Sistemas Operativos

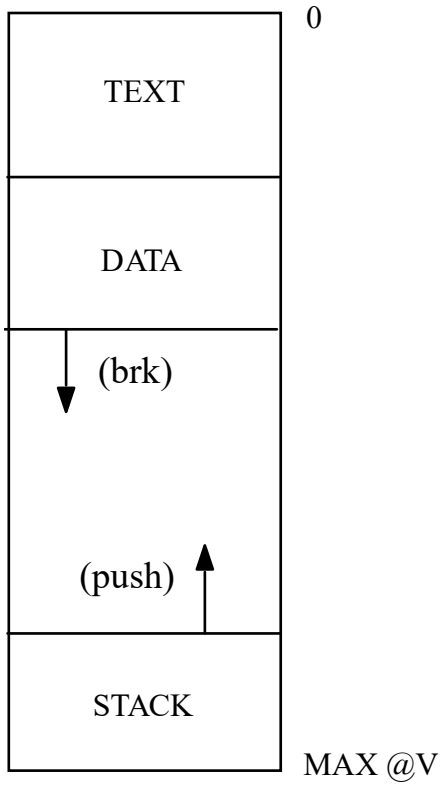
Gestión de Memoria en UNIX

Gestion de Memoria

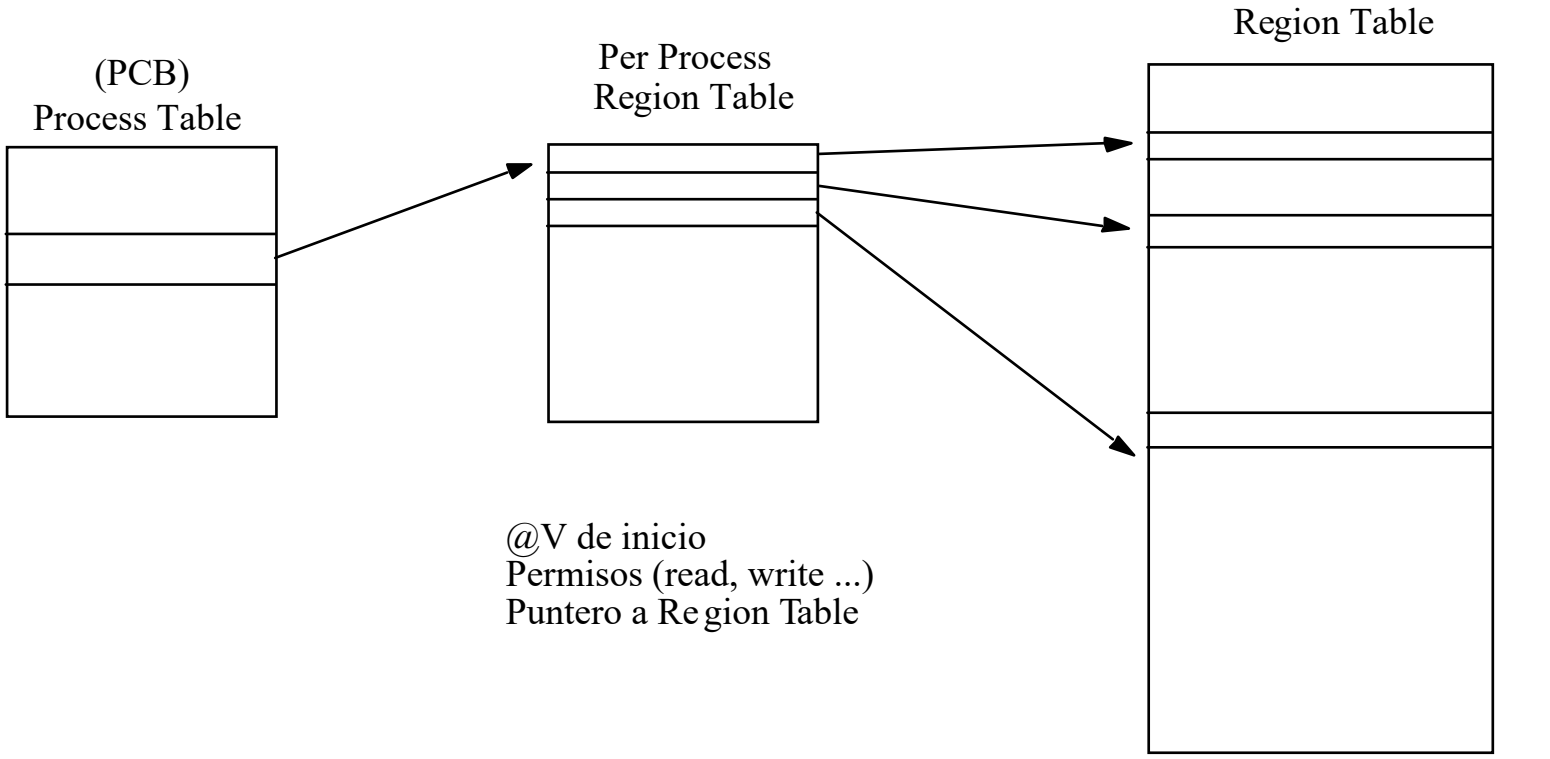
- Regiones y Tablas
- Sistemas basados en SWAP
- Sistemas basados en Demand Paging
- Ejemplo fork()
  - fork() y Swap
  - fork() y Demand Paging
- Llamadas asociadas
- Funciones de librería, ejemplos
- Mapeo de ficheros en memoria: ejemplos

