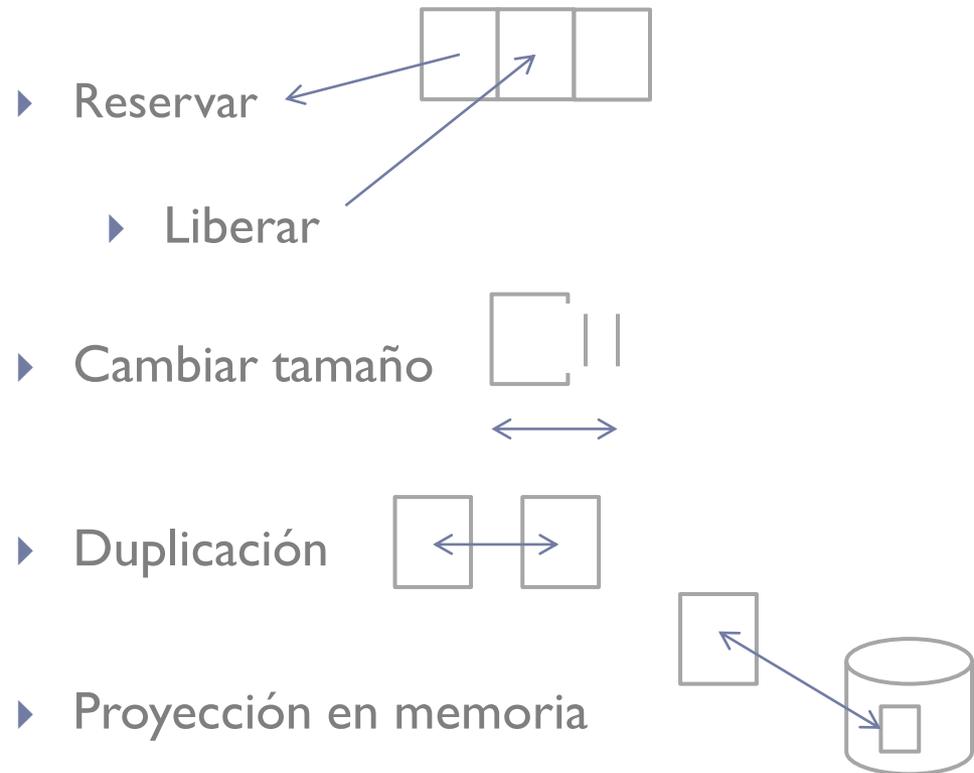
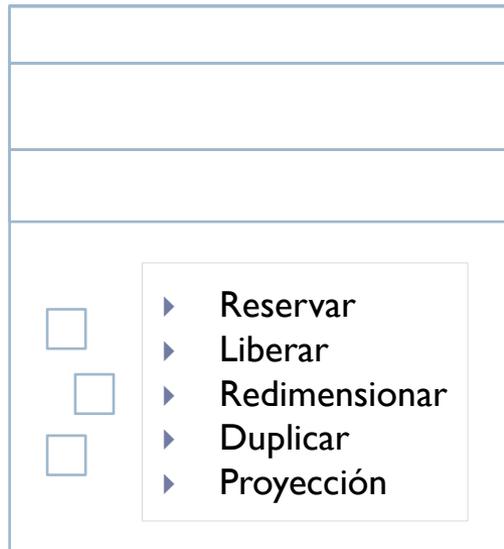
# Gestor de memoria (*memory allocator*)

*memory allocator* = Bloque + Interfaz

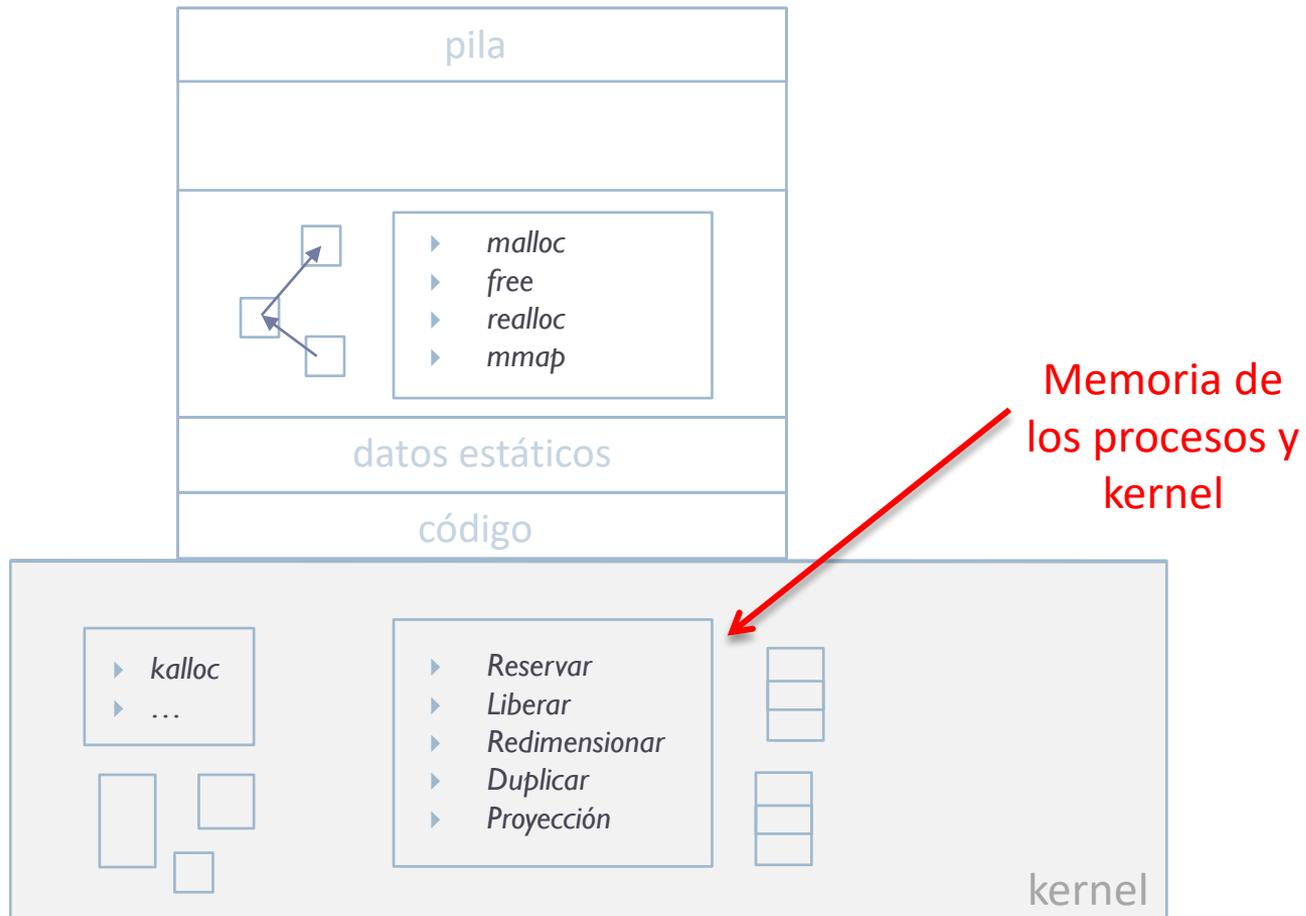


# Gestor de memoria (*memory allocator*)

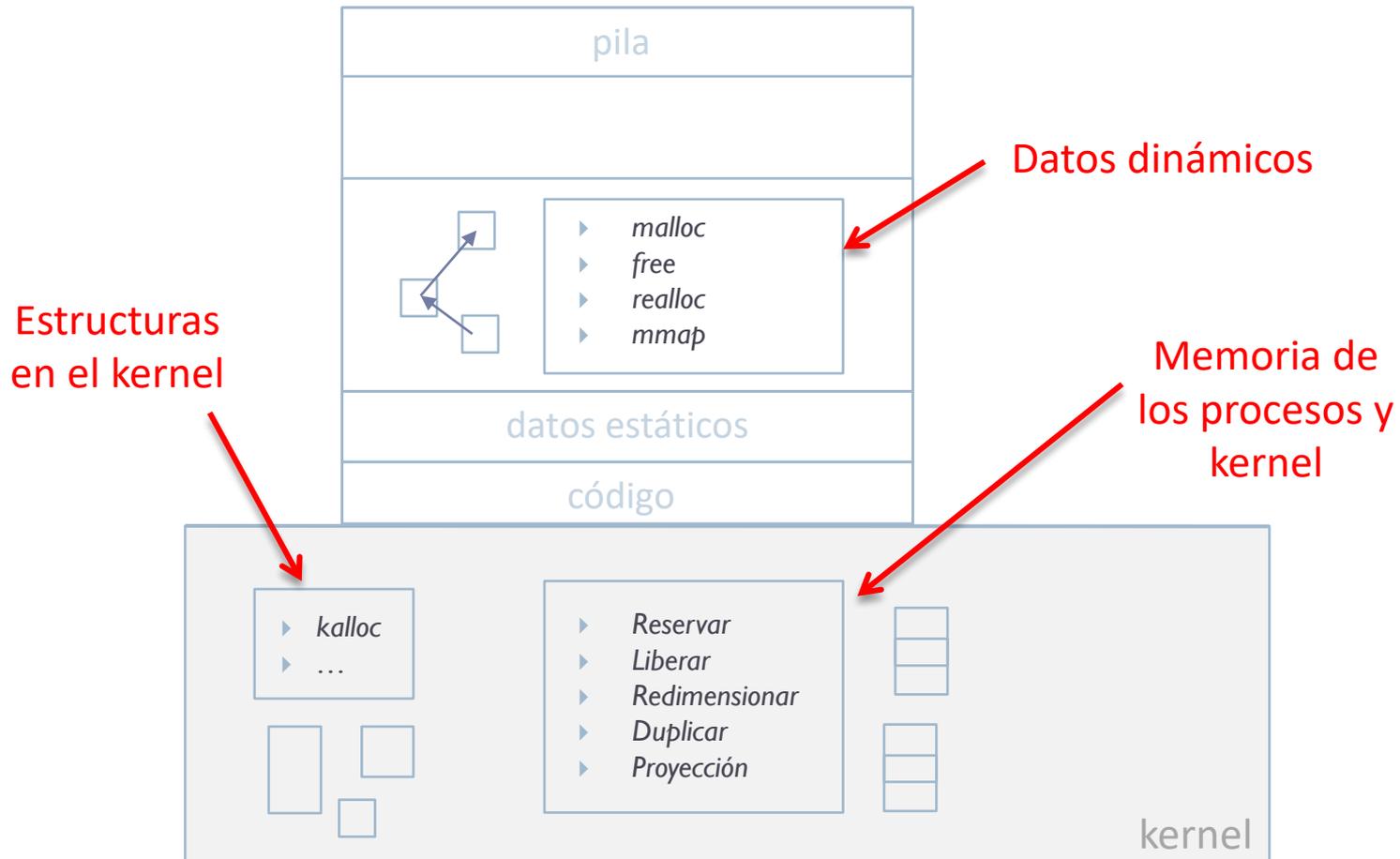
*memory allocator* = Bloque + Interfaz + Metadatos



