Sistemas operativos II

Memoria en UNIX

- Introducción
- Espacio de direcciones de los procesos
- Organización de la memoria física
- Robo de páginas e intercambio

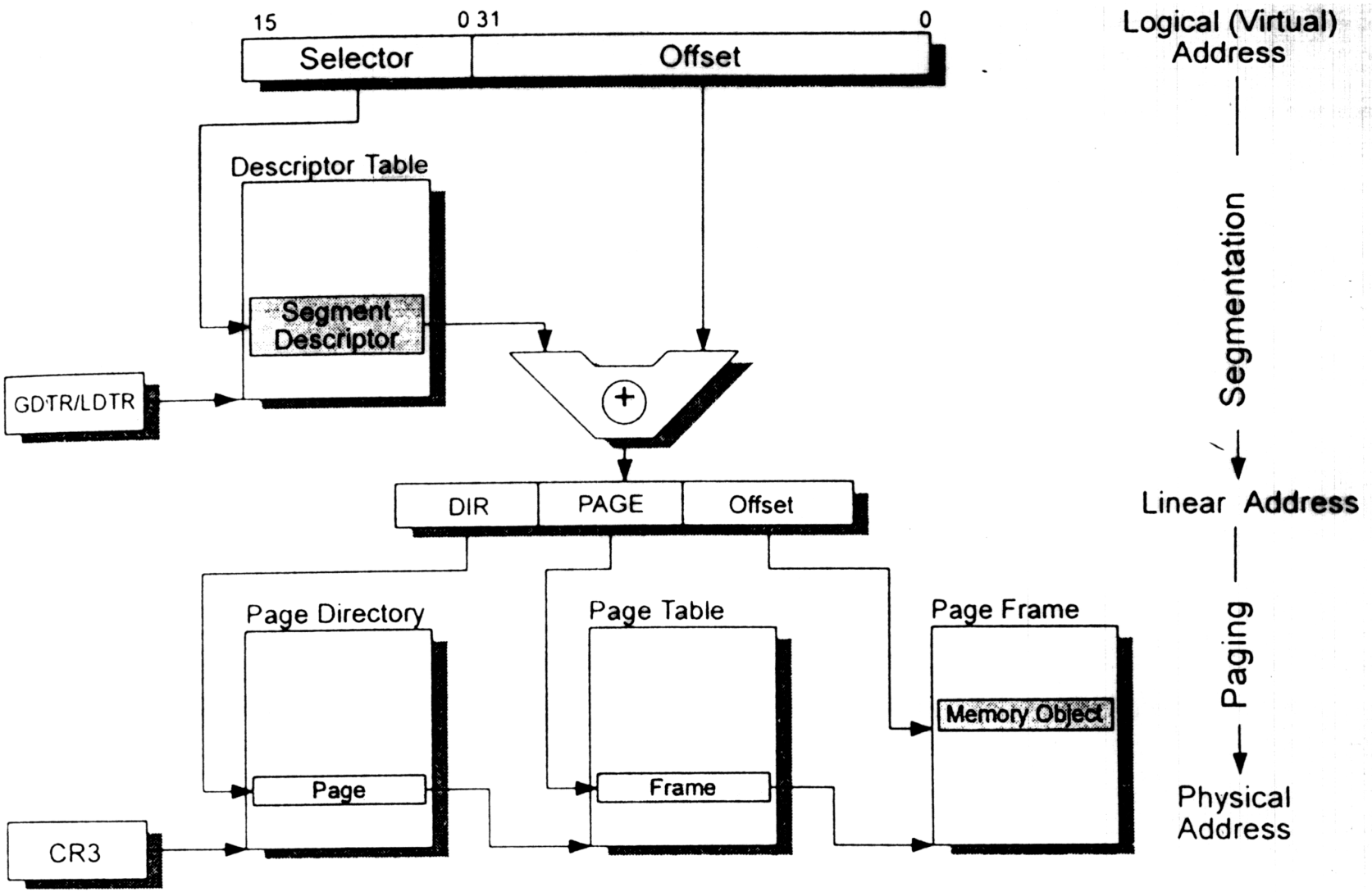
Memoria en UNIX

- Introducción
- Espacio de direcciones de los procesos
- Organización de la memoria física
- Robo de páginas e intercambio