Gestión de memoria en UNIX: Regiones

- Región:
  - Area contigua del espacio virtual de un proceso
  - Tratada como un único objeto
  - Uniforme en cuanto a permisos, shared...



Gestión de memoria en UNIX: Tablas

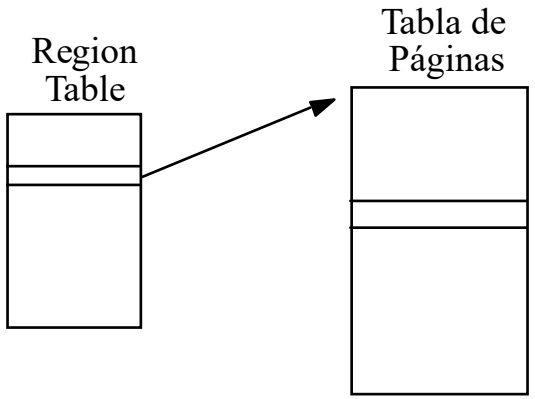


Sistemas basados en SWAP

- Espacio en disco(swap device):
  - una partición de disco, almacena procesos expulsados
  - gestión de espacio por particiones variables
- Razones de expulsión de un proceso
  - fork(), brk(), crecimiento de pila
  - Se requiere espacio para recuperar otro proceso
- Solo se copia a disco la parte del espacio virtual usada
- Proceso swapper
  - Es un proceso mas
  - Devuelve procesos de disco a memoria
    - Entra en ejecución cada cierto tiempo
    - Mira si hay procesos “preparados en disco”
    - Busca espacio en memoria y los copia
    - Si no lo hay busca víctima y la expulsa