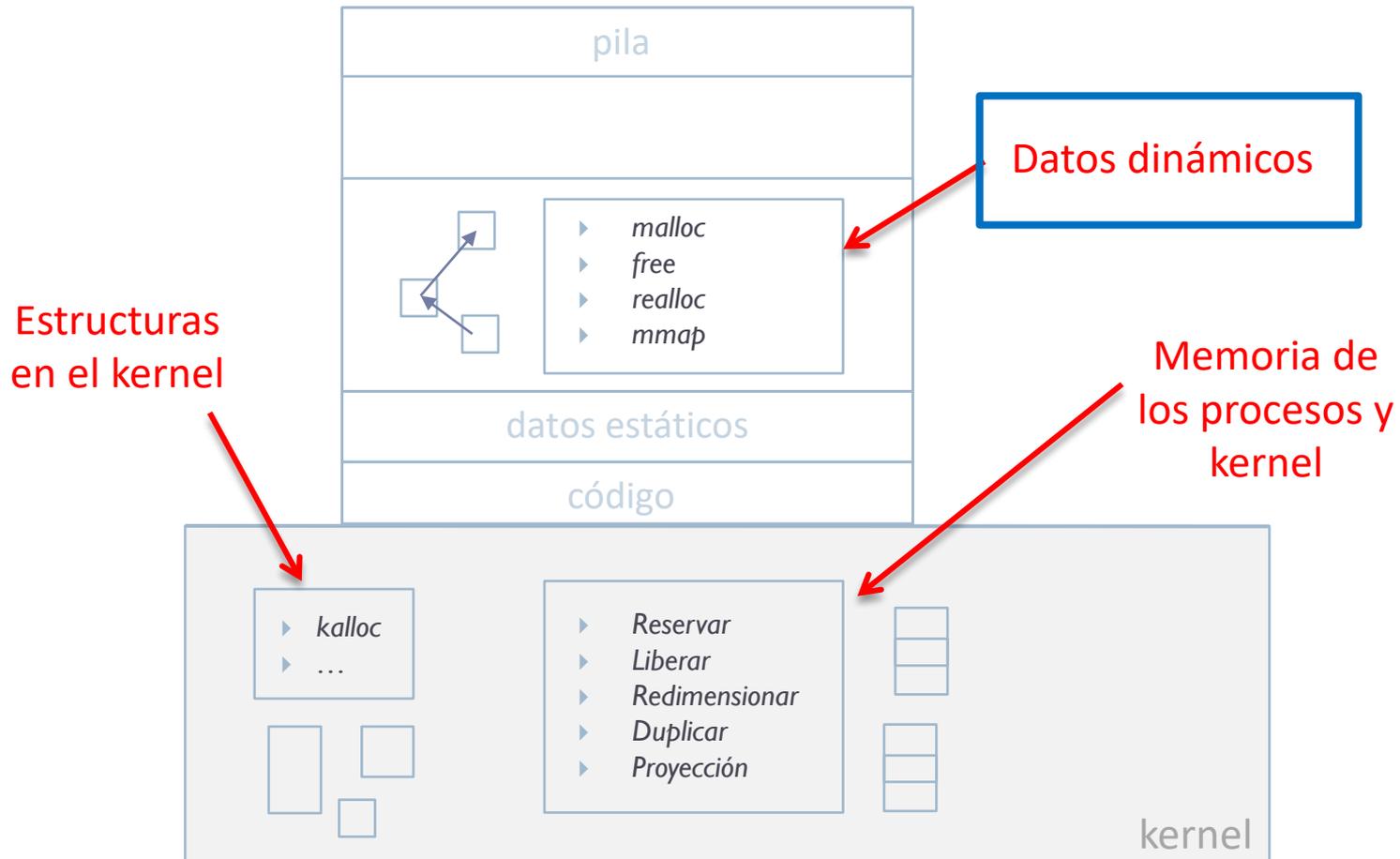
# Gestores a varios niveles: N1



# Gestores a varios niveles: N2



# Gestores a varios niveles



# Contenidos

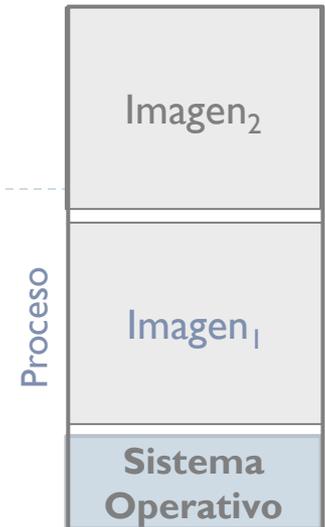
---

## 1. Introducción

- a. Punteros en C
- b. Memoria en un proceso

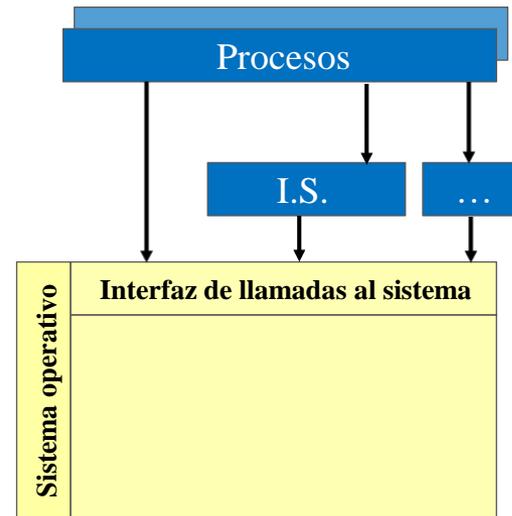
## 2. Gestor de memoria

## 3. Interfaz en espacio de usuario a llamadas al sistema: gestión de la memoria dinámica en libc



# Interfaz de servicios vs ll. al sistema

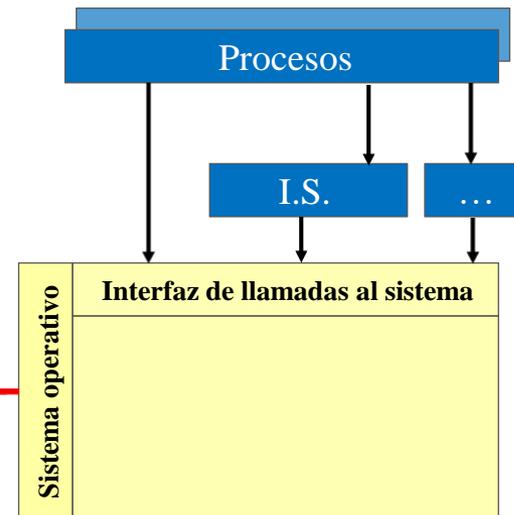
---



# Interfaz de servicios vs ll. al sistema



“servicios muy básicos de la casa”