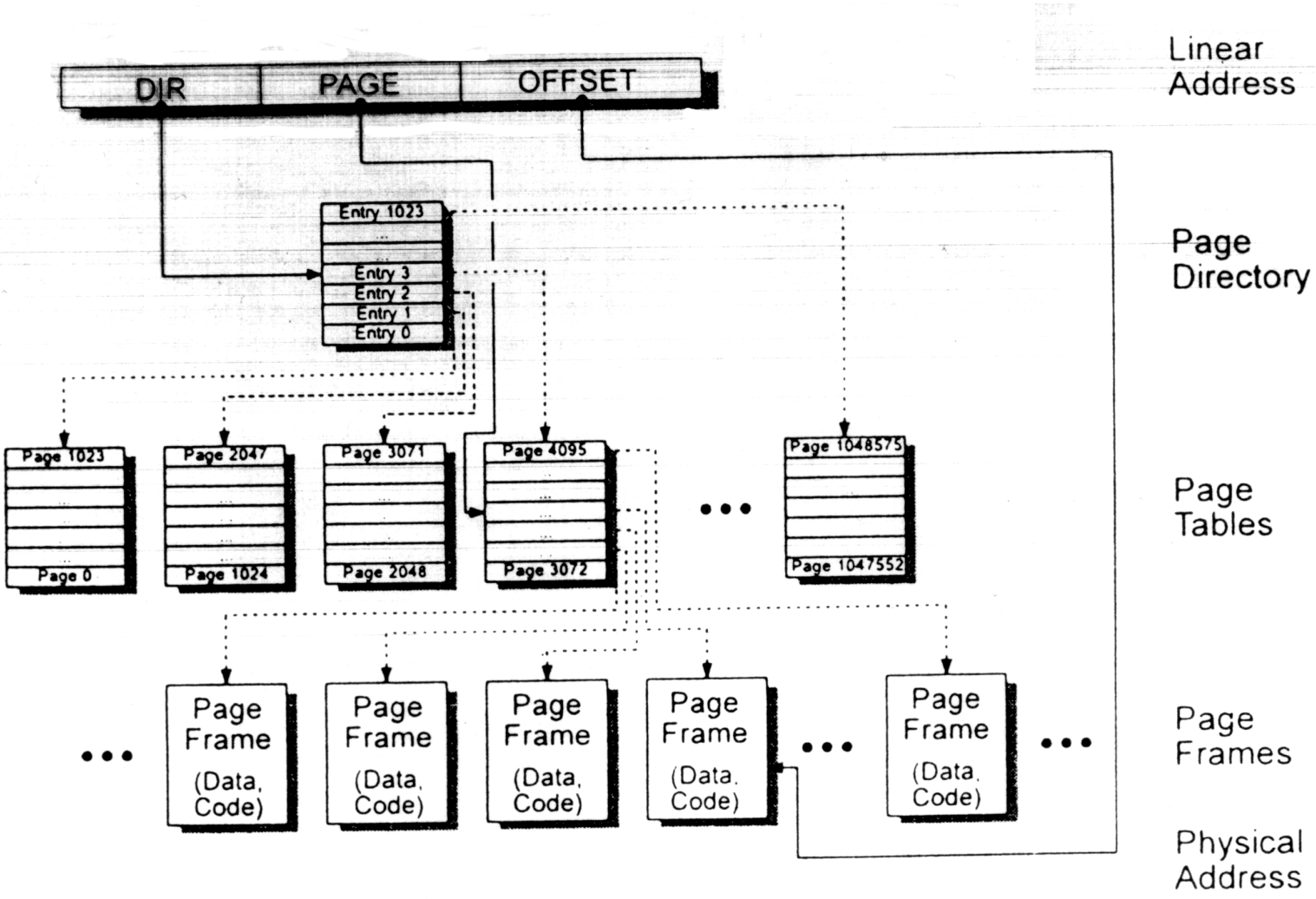
Generalidades

- Los procesos se ejecutan en un espacio de direcciones virtual
- Usualmente la memoria física está dividida en trozos de tamaño fijo denominados marcos o páginas físicas
- Tamaño de página depende del procesador. En sistemas actuales típicamente 4Kbytes
- Lo procesos trabajan con direcciones virtuales compuestas por número de página y desplazamiento dentro de la página. Ejemplo: en un sistema con páginas de 4K la dirección 0x800c03a9, corespondería a desplazamiento 0x3a9 en la página 0x800c0
- La traslación de número de página y desplazamiento a número de página física (marco) y desplazamiento se hace en tiempo de ejecución con ayuda del *hardware*

Esquema del direccionamiento en la arquitectura i386



Esquema del direccionamiento en la arquitectura i386



Generalidades

- Si el sistema soporta memoria virtual
  - un proceso no tiene que estar en memoria totalmente para poderse ejecutar: la tabla de páginas ha de tener un bit que indica si está o no en memoria.
  - El S.O. guarda información de dónde en el almacenamiento secundario está cada página no residente en memoria física
  - Cuando se crea un proceso se necesita memoria para las estructuras `proc` y `u_area`
  - el código y los datos inicializados residen en el fichero de disco
  - la pila y los datos sin inicializar se ajustan durante la ejecución
  - solo se asigna memoria para aquellas páginas que son referenciadas: paginación bajo demanda pura

Generalidades

- Las páginas de memoria *no usadas* se sacan de la memoria: *robo de páginas*.
  - En unix el proceso que realiza esta tarea es *pageout*, *pagedaemon o paged*
- Si se produce hiperpaginación (los procesos tienen menos páginas que su *working set* en memoria), se intercambian procesos enteros a disco.
  - En unix esto lo hace el *swapper* (*sched*)

Memoria en unix: regiones

- En su acepción mas simple, un proceso consta de tres regiones: código, datos y pila.
  - **código** (*text*) Contiene el código de las funciones del programa. Es de solo lectura
  - **datos** (*data* + *bss*) Contiene los datos (variables globales, tanto inicializadas como sin inicialización explicita) del proceso. El montículo (*heap*) suele ser parte de esta región
  - **pila** (*stack*). Usado para pasar los parámetros a las funciones y por éstas para sus variables locales

memoria en unix: regiones

- Cada proceso tiene una tabla de regiones del proceso con referencias a las entradas en la tabla de regiones del kernel que constituyen el proceso
- cada entrada de la tabla de regiones que contiene
  - donde (dirección virtual) comienza y termina la región
  - permisos de la región (solo lectura, lectura-escritura, lectura ejecución ...)
  - puntero a entrada en la tabla de regiones del kernel

memoria en unix: regiones

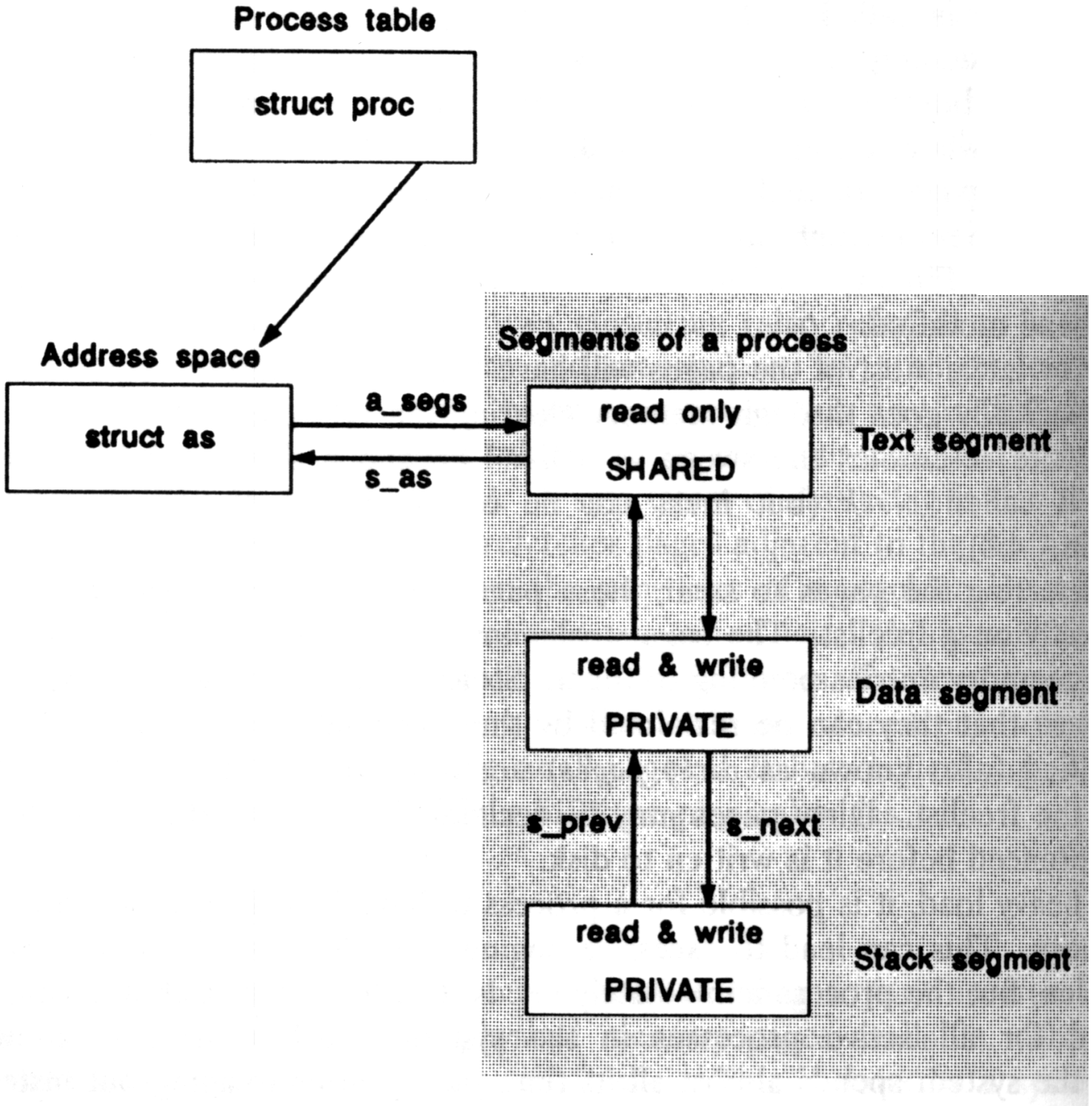
- El kernel contiene una tabla de regiones del kernel
- En la tabla de regiones del kernel hay un puntero a los mapas de traducción de direcciones virtuales a reales (dependientes de la implementación)
- Cada entrada en la tabla de regiones del kernel contiene
  - puntero al inodo cuyos contenidos se cargan en esa región
  - tipo de región (código, datos, pila, memoria compartida, *mmaped file*)
  - tamaño de la región
  - localización de la región en memoria física
  - contador de referencias