Sistemas basados en Demand Paging

Estructuras de datos del Kernel

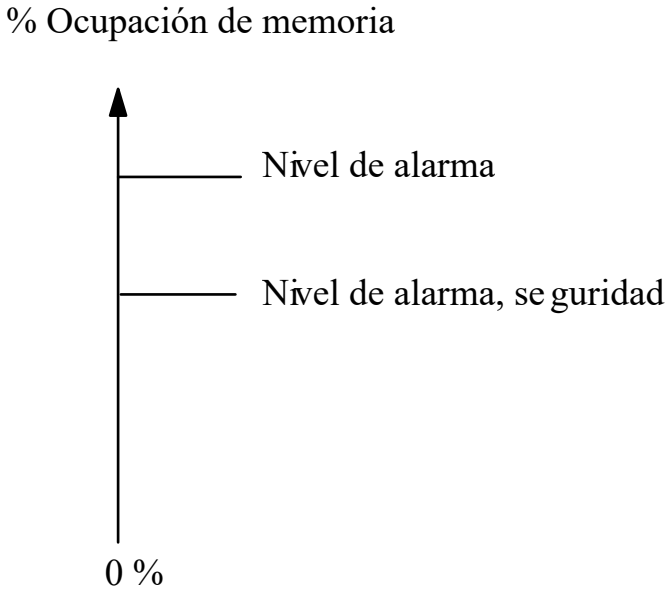


Contenedor
Acceso: read, write
Estado: V, M, copy-on-write
Disk Block descriptor

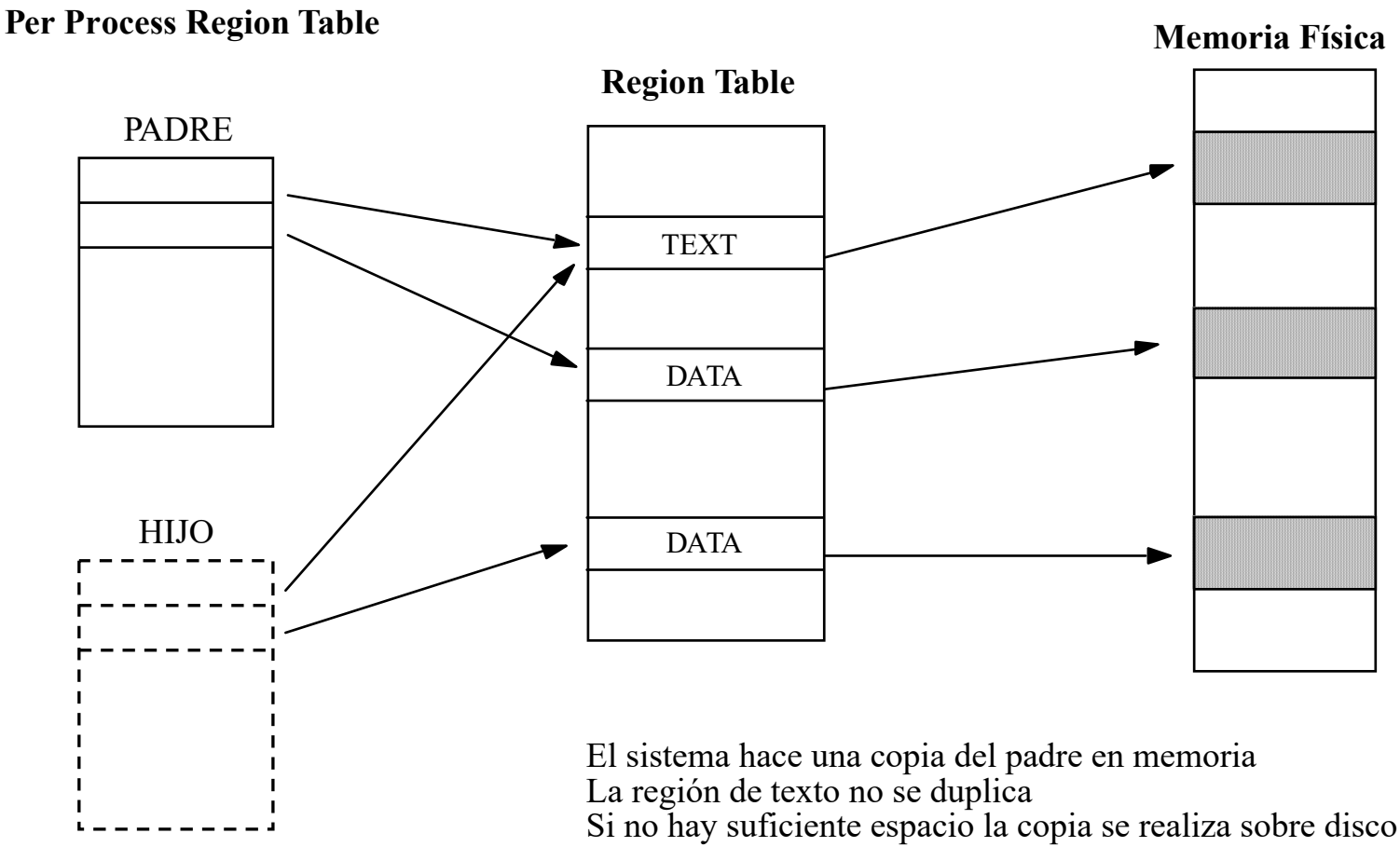
Tabla global de estado de contenedores
Tabla de páginas en disco