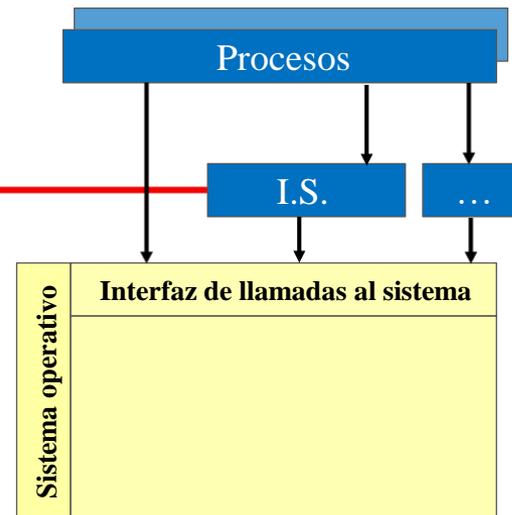


# Interfaz de servicios vs ll. al sistema

---



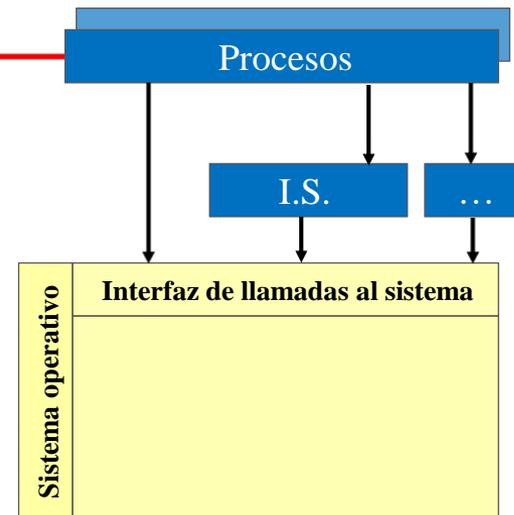
“servicios básicos de la casa”



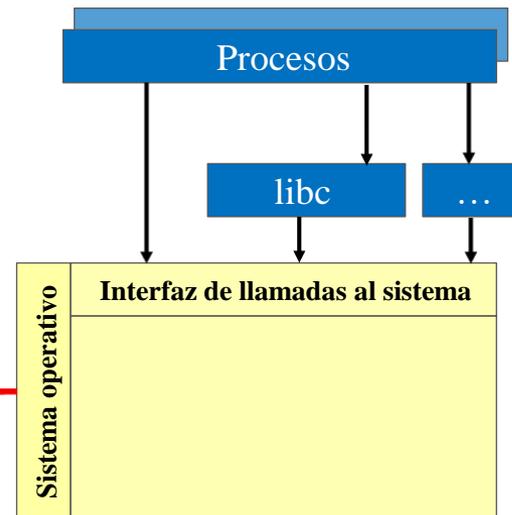
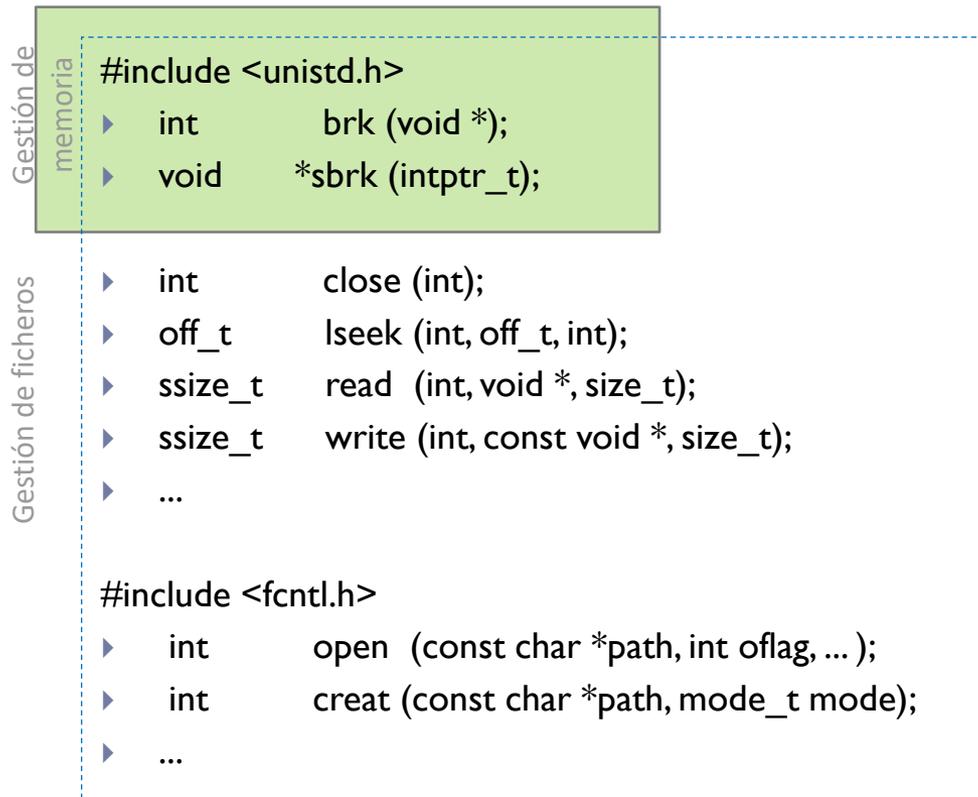
# Interfaz de servicios vs ll. al sistema



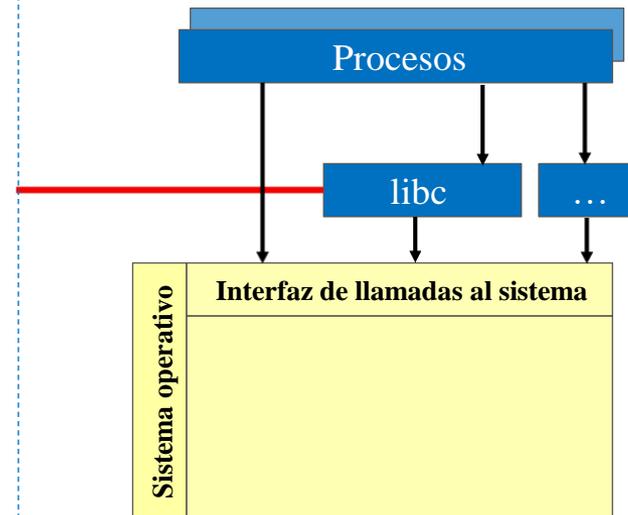
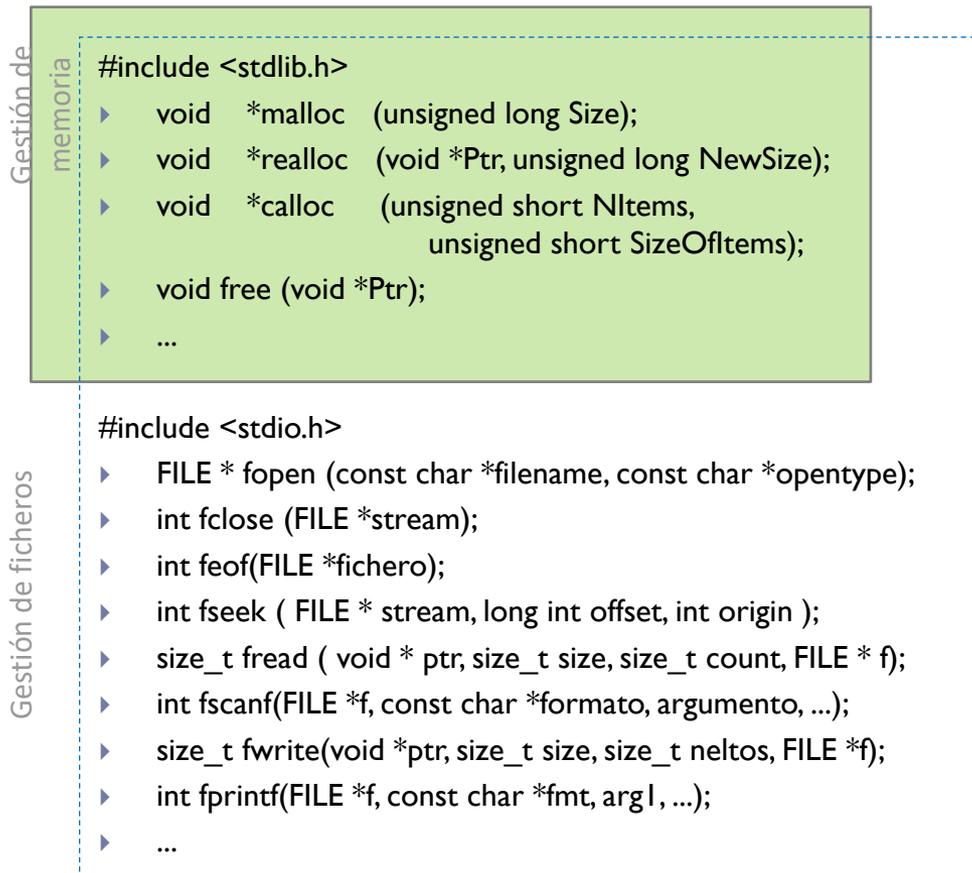
“personas que utilizan los servicios”



# Interfaz en espacio de usuario a ll. al sistema vs ll. al sistema



# Interfaz en espacio de usuario a ll. al sistema vs ll. al sistema

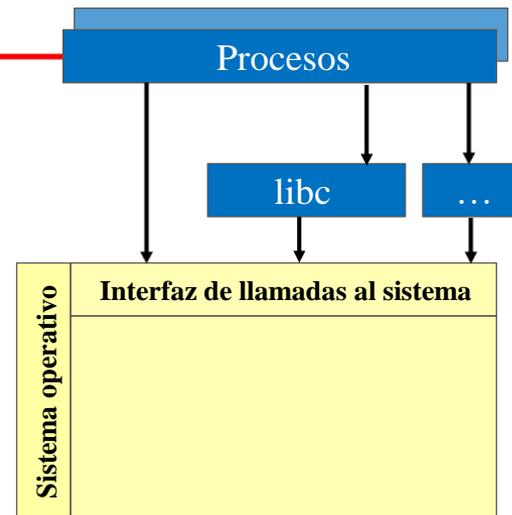


# Interfaz en espacio de usuario a ll. al sistema vs ll. al sistema

```
#include <stdlib.h>
#include <stdio.h>

int main ( int argc, char *argv[] )
{
    int *ptr1 ;
    int i ;

    ptr1 = (int *)malloc (100*sizeof(int)) ;
    for (i=0; i<100; i++)
        ptr1[i] = 10 ;
    free(ptr1);
}
```