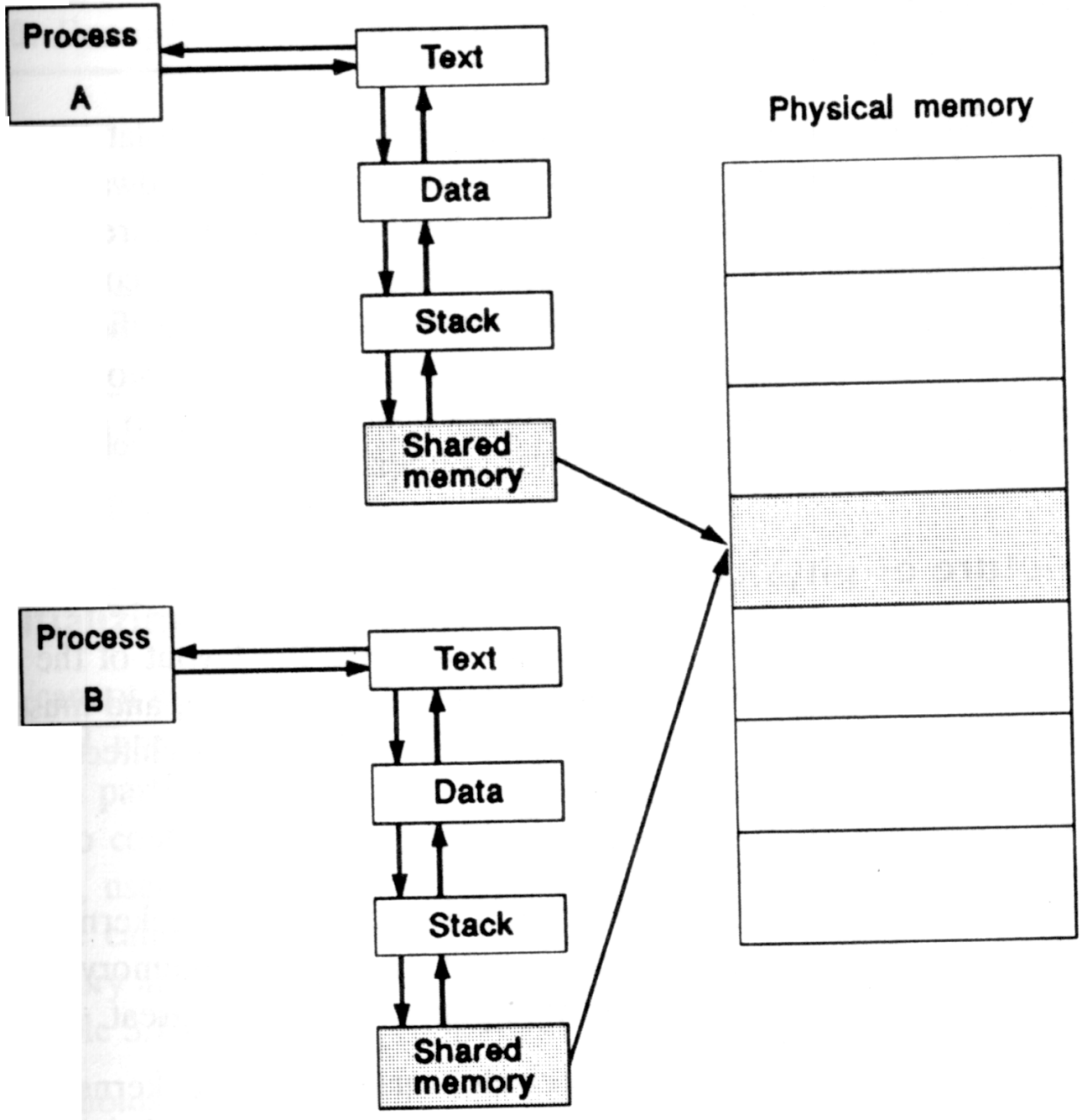
operaciones sobre regiones

- las operaciones sobre regiones que hace el kernel son
- **asignar una región** (*fork()*, *exec()*, *shmget()*)
- **encadenar una región a un proceso** (*fork()*, *exec()*, *shmat()*)
- **cambiar el tamaño de una región** (*brk()*, *sbrk()*, *(malloc())*)
- **cargar una región** (*exec()*)
- **desasignar una región** (*exec()*, *exit()*, *shmctl(IPC_RMID,)*)
- **desencadenar una región del espacio de direcciones de un proceso** (*exec()*, *exit()*, *shmdt()*)
- **duplicar una región** (*fork()*)