Page stealer

Proceso encargado de sacar páginas de memoria a disco

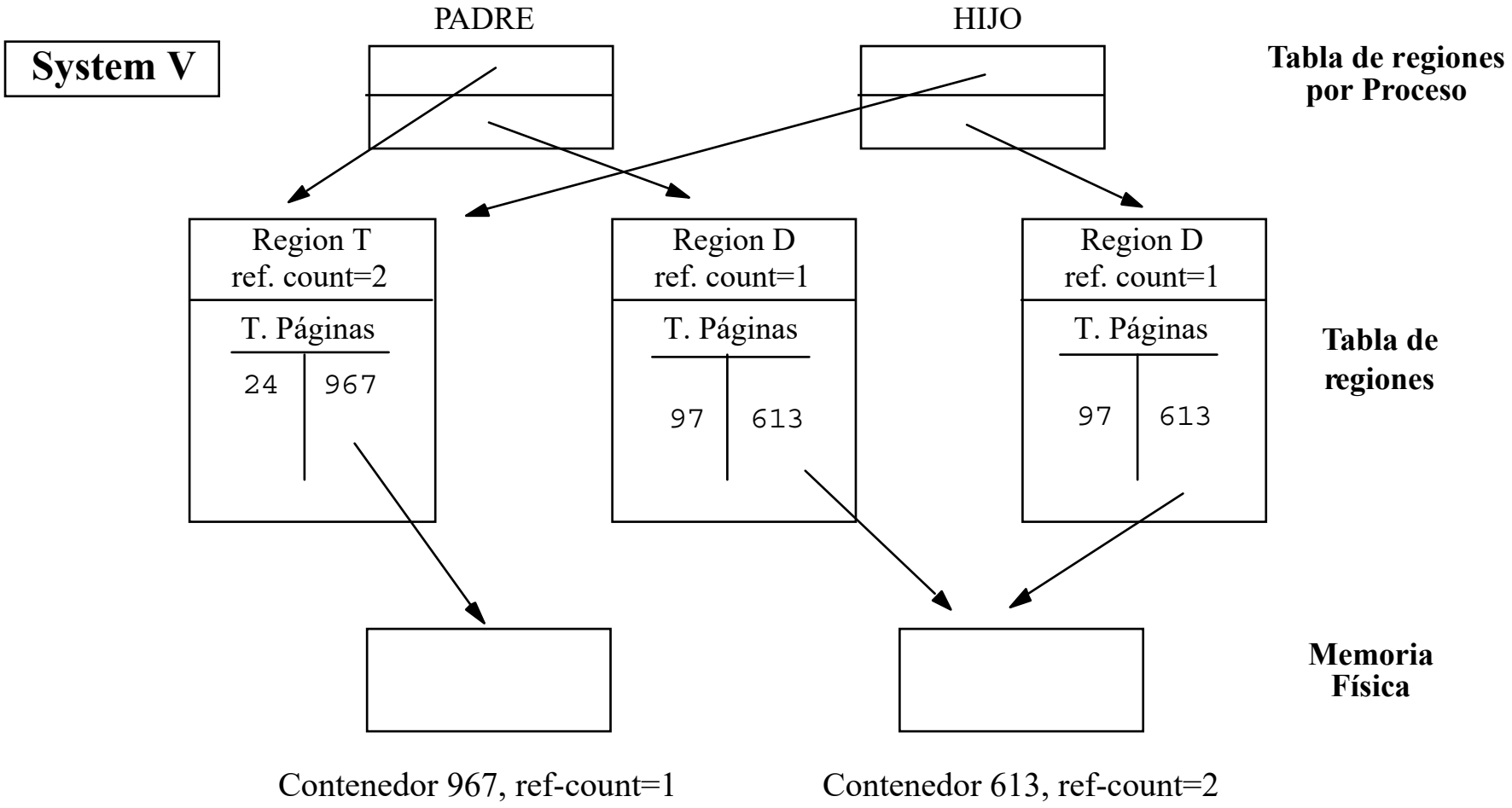


fork() y Swap



El sistema hace una copia del padre en memoria
La región de texto no se duplica
Si no hay suficiente espacio la copia se realiza sobre disco

fork() y Demand Paging



fork() y Demand Paging

BSD

- Crea una copia de todas las páginas privadas:
  - Data, Stack
- Ofrece una llamada alternativa a fork()
- vfork(): no copia ni las tablas de página
  - Trata las regiones de datos como la de texto
  - Padre e hijo comparten memoria física
  - Solo debe usarse para llamar a exec() inmediatamente

Llamadas asociadas

- `brk(end_ds)`
  - Cambia límite superior del segmento de datos a `end_ds`
- `old_end_ds= sbrk(increment)`
  - Suma `increment` bytes al límite superior del segmento de datos
  - `Increment` puede ser negativo
  - devuelve el valor antiguo de `end_ds`
- `brk()` puede fallar por varias razones:
  - Al ampliar la región se colisiona con otra
  - Al ampliar la región se sobrepasa el espacio virtual máximo del proceso
  - Faltan recursos:
    - No hay espacio físico en memoria o en disco (dependiendo de la implementación)

Funciones de librería