# 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 símbolos desde /home/acaldero/work/infodso/memoria/ptr...hecho.
```

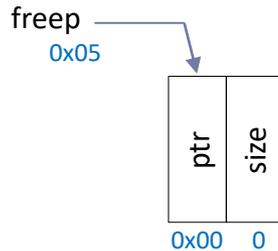
```
(gdb) run
```

```
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

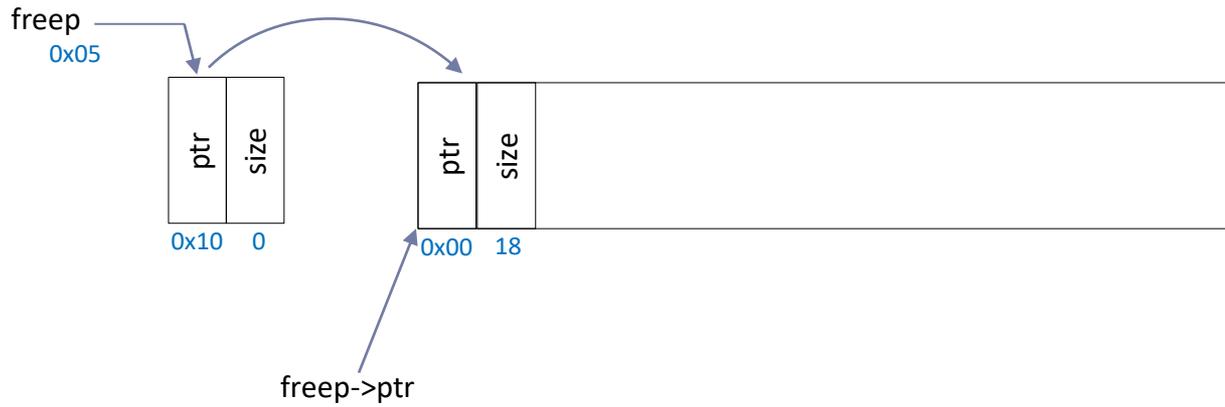


- ▶ **static Header base :**
  - ▶ Primer elemento de la lista
  - ▶ Con tamaño 0 (cabeceras)

```
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



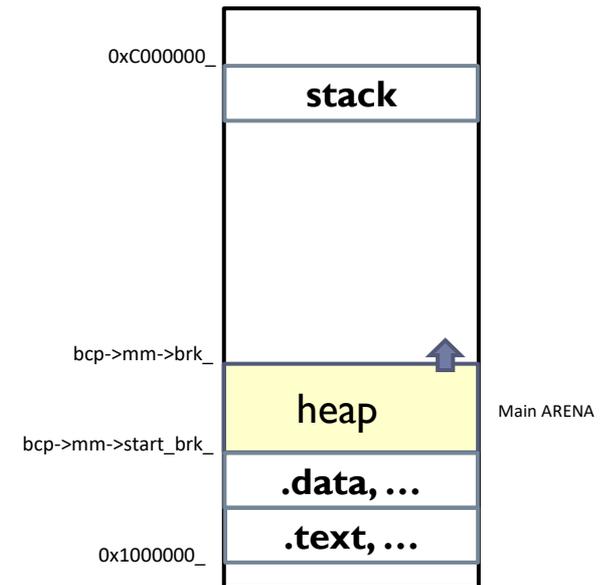
## ▶ morecore (int n\_cab)

▶ SI ( $n\_cab < \text{min\_ncab}$ )  
 $n\_cab = \text{min\_ncab};$  // 144 bytes = 18 cabeceras

▶  $\text{freep->ptr} = \text{sbrk}(n\_cab * \text{sizeof}(\text{Header}))$

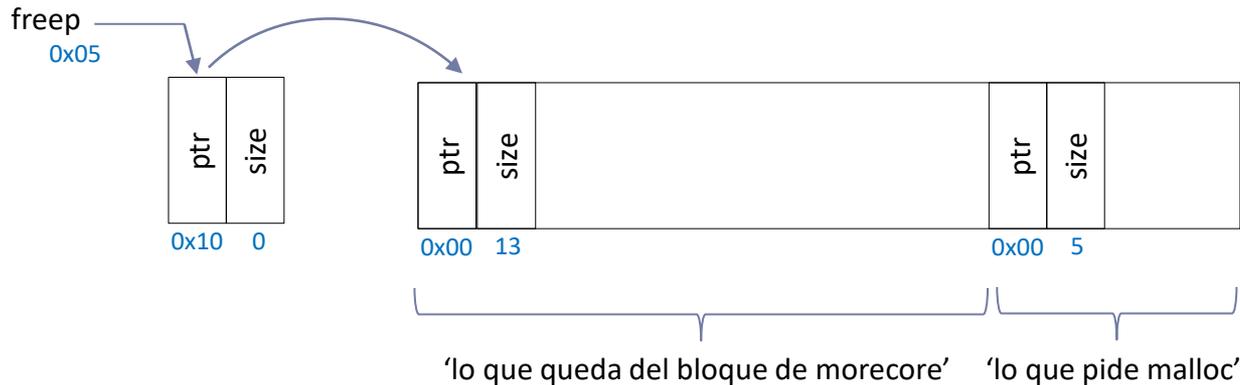
▶  $\text{freep->ptr->ptr} = \text{null};$

▶  $\text{freep->ptr->size} = n\_cab;$



# Ejemplo de *libc* storage allocator

## malloc

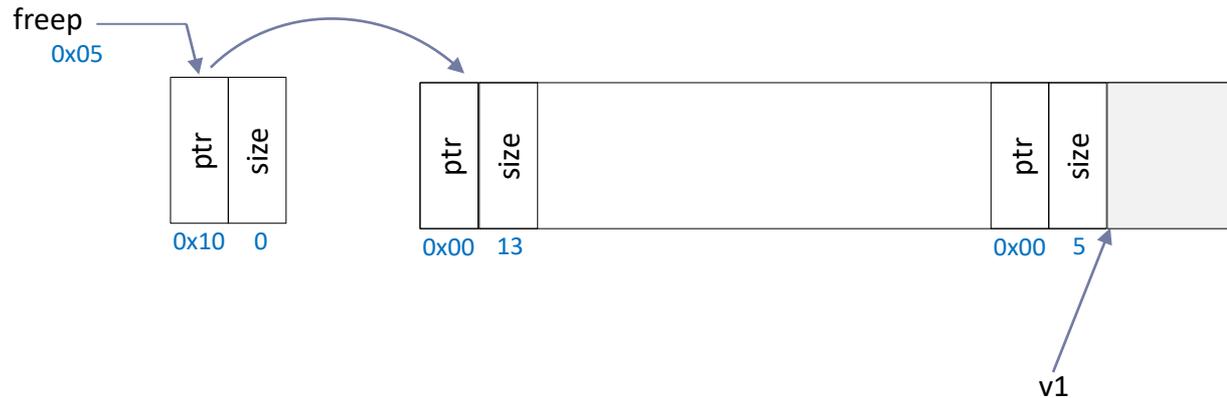


- ▶ `int *v1;`
- ▶ `char *v2 ;`
  
- ▶ `v1 = malloc(8*sizeof(int)) ; // 8 int = 4 pares de int = 4 cabeceras`  
`// + 1 par más para metainformación`

# Ejemplo de *libc* storage allocator

## malloc

---

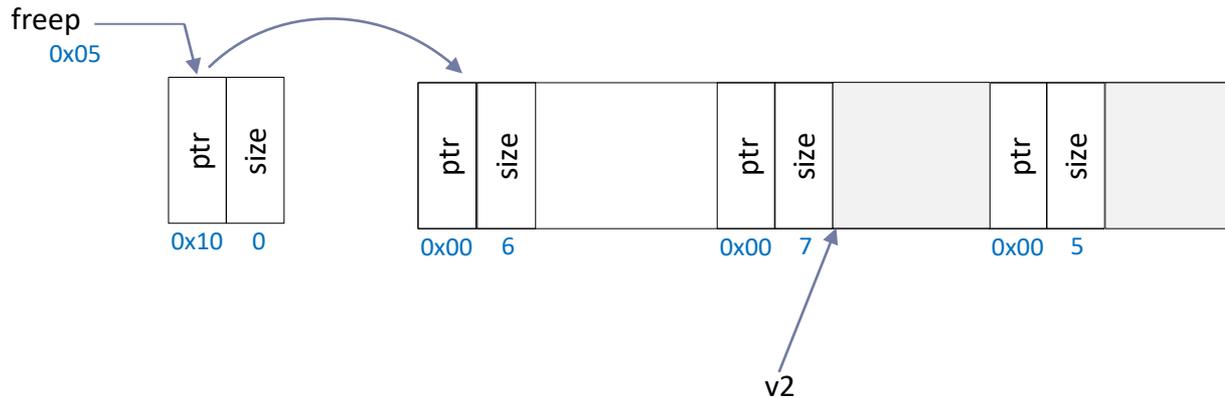


- ▶ `int *v1;`
- ▶ `char *v2 ;`
  
- ▶ `v1 = malloc(8*sizeof(int)) ;`