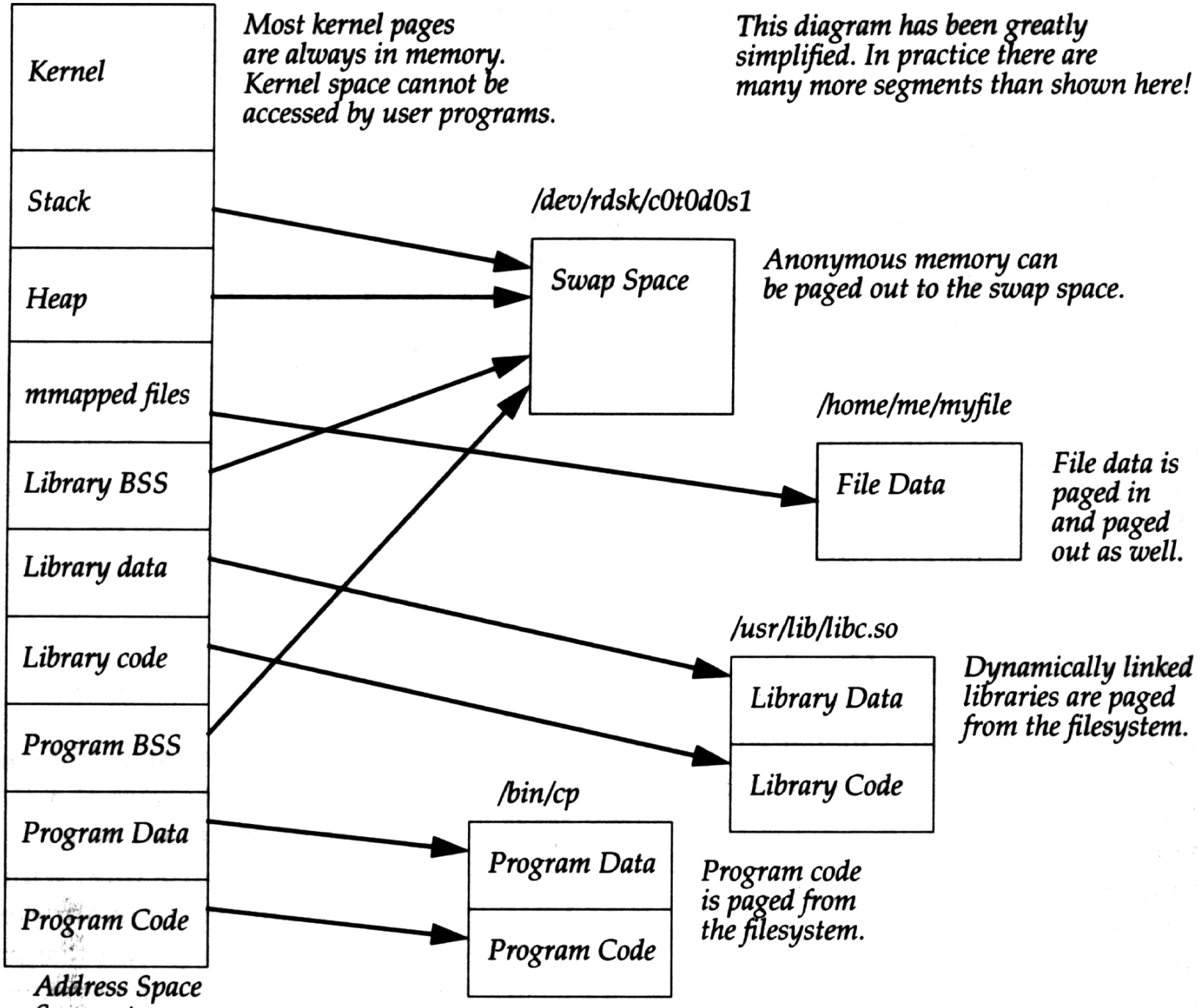
memoria en System V R4

- El espacio de direcciones de un proceso se almacena como una lista enlazada de sus segmentos (código, datos, pila . . .)
- en la estructuta *proc* hay una referencia a una estructura as (*address space*)
- la estructura *as* contiene la referencia a la primera de las estructuras de la lista que describe los segmentos que constituyen el espacio de direcciones
- cada segmento esta descrito por una estructura *seg* que contiene una referencia a una estructura *segvn_data*.


Process table
struct proc
Address space
struct as
a_segs
s_as
Segments of a process
read only
SHARED
Text segment
read & write
PRIVATE
Data segment
s_prev
s_next
read & write
PRIVATE
Stack segment


Process
A
Text
Data
Stack
Shared
memory
Physical memory
Process
B
Text
Data
Stack
Shared
memory

Mapa de memoria en solaris



Salida del comando pmap

- en solaris

```
08046000       8K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000       8K rwx--    [ heap ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000      36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000       4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000       4K rw---    [ anon ]
b7e7c000       4K rw---    [ anon ]
b7e7d000    1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000      20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000       8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000      12K rw---    [ anon ]
b7fbb000      16K rw---    [ anon ]
b7fbf000       4K r-x--    [ anon ]
b7fc0000      84K r-x--  /lib/ld-2.3.6.so
b7fd5000       8K rw---  /lib/ld-2.3.6.so
bf8a0000      88K rw---    [ stack ]
```

tras asignarse 100M de memoria . . .

- en solaris

```
08046000       8K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000   97672K rwx--    [ heap ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000      36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000       4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000     136K rw---    [ anon ]
b1f1d000   97664K rw---    [ anon ]
b7e7d000    1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000      20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000       8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000      12K rw---    [ anon ]
b7fbb000      16K rw---    [ anon ]
b7fbf000       4K r-x--    [ anon ]
b7fc0000      84K r-x--  /lib/ld-2.3.6.so
b7fd5000       8K rw---  /lib/ld-2.3.6.so
bf8a0000      88K rw---    [ stack ]
```

tras crear una zona de memoria compartida . . .

- en solaris

```
08024000     144K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000   97672K rwx--    [ heap ]
D1693000     236K rwxs-    [ shmid=0xe ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000     36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000      4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000    136K rw---    [ anon ]
b1ee2000    236K rw-s-    [ shmid=0x3c800f ]
b1f1d000  97664K rw---    [ anon ]
b7e7d000   1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000     20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000      8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000     12K rw---    [ anon ]
b7fbb000     16K rw---    [ anon ]
b7fbf000      4K r-x--    [ anon ]
b7fc0000     84K r-x--  /lib/ld-2.3.6.so
b7fd5000      8K rw---  /lib/ld-2.3.6.so
bf891000    148K rw---    [ stack ]
```

tras mapear un fichero de 152739840 bytes . . .

- en solaris

```
08024000     144K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000   97672K rwx--    [ heap ]
C8400000  149160K r----  dev:102,7 ino:29125
D1693000     236K rwxs-    [ shmid=0xe ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000     36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000      4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000    136K rw---    [ anon ]
a8d38000 149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
b1ee2000    236K rw-s-    [ shmid=0x3c800f ]
b1f1d000  97664K rw---    [ anon ]
b7e7d000   1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000     20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000      8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000     12K rw---    [ anon ]
b7fbb000     16K rw---    [ anon ]
b7fbf000      4K r-x--    [ anon ]
b7fc0000     84K r-x--  /lib/ld-2.3.6.so
b7fd5000      8K rw---  /lib/ld-2.3.6.so
bf891000    148K rw---    [ stack ]
```

tras crear otra zona de memoria compartida ...

- en solaris

```
08024000     144K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000   97672K rwx--    [ heap ]
C6800000   27920K rwxs-    [ shmid=0xf ]
C8400000  149160K r----  dev:102,7 ino:29125
D1693000     236K rwxs-    [ shmid=0xe ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000      36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000       4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000     136K rw---    [ anon ]
a71f4000   27920K rw-s-    [ shmid=0x3e0010 ]
a8d38000  149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
b1ee2000     236K rw-s-    [ shmid=0x3c800f ]
b1f1d000   97664K rw---    [ anon ]
b7e7d000    1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000      20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000       8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000      12K rw---    [ anon ]
b7fbb000      16K rw---    [ anon ]
b7fbf000       4K r-x--    [ anon ]
b7fc0000      84K r-x--  /lib/ld-2.3.6.so
b7fd5000       8K rw---  /lib/ld-2.3.6.so
bf891000     148K rw---    [ stack ]
```

tras mapear otra vez el fichero anterior . . .