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

- `void * malloc(size_t size);`
  - Reserva espacio para `size` bytes. No inicializa el espacio
- `void * calloc(size_t nelem, size_t elsize);`
  - Reserva espacio para `nelem` elementos de `elsize` bytes
  - Inicializa el espacio reservado con ceros
- `void * free(void * ptr);`
  - Libera el espacio apuntado por `ptr`. Este espacio no queda liberado para el Kernel
  - Hay una estructura de gestión de espacios propia de la librería
  - No mezclar llamadas a `brk()` con funciones de librería.
- `void * realloc(void * ptr, size_t newsize);`
  - Cambia el tamaño del bloque apuntado por `ptr` al valor `newsize` (sin inicializar si aumenta el espacio)
  - Devuelve puntero al inicio de bloque, puede ser distinto del original
  - Si `ptr=NULL` actua como `malloc()`
    Si `size=0` actua como `free()`

Ejemplo malloc(), calloc()

```
main(argc, argv)
int argc; char *argv[];
{
   int x[1000], y[1000]; /* reserva 1000 elem */
   int i, n, res=0;

   n=atoi(argv[1]);
   leer_vector(x,n);
   leer_vector(y,n);
   for(i=0;i<n;i++)      /* solo se usan n */
       res+=x[i]*y[n-i-1];
   printf("res=%d\n",res);
}
```