# Ejemplo de *libc* storage allocator

## malloc

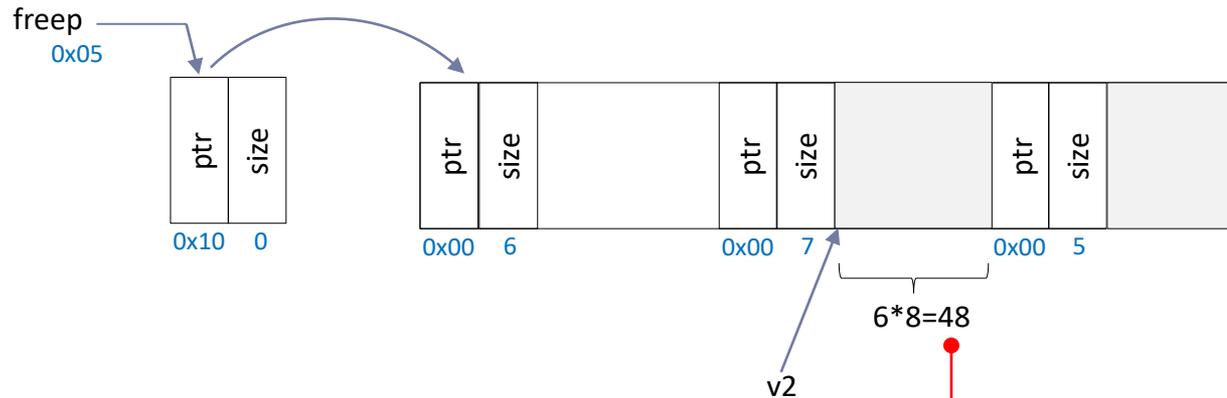
---



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

# Ejemplo de *libc* storage allocator

## problema de fragmentación interna



▶ `int *v1;`

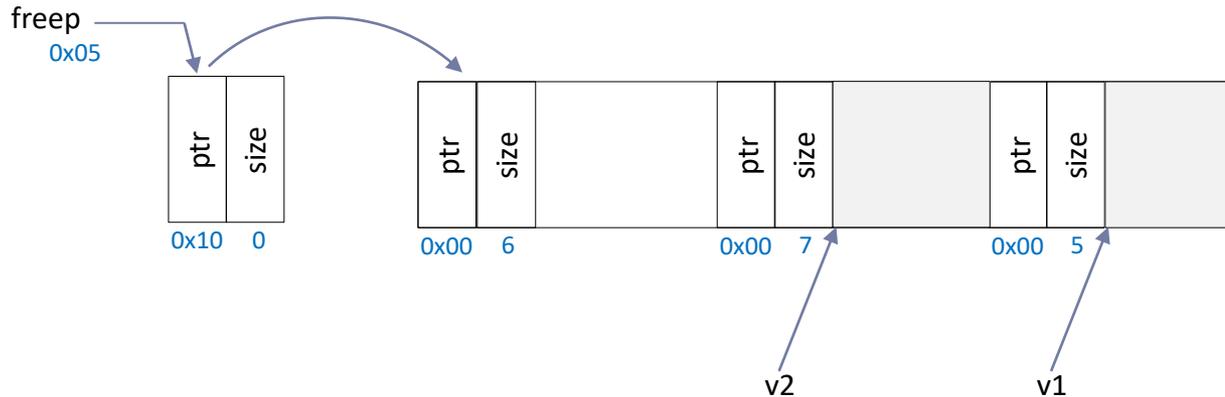
▶ `char *v2;`

▶ `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

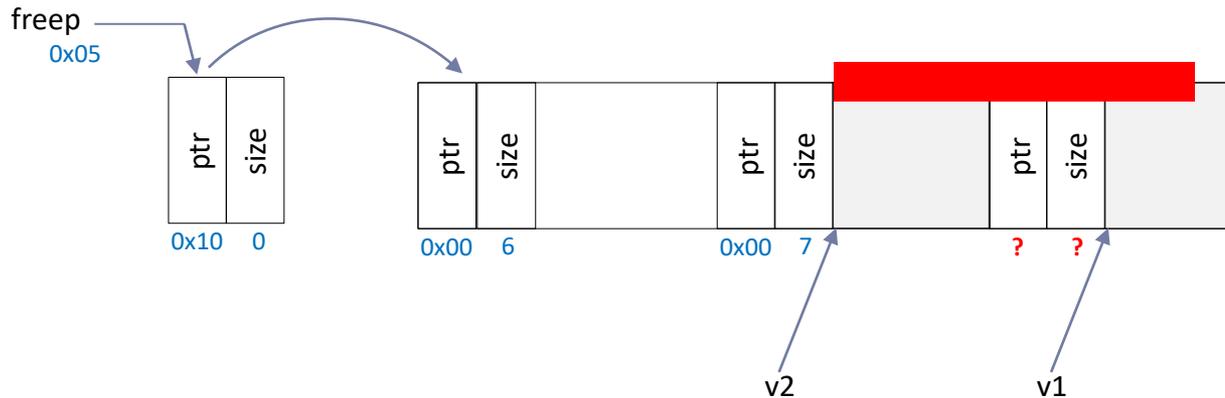
# Ejemplo de *libc storage allocator* problema de sobrescritura



- ▶ // 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

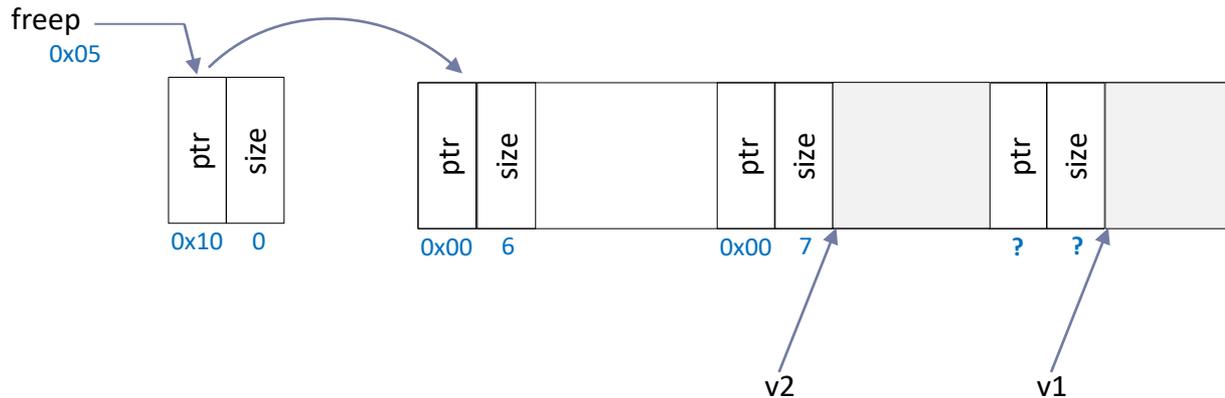
## problema de sobrescritura



- ▶ // 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 sobrescritura

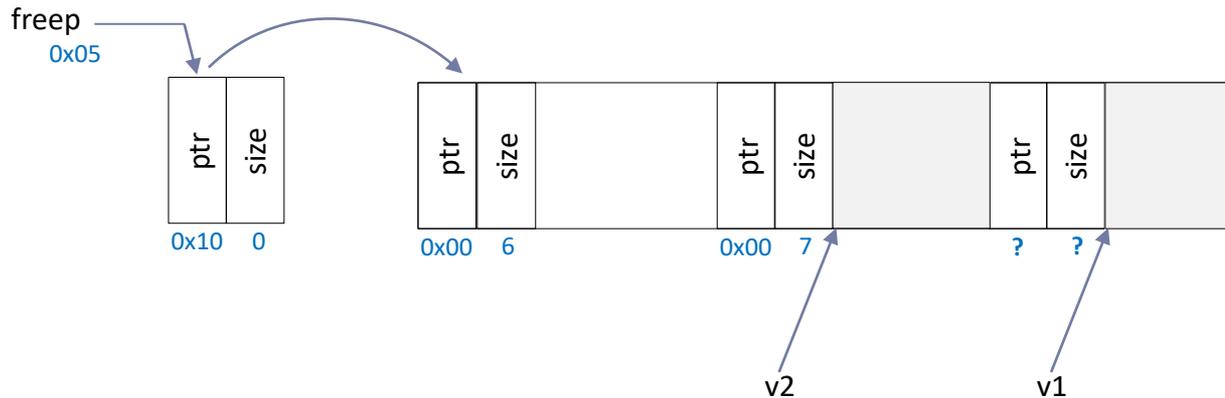


- ▶ // se ha reservado solo 41 caracteres para v2
- ▶ for (int i=0; i<64; i++)
  - ▶ v2[i] = 'x' ;
- ▶ free(v1) ; <- incapaz de recuperar la cabecera válida... SIGSEGV

# Ejemplo de *libc* storage allocator

## otros problemas típicos

---

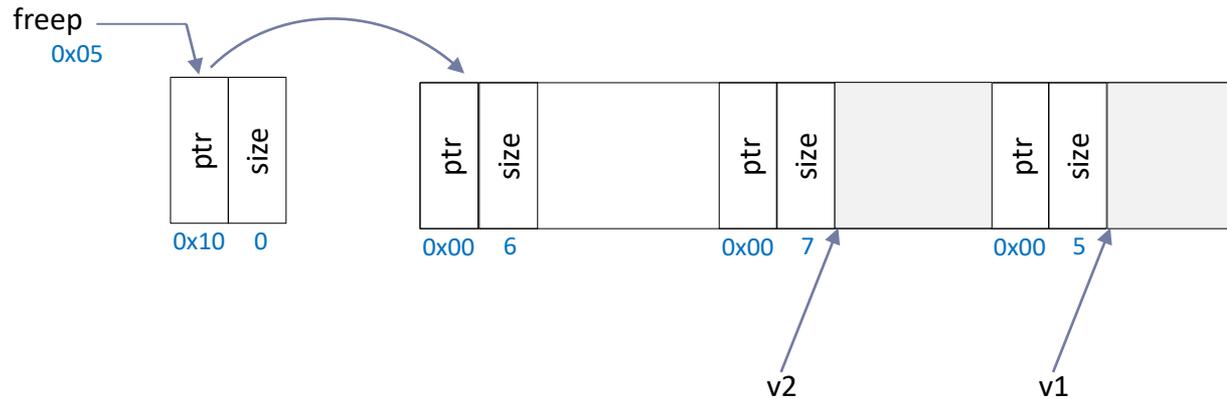


- ▶ Liberar una zona de memoria no gestionada:
  - ▶ `int i; free(&i);`
- ▶ Liberar dos veces una misma zona de memoria
- ▶ Acceder a memoria no pedida anteriormente
  - ▶ `char *pchar; printf("%s",pchar);`