- en solaris

```
08024000     144K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000   97672K rwx--    [ heap ]
BD400000  149160K r----  dev:102,7 ino:29125
C6800000   27920K rwxs-    [ shmid=0xf ]
C8400000  149160K r----  dev:102,7 ino:29125
D1693000     236K rwxs-    [ shmid=0xe ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000     36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000      4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000    136K rw---    [ anon ]
9e04a000 149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
a71f4000  27920K rw-s-    [ shmid=0x3e0010 ]
a8d38000 149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
b1ee2000    236K rw-s-    [ shmid=0x3c800f ]
b1f1d000  97664K rw---    [ anon ]
b7e7d000   1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000     20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000      8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000     12K rw---    [ anon ]
b7fbb000     16K rw---    [ anon ]
b7fbf000      4K r-x--    [ anon ]
b7fc0000     84K r-x--  /lib/ld-2.3.6.so
b7fd5000      8K rw---  /lib/ld-2.3.6.so
bf891000    148K rw---    [ stack ]
```

tras ejecutar una función recursiva que asigna memoria un elevado nḿero de veces . . .

- en solaris

```
07AEE000    5480K rwx--    [ stack ]
08050000      44K r-x--  /home/antonio/so2tex/Practicas/a.out
0806A000       4K rwx--  /home/antonio/so2tex/Practicas/a.out
0806B000  102752K rwx--    [ heap ]
BD400000  149160K r----  dev:102,7 ino:29125
C6800000   27920K rwxs-    [ shmid=0xf ]
C8400000  149160K r----  dev:102,7 ino:29125
D1685000       4K rwxs-    [ anon ]
D1693000     236K rwxs-    [ shmid=0xe ]
D16D0000       4K rwx--    [ anon ]
D16E0000     740K r-x--  /lib/libc.so.1
D17A9000      24K rw---  /lib/libc.so.1
D17AF000       8K rw---  /lib/libc.so.1
D17C0000      24K rwx--    [ anon ]
D17CA000     132K r-x--  /lib/ld.so.1
D17FB000       4K rwx--  /lib/ld.so.1
D17FC000       8K rwx--  /lib/ld.so.1
```

- en linux

```
08048000     36K r-x--  /home/antonio/so2tex/Practicas/a.out
08051000      4K rw---  /home/antonio/so2tex/Practicas/a.out
08052000   5152K rw---    [ anon ]
9e04a000 149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
a71f4000  27920K rw-s-    [ shmid=0x3e0010 ]
a8d38000 149160K r----  /home/antonio/so2tex/Practicas/fichero.tar
b1ee2000    236K rw-s-    [ shmid=0x3c800f ]
b1f1d000  97664K rw---    [ anon ]
b7e7d000   1180K r-x--  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa4000     20K r----  /lib/tls/i686/cmov/libc-2.3.6.so
b7fa9000      8K rw---  /lib/tls/i686/cmov/libc-2.3.6.so
b7fab000     12K rw---    [ anon ]
b7fbb000     16K rw---    [ anon ]
b7fbf000      4K r-x--    [ anon ]
b7fc0000     84K r-x--  /lib/ld-2.3.6.so
b7fd5000      8K rw---  /lib/ld-2.3.6.so
bf35b000   5484K rw---    [ stack ]
```

memoria en System V R4

- Según el tipo de segmento (*vnode* o anónimo) en la estructura segvn_data hay la referencia adecuada
  - segmento *vnode*. Es un segmento asociado con un fichero que tiene un *vnode*. Se trata de código o de un fichero sobre el que se ha hecho *mmap()*. El miembro *vp* de la estructura *segvn_data* tiene esta referencia
  - segmento anónimo. Segmento no asociado a un fichero, p.e. datos o pila. Está descrito por una estructura *anon_map*. Las páginas anónimas de este segmento ocupan direcciones virtuales contiguas y sus descriptores se mantienen en el array *anon* de estas estructuras

memoria en System V R4: estructura *as*

La información mas relevante en la estructura *as* es

- primer segmento (región) en el espacio de direcciones (`struct seg *a_segs`)
- último segmeto referenciado (`struct seg *a_seglast`)
- tamaño espacio virtual (`size_t a_size`)
- memoria fisica usada (`size_t a_rss`)
- informacio traducción direcciones (hardware address translation) (`struct hat a_hat`)

Ejemplo struct as en OpenSolaris

```
struct as {
   kmutex_t a_contents;  /* protect certain fields in the structure */
   uchar_t  a_flags;     /* as attributes */
   uchar_ta_vbits;       /* used for collecting statistics */
   kcondvar_t a_cv;      /* used by as_rangelock */
   structhat *a_hat;     /* hat structure */
   structhrmstat *a_hrm; /* ref and mod bits */
   caddr_ta_userlimit;   /* highest allowable address in this as */
   struct seg *a_seglast;/* last segment hit on the addr space */
   krwlock_t a_lock;     /* protects segment related fields */
   size_ta_size;         /* size of address space */
   struct seg *a_lastgap;/* last seg found by as_gap() w/ AS_HI (mmap) */
   struct seg *a_lastgaphl; /* last seg saved in as_gap() either for */
                          /* AS_HI or AS_LO used in as_addseg() */
   avl_tree_t a_segtree;  /* segments in this address space. (AVL tree) */
   avl_tree_t a_wpage;    /* watched pages (procfs) */
   uchar_ta_updatedir;    /* mappings changed, rebuild a_objectdir */
   timespec_t a_updatetime;/* time when mappings last changed */
   vnode_t**a_objectdir;   /* object directory (procfs) */
   size_ta_sizedir;        /* size of object directory */
   struct as_callback *a_callbacks; /* callback list */
   void *a_xhat;          /* list of xhat providers */
   proc_t*a_proc;         /* back pointer to proc */
};
```

memoria en System V R4: estructura *seg*

La estructura *seg* describe los segmentos del espacio de direcciones del proceso

- direccion virtual cominezo del segmento (`addr_t s_base`)
- tamaño del segmento (`unsigned s_size`)
- referencia a la estructura *as* que contiene este segmento (`struct as *s_as`)
- punteros para manterner la lista de estructuras *seg* que configuran el espacio de direcciones del proceso (`struct seg *s_next,*s_prev`)
- array de punteros a las posibles operaciones sobre el segmento (`struct seg_ops *s_ops`)
- referencia a la estructura segvn_data con los detalles específicos de este segmento (`void *s_data`)

Ejemplo struct seg en OpenSolaris

```
struct seg {
  caddr_ts_base;    /* base virtual address */
  size_ts_size;     /* size in bytes */
  uint_ts_szc;      /* max page size code */
  uint_ts_flags;    /* flags for segment, see below */
  structas *s_as;   /* containing address space */
  avl_node_t s_tree;/* AVL tree links to segs in this as */
  structseg_ops *s_ops;/* ops vector: see below */
  void *s_data;     /* private data for instance */
};
```

memoria en System V R4: estructura *segvn_data*

- la estructura *segvn_data*, accesible desde la estructura *seg* contiene los detalles específicos del segmento
  - protección del segmento (miembro `unsigned char prot`): indica qué accesos están permitidos al segmento (lectura escritura o ejecución)
  - protección de las páginas (miembro `unsigned char pageprot`): indica si deben comprobarse los bits de protección de cada página
  - indicación de si se ha reservado *swap* para este segmento `size_t swresv`
  - tipo de compartición del segmento (miembro `type`: `MAP_PRIVATE, MAP_SHARED`)
  - si el segmento es un segmento asociado a un vnodo, una referencia a dicho vnodo, asi como el offset, figuran en esta estructura en los miembros `vp` y `offset`
  - si el segmento está formado por páginas anónimas la información relativa a dichas páginas esta accesible a través de `struct anon_map *amp`