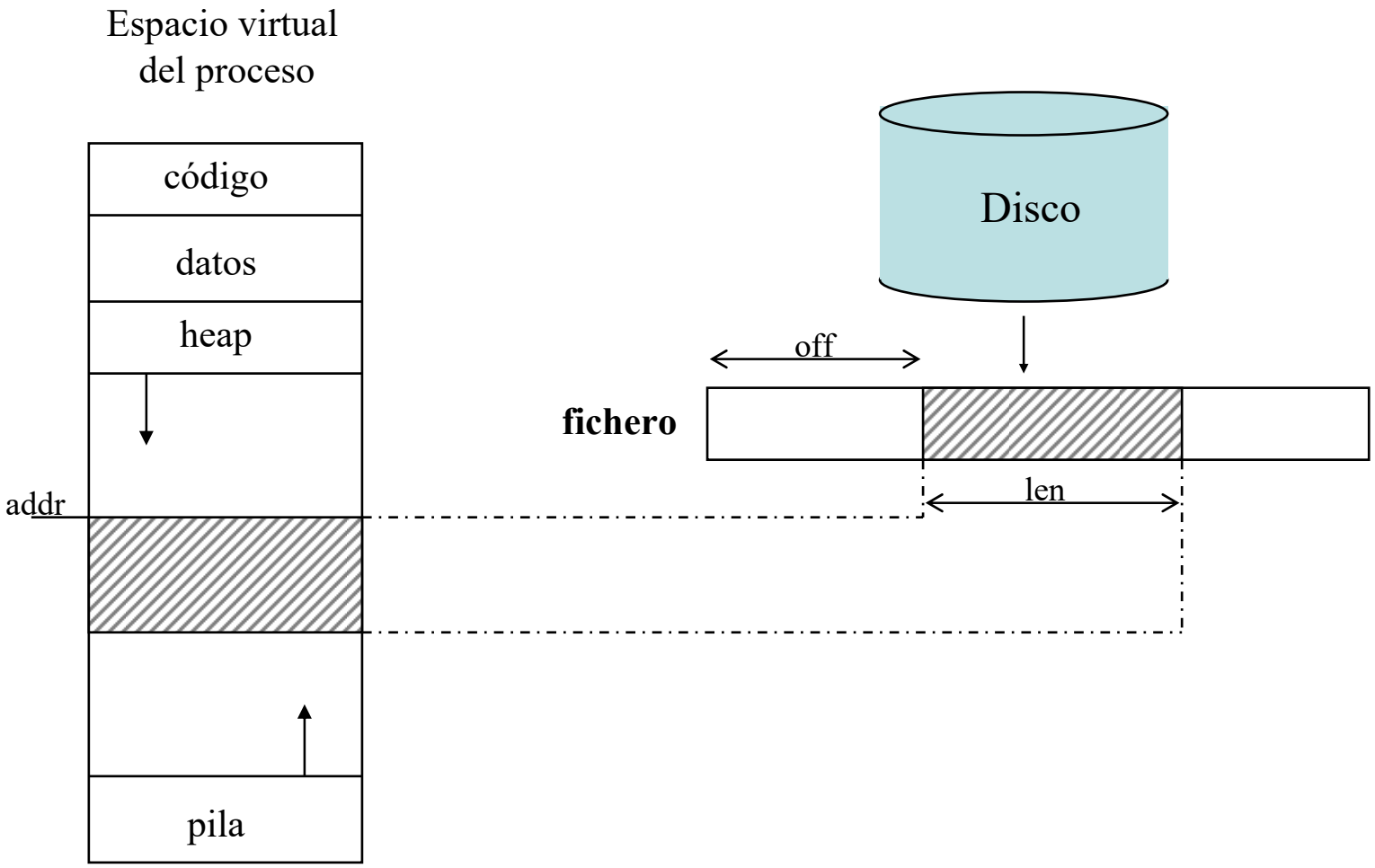
Ejemplo malloc() , calloc()

```
main(argc, argv)
int argc; char *argv[];
{
   int *x,*y;
   int i, n, res=0;
   n=atoi(argv[1]);
   x=malloc(n*sizeof(int));/* reserva n*4 bytes */
   y=calloc(n,sizeof(int));/* reserva espacio n int */
   leer_vector(x,n);
   leer_vector(y,n);
   for(i=0;i<n;i++)
       res+=x[i]*y[n-i-1];
   printf("res=%d\n",res);
}
```