# Ejemplo de *libc* storage allocator

## free

---

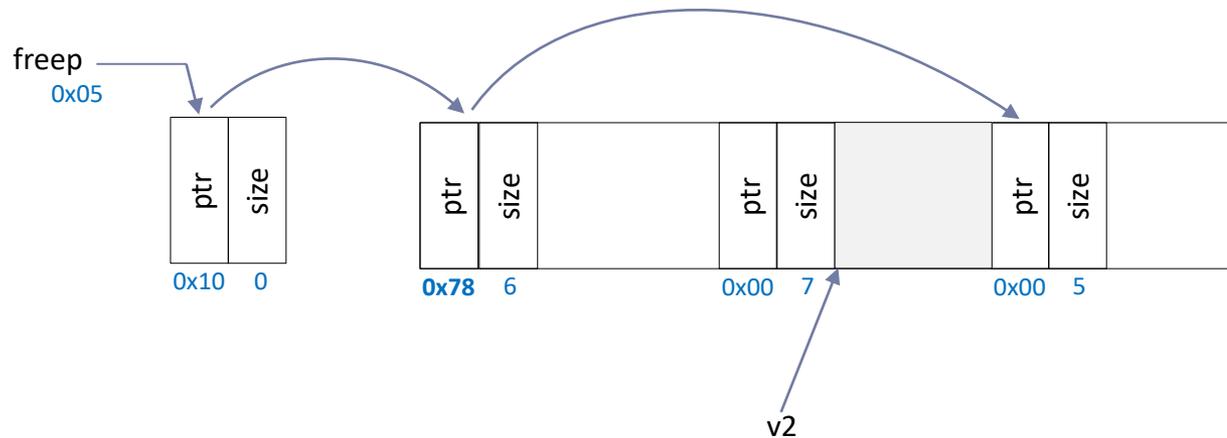


► `free(v1);`

# Ejemplo de *libc* storage allocator

## free

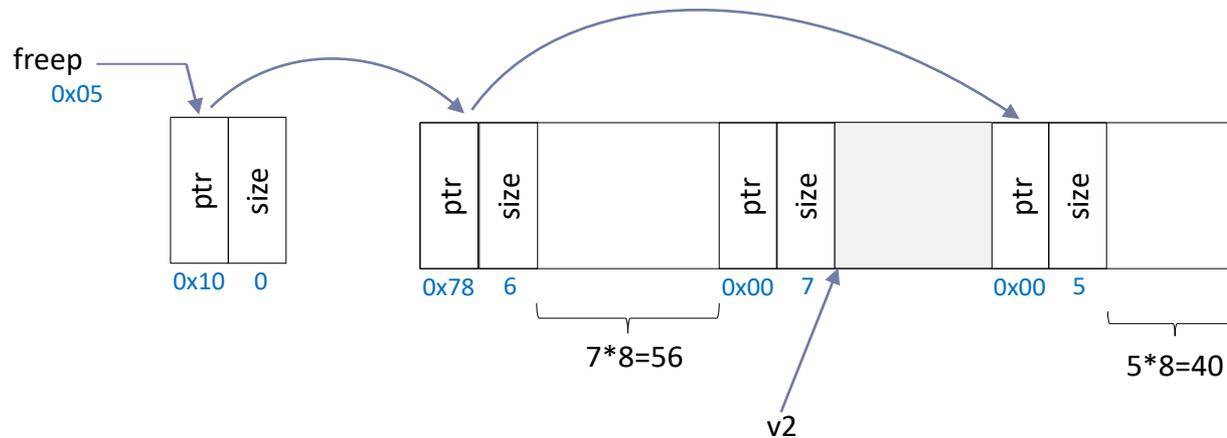
---



► `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 (se precisa una llamada a `morecore` para pedir más memoria)

# Gestión de memoria dinámica

---

- ▶ Principales posibles problemas en la gestión clásica:
  - ▶ Fragmentación interna
  - ▶ Sobre-escritura
  - ▶ Liberar zona de memoria no gestionada
  - ▶ Liberar dos veces la misma zona de memoria
  - ▶ Acceder a memoria no pedida anteriormente
  - ▶ Fragmentación externa
- ▶ ¿Alguna ventaja?
  - ▶ Sencillez... o eso dicen.
  - ▶ Rapidez... o eso dicen.

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

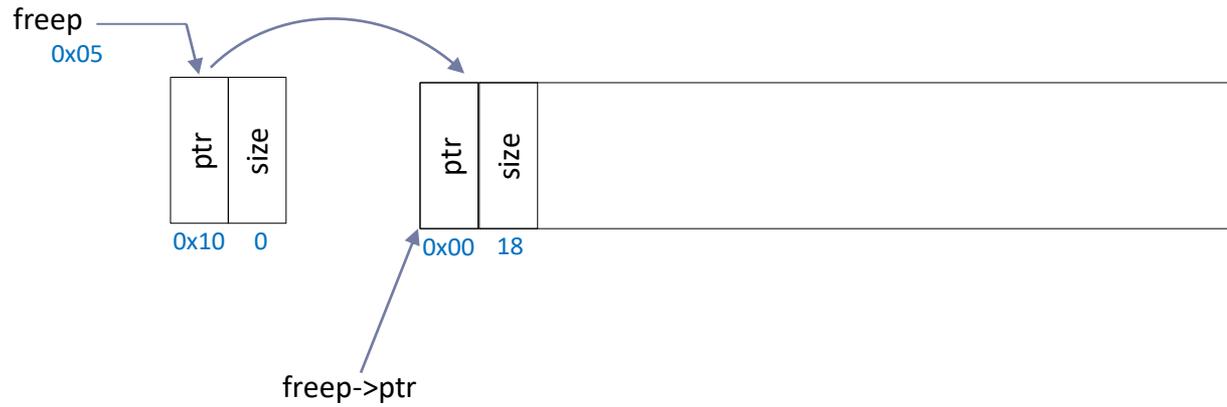
acaldero@phoenix:~/infodso/$ gdb ptr
GNU gdb (GDB) 7.2-abuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted 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 símbolos desde /home/acaldero/work/infodso/memoria/ptr...hecho.
(gdb) run
Starting program: /home/acaldero/work/infodso/memoria/ptr

Program received signal SIGSEGV, Segmentation fault.
0xb7e79221 in ?? () from /lib/libc.so.6
```