Ejemplo struct segvn_data en OpenSolaris

```
typedef struct  segvn_data {
  krwlock_t lock;          /* protect segvn_data and vpage array */
  kmutex_t segp_slock;     /* serialize insertions into seg_pcache */
  uchar_t pageprot;        /* true if per page protections present */
  uchar_t prot;            /* current segment prot if pageprot == 0 */
  uchar_t maxprot;         /* maximum segment protections */
  uchar_t type;            /* type of sharing done */
  u_offset_t offset;       /* starting offset of vnode for mapping */
  struct  vnode *vp;       /* vnode that segment mapping is to */
  ulong_t anon_index;      /* starting index into anon_map anon array */
  struct  anon_map *amp;   /* pointer to anon share structure, if needed */
  struct  vpage *vpage;    /* per-page information, if needed */
  struct  cred *cred;      /* mapping credentials */
  size_t  swresv;   /* swap space reserved for this segment */
  uchar_t advice;          /* madvise flags for segment */
  uchar_t pageadvice;      /* true if per page advice set */
  ushort_t flags;          /* flags - from sys/mman.h */
  ssize_t softlockcnt;     /* # of pages SOFTLOCKED in seg */
  lgrp_mem_policy_info_t policy_info; /* memory allocation policy */
} segvn_data_t;
```

memoria en System V R4: estructura *anon_map*

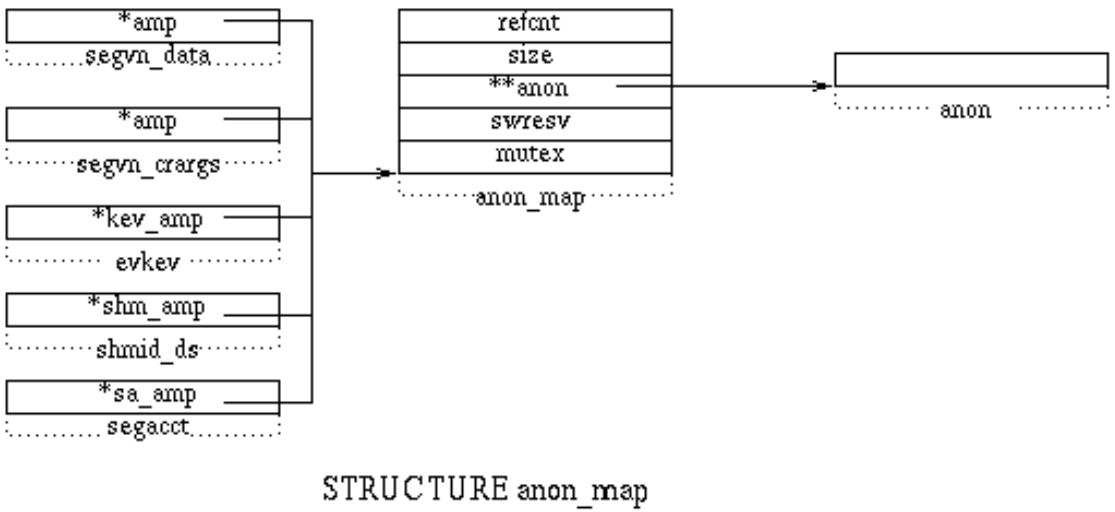
- la estructura *anon_map* permite al sistema operativo encontrar las páginas asociadas a un segmento dado
- la estructura *anon_map* contiene, entre otras cosas)
  - un contador de referencias
  - el tamaño del espacio mapeado por dicha estructura
  - array de punteros a la estructuras que representan cada página
  - indicador de espacio de intercambio reservado
- se utiliza una estructua *anon* para cada página de un segmento constituido por páginas anónimas

memoria en System V R4: estructura *anon*

- en la estructuta *anon_map* hay un array de punteros a estructuras *anon*. Elementos consecutivos de este array mapean bloques de direcciones virtuales consecutivas
- los elementos mas relevantes de la estructura *anon* son
  - contador de referencias `int an_refcnt`
  - referencia a la estructura *page* que describe la página `struct page *an_page`
  - puntero para mantener la lista de estructuras *anon* libres `struct anon *an_next`
  - puntero a la array de estructuras anon `struct anon *an_bap`

Ejemplo struct anon_map en Unixware 7

```
struct anon_map{
  u_int refcnt;          /*reference count on this structure */
  u_size;                /* size in bytes mapped by the anon array */
  struct anon **anon;    /* pointer to an array of anon * pointers */
  u_int swresv;          /* swap space reserved for this anon_map */
  struct simplelock mutex; /* Multiprocessing lock for segment manipulation */
};
```



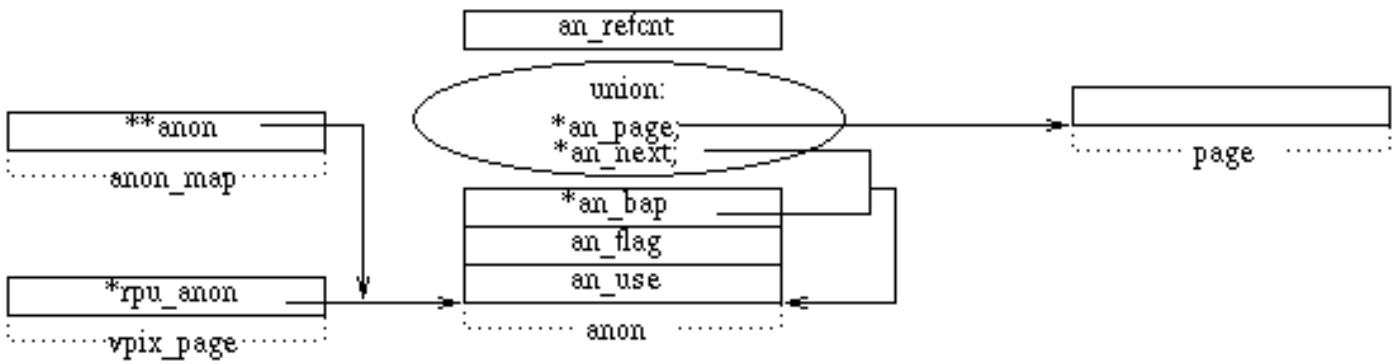
STRUCTURE anon_map

Ejemplo struct anon_map en OpenBSD

```
struct vm_amap {
        simple_lock_data_t am_l; /* simple lock [locks all vm_amap fields] */
        int am_ref;              /* reference count */
        int am_flags;            /* flags */
        int am_maxslot;          /* max # of slots allocated */
        int am_nslot;            /* # of slots currently in map ( <= maxslot) */
        int am_nused;            /* # of slots currently in use */
        int *am_slots;           /* contig array of active slots */
        int *am_bckptr;          /* back pointer array to am_slots */
        struct vm_anon **am_anon; /* array of anonymous pages */
#ifdef UVM_AMAP_PPREF
        int *am_ppref;           /* per page reference count (if !NULL) */
#endif
};
```