Ejemplo realloc()

```
main()
{
   int *p, *q;
       ...
   p=calloc(1000,4);
       ...
   q=p+50;
       ...
   p=realloc(p,8000);
       ...
   printf("p[50]=%d\n",*q);
}
```

Mapeo de ficheros en memoria



Mapeo de ficheros en memoria: mmap()

- Permite mapear un fichero de disco en un buffer de memoria
- Para realizar operaciones E/S sin read/write
- Fichero previamente abierto

```
#include <sys/mman.h>
void * mmap(addr, len, prot, flag, filedes, off)
char *addr;
size_t len;
int prot, flag, filedes;
off_t off;
```

- `addr=0`: colócalo donde quieras (recomendable)
- `addr!=0`: hint (intenta colocarlo en addr) addr debe ser múltiplo del tamaño de página
- `len`: número de bytes mapeados
- `off`: a partir de que punto del fichero se mapea (normalmente debe ser múltiplo del tamaño de página)

Mapeo de ficheros en memoria: mmap()

```
#include <sys/mman.h>
void * mmap(addr, len, prot, flag, filedes, off)
char * addr;
size_t len;
int prot, flag, filedes;
off_t off;
```

- `filedes`: descriptor del fichero, tiene que estar abierto
- `prot`: intención de uso: debe respetar los del open()
  PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
  ejemplo: PROT_READ | PROT_WRITE
- `flag`: MAP_FIXED: addr pasa de hint a obligación
  MAP_SHARED: un store sobre la región = write sobre fichero
  MAP_PRIVATE: un store provoca una copia privada
  el fichero nunca se modifica
- mmap devuelve:
  - Si todo va bien -> @ comienzo zona de mapeo
  - Si hay error: `MAP_FAILED ((void *)-1)`

/* reverse.c */

```
#include <stdio.h>
#include <fcntl.h>
#include "error.h"

main(argc,argv)
int argc;    char *argv[];
{    char c;
     int i, fdfnt;
     long where;

     if(argc != 2){ printf( "Uso: %s fichero_a_invertir" argv[0]); exit(1); }

     if((fdfnt = open( argv[1], O_RDONLY )) == -1) syserr("open");
     if((where = lseek( fdfnt, -1L ,2 )) == -1 )   syserr("lseek");

     while(where >= 0){
          read(fdfnt, &c, 1);
          write(1, &c, 1);
          where = lseek ( fdfnt, -2L ,1 );
     };
}
```

Invierte el contenido de un fichero

formato long

Ejemplo mmap(): mreverse (1de2)

```
/* mreverse.c    Invierte el contenido de un fichero.*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>    /* mmap() */
#include <fcntl.h>
#include "error.h"

main(argc,argv)
int argc;
char *argv[]; {
    int fdfnt;
    long fsize;
    char *src;

    if(argc != 2){
         printf( "Uso: %s fichero_a_invertir", argv[0]);
         exit(1);
    }
```

Ejemplo mmap(): mreverse (2de2)

```
    if((fdfnt = open(argv[1], O_RDONLY)) == -1)
          syserr("open del primer fichero");
    if((fsize=lseek(fdfnt,0,SEEK_END)) == -1)
          syserr("lseek al final del fichero");

    src=mmap(0,fsize,PROT_READ,MAP_SHARED,fdfnt,0);
    if (src == MAP_FAILED) syserr("mmap error for input");

    fsize--;
    while(fsize >= 0){
          write(1, &src[fsize], 1);
          fsize--;
    }
}
```