# Ejemplo de *libc storage allocator*

paralelismo, escalabilidad, ...

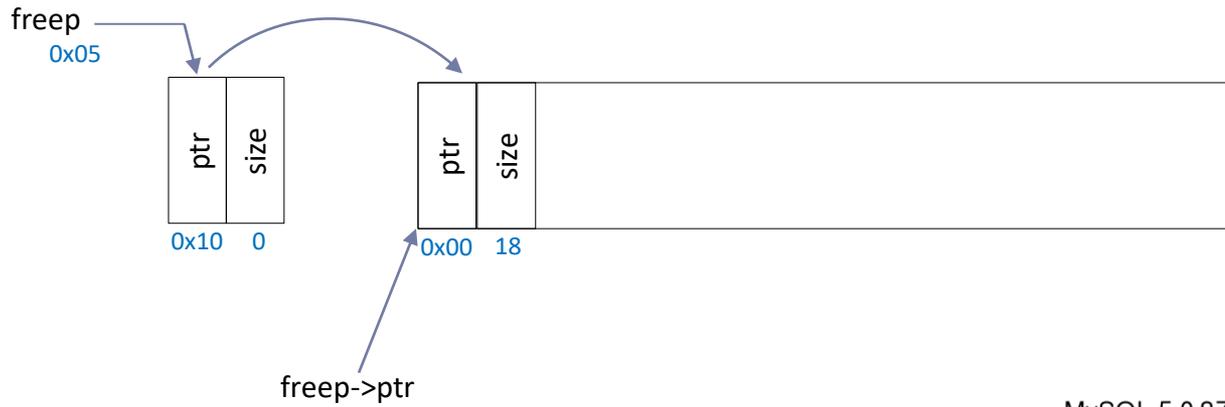
---



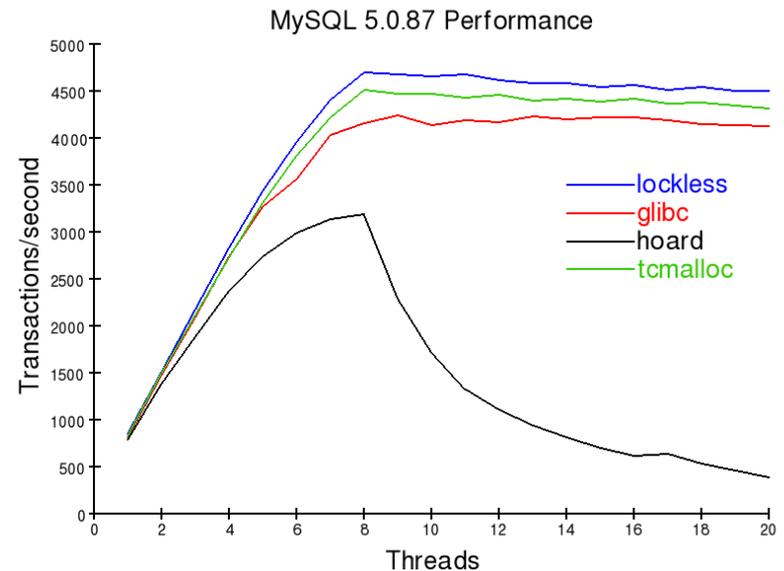
- ▶ ¿Acceso de varios hilos?
  - ▶ Cerrojos
- ▶ ¿Escalabilidad?
  - ▶ ¿El acceso de muchos hilos y cerrojos?
  - ▶ Pool separado por hilo, *lockless*, etc.

# Ejemplo de *libc* storage allocator

paralelismo, escalabilidad, ...

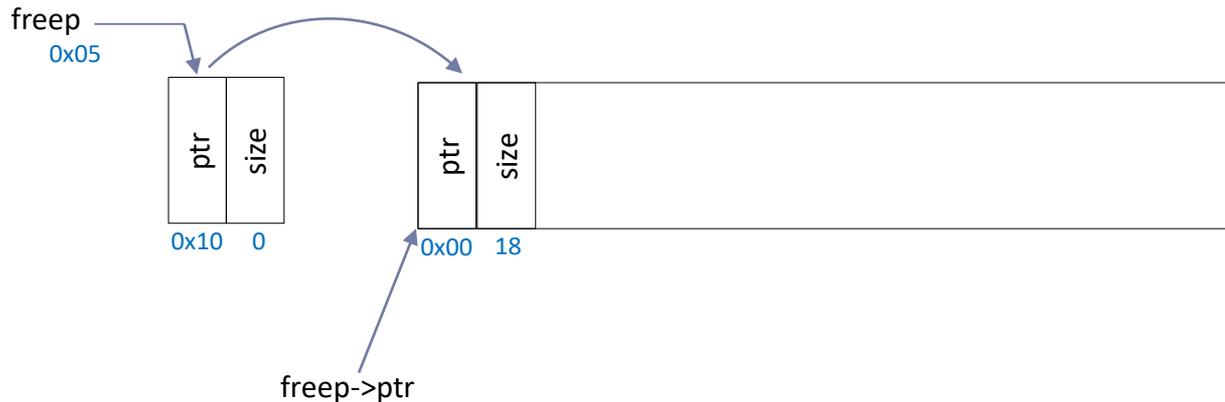


- ▶ ¿Acceso de varios hilos?
  - ▶ Cerrojos
- ▶ ¿Escalabilidad?
  - ▶ ¿El acceso de muchos hilos y cerrojos?
  - ▶ Pool separado por hilo, *lockless*, etc.



# Ejemplo de *libc* storage allocator

## contención...



### ▶ `_get_mem (int n_cab)`

▶ `if (n_cab < min_ncab)`  
`n_cab = min_ncab;`

▶ `if (n_cab*2*sizeof(int) < M_MMAP_THRESHOLD) ||`  
`(times_mmap_used > M_MMAP_MAX)`

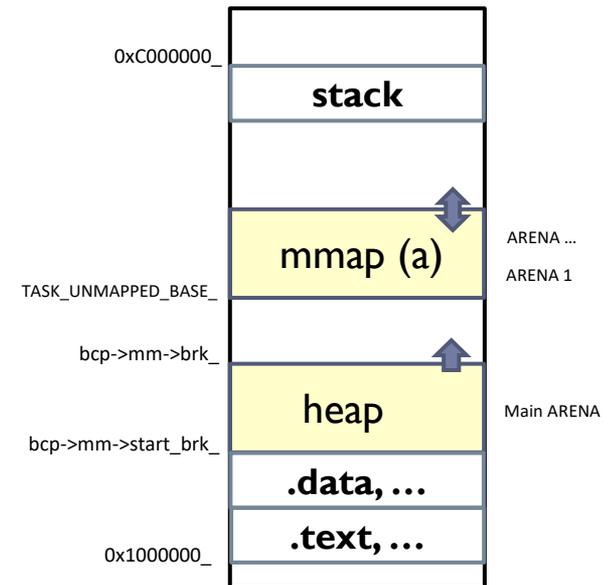
▶ `freep->ptr = sbrk(↑ n_cab*2*sizeof(int))`

▶ `else`

▶ `freep->ptr = mmap(↕ (... , n_cab*2*sizeof(int))`

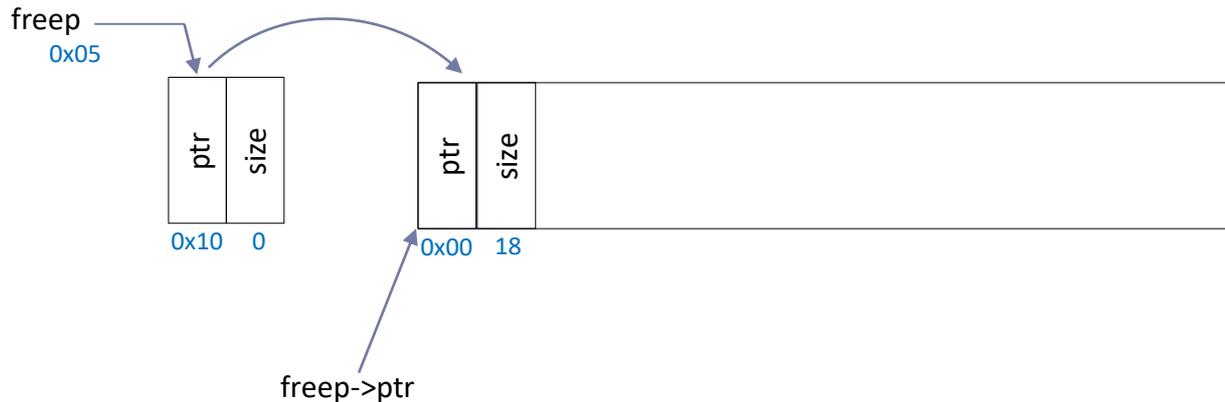
▶ `freep->ptr->ptr=null;`

▶ `freep->ptr->size=n_cab;`

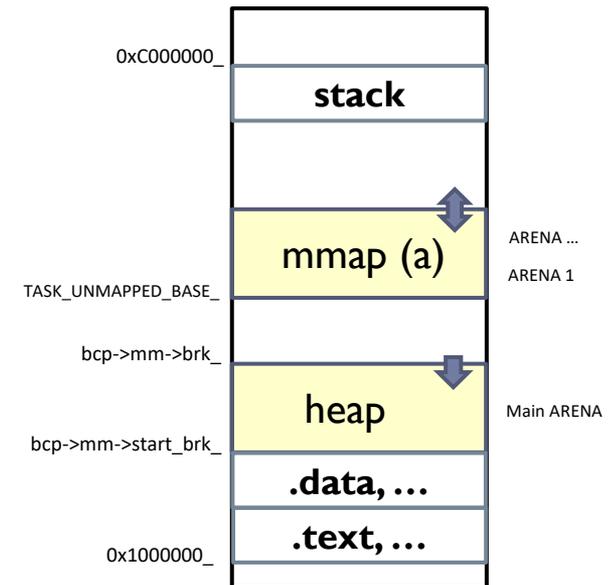


# Ejemplo de *libc* storage allocator

## contención...



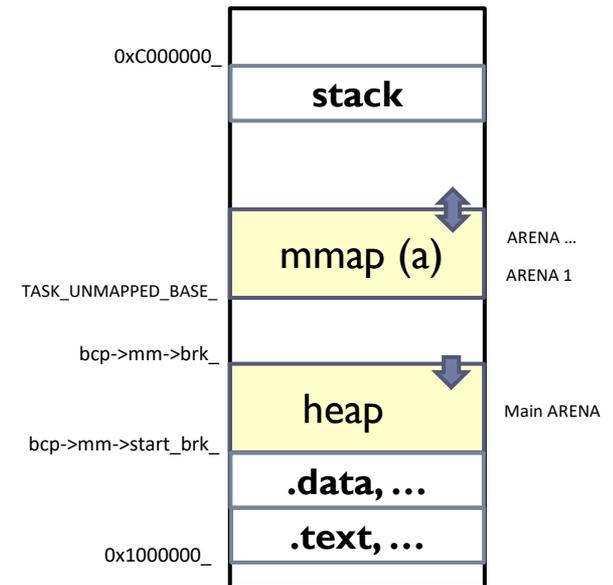
- ▶ `_ret_mem (void *ptr)`
  - ▶ `nbytes = ptr->size*2*sizeof(int)`
  - ▶ `If (nbytes < M_MMAP_THRESHOLD) || (times_mmap_used > M_MMAP_MAX)`
    - ▶ `If (nbytes > M_TRIM_THRESHOLD)`
      - `freep->ptr = sbrk(- n_cab*2*sizeof(int))`
  - ▶ `else`
    - ▶ `freep->ptr = munmap(..., n_cab*2*sizeof(int))`