Ejemplo struct anon en Unixware 7

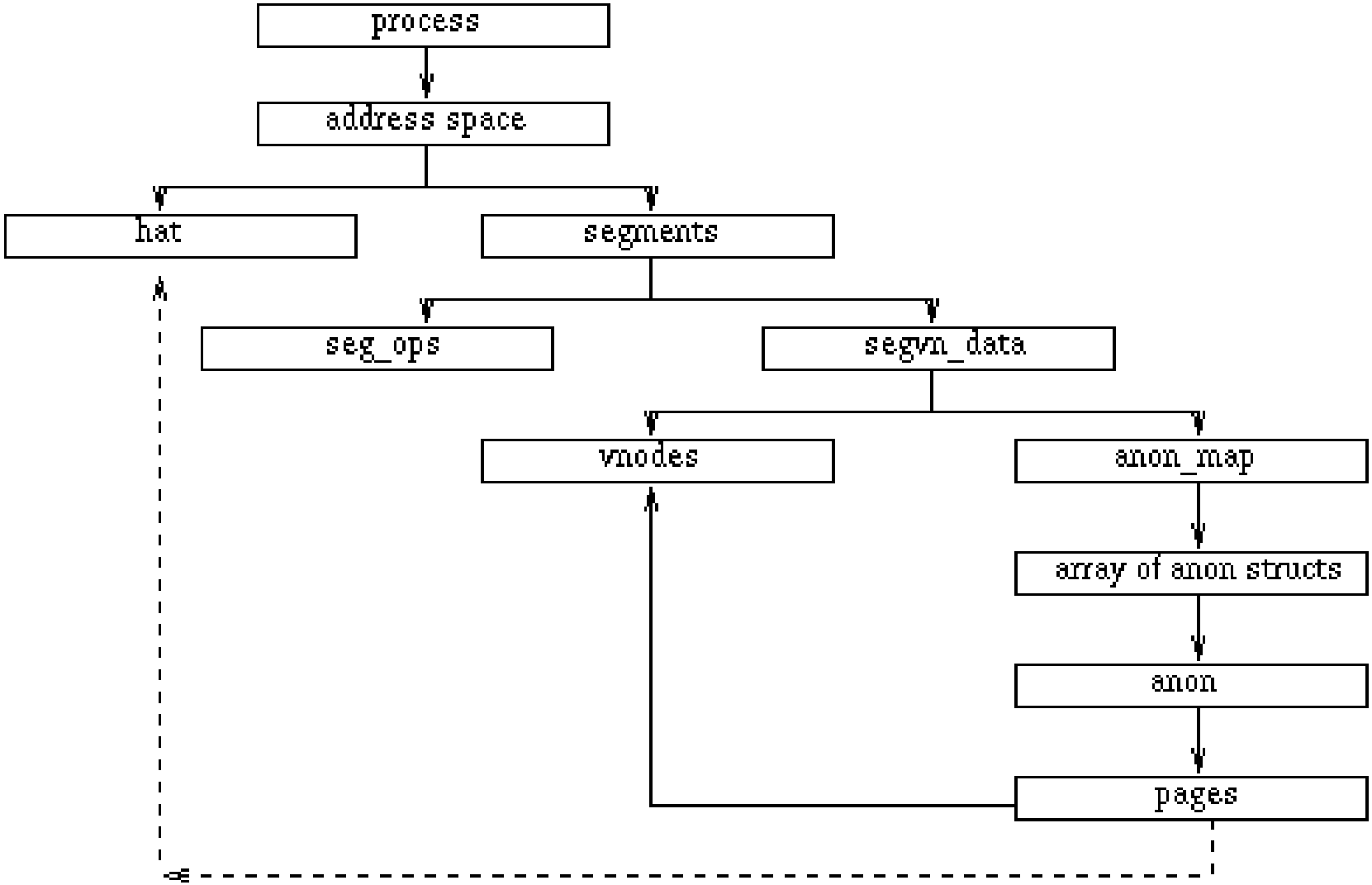
```
struct anon {
  u_int an_refcnt;                /* reference count */
  union {
         struct  page *an_page;   /* ‘‘hint’’ to the real page */
         struct  anon *an_next;   /* free list pointer */
        } un;                     /* union of page and anon */
  struct anon *an_bap;            /* pointer to real anon */
  int an_flag;                     /* an_flag values */
  int an_use;                    /* used for debuggin */
};
```



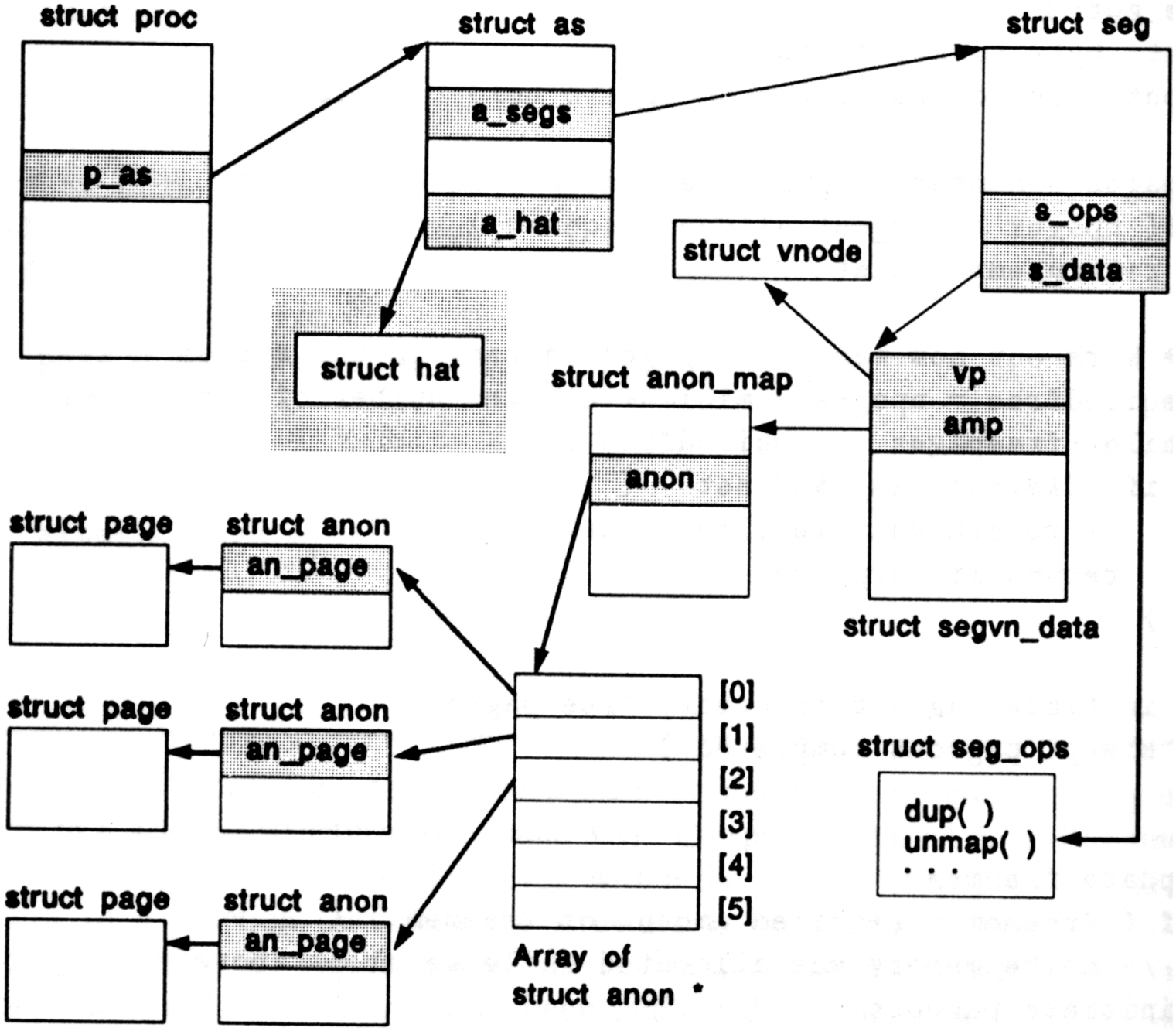
Structure anon

Ejemplo struct anon openBSD

```
struct vm_anon {
        int an_ref;                         /* reference count [an_lock] */
        simple_lock_data_t an_lock;         /* lock for an_ref */
        union {
                struct vm_anon *an_nxt; /* if on free list [afreelock] */
                struct vm_page *an_page;/* if in RAM [an_lock] */
        } u;
        int an_swslot;              /* drum swap slot # (if != 0)
                                       [an_lock.  also, it is ok to read
                                       an_swslot if we hold an_page PG_BUSY] */
};
```

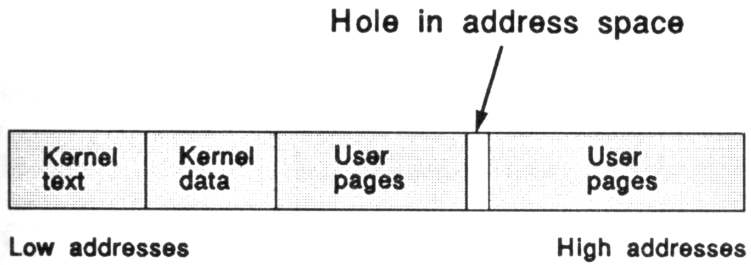


VIRTUAL MEMORY SYSTEM ARCHITECTURE

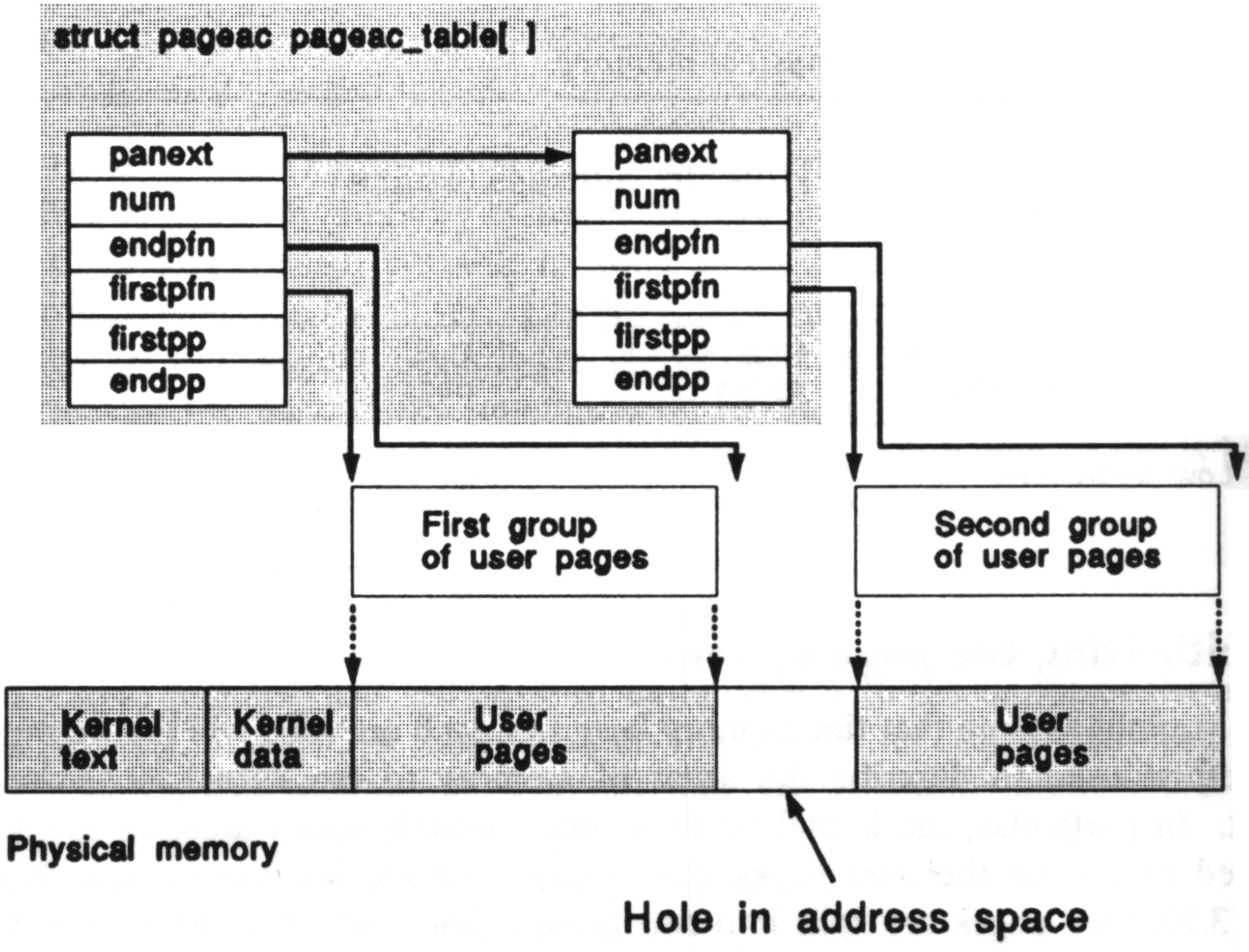

struct proc
p_as
struct as
a_segs
a_hat
struct hat
struct seg
s_ops
s_data
struct vnode
vp
amp
struct anon_map
anon
struct segvn_data
struct page
struct anon
an_page
struct page
struct anon
an_page
struct page
struct anon
an_page
[0]
[1]
[2]
[3]
[4]
[5]
Array of
struct anon *
struct seg_ops
dup( )
unmap( )
. . .

memoria física

- la memoria física se utiliza
  - código del kernel
  - datos del kernel
  - datos dinámicos del kernel: en SVR4 muchas de la estructuras que en otras versiones eran un array, se asignan dinamicamente: p.e. las estructuras