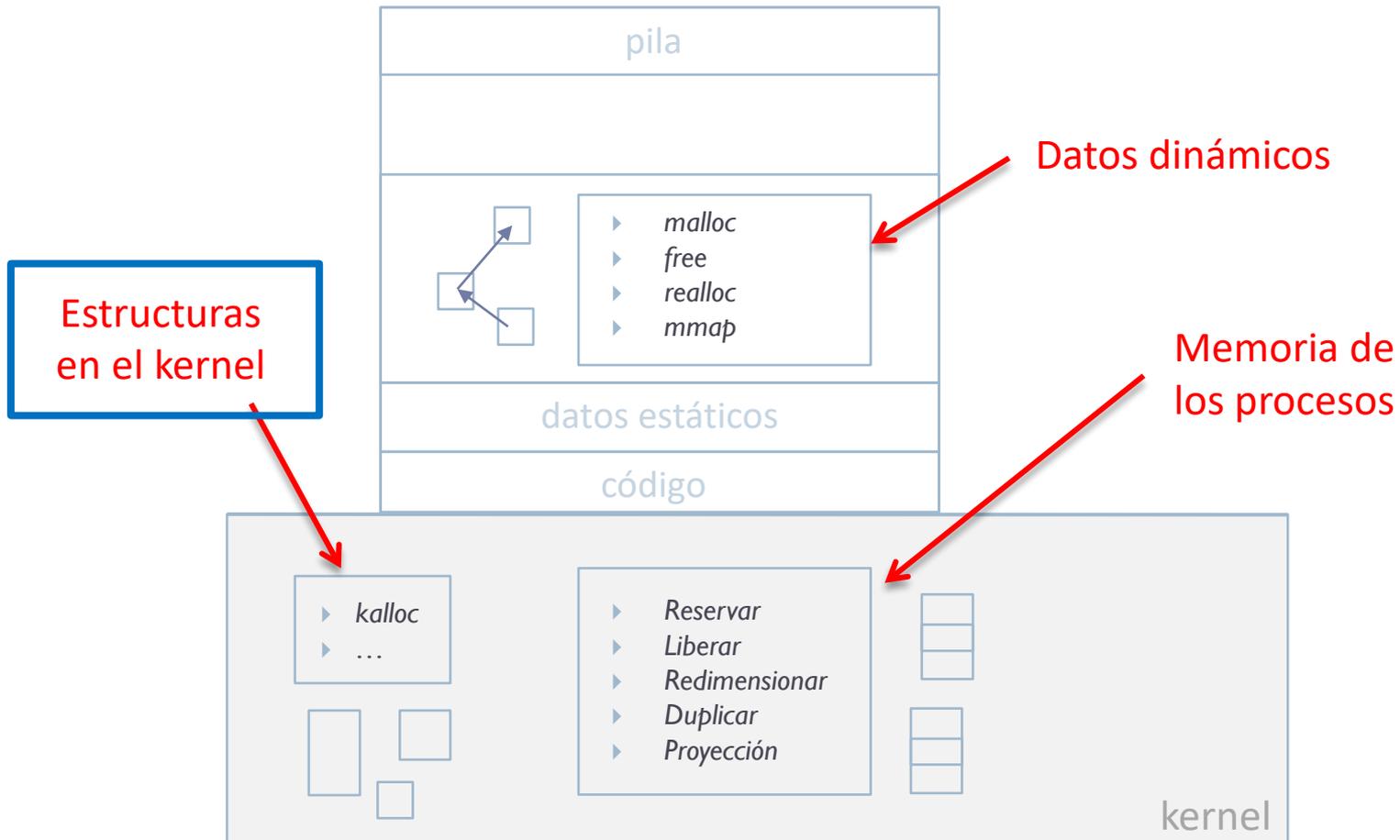
# Ejemplo de *libc* storage allocator contención...

Malloc	User (sec)	System (sec)	Elapsed	Major Faults	Minor Faults
Normal	216.0	166.7	6:29.20	170	23099385
Tuned	196.7	14.3	3:41.71	168	16820

- ▶ `_ret_mem (void *ptr)`
  - ▶ `nbytes = ptr->size*2*sizeof(int)`
  - ▶ `If (nbytes < M_MMAP_THRESHOLD) || (times_mmap_used > M_MMAP_MAX)`
    - ▶ `If (nbytes > M_TRIM_THRESHOLD)`
      - `freep->ptr = sbrk(- n_cab*2*sizeof(int))`
  - ▶ `else`
    - ▶ `freep->ptr = munmap(..., n_cab*2*sizeof(int))`



# Gestores a varios niveles



# Gestión de la memoria en el kernel

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

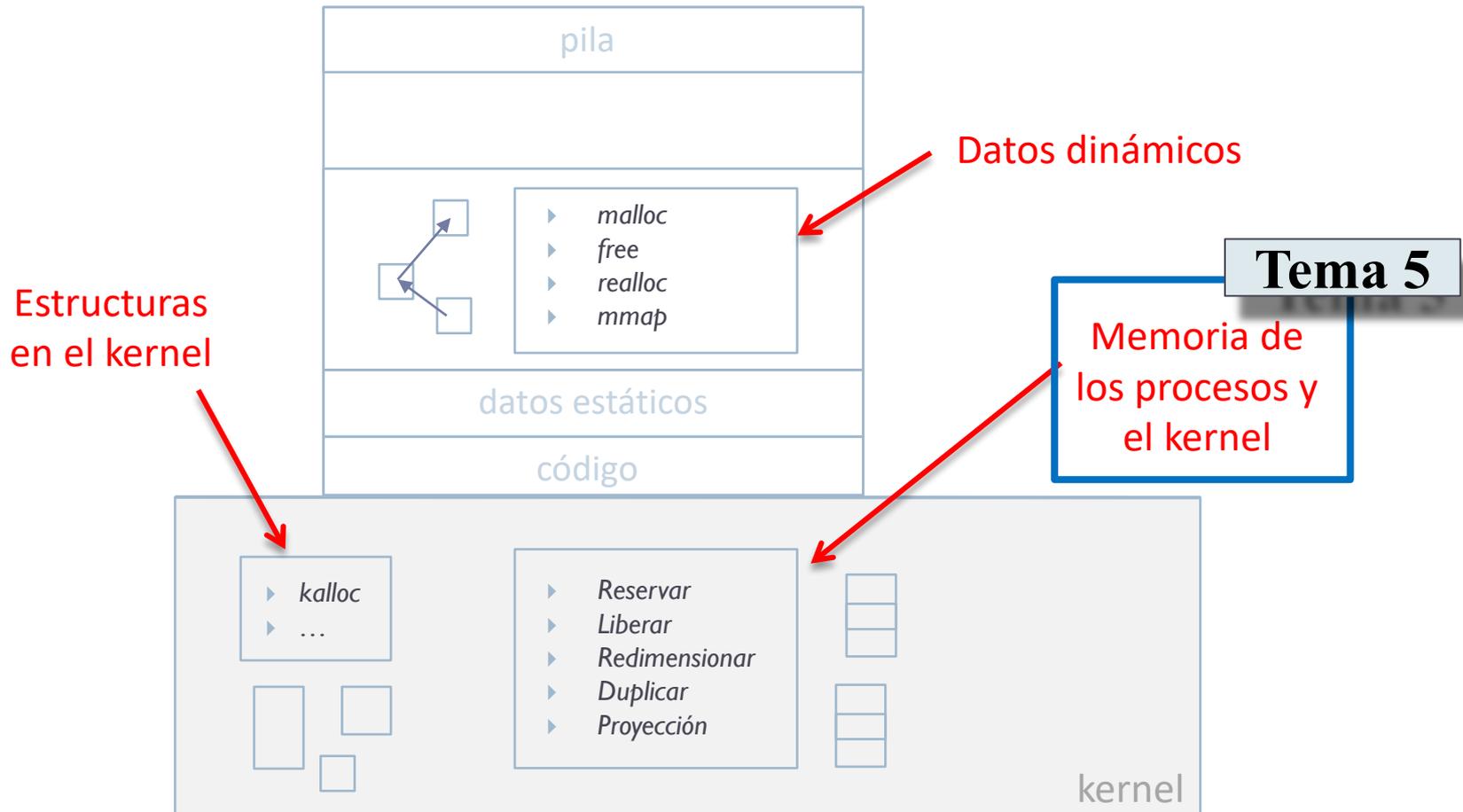
	0	128k	256k	512k	1024k	
start	1024k					
A=70K	A	128	256	512		
B=35K	A	B	64	256	512	
C=80K	A	B	64	C	128	512
A ends	128	B	64	C	128	512
D=60K	128	B	D	C	128	512
B ends	128	64	D	C	128	512
D ends	256		C	128	512	
C ends	512			512		
end	1024k					

# 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`

# Gestores a varios niveles



**Tema 5**

Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 1 (c)

## libc: gestión de memoria dinámica

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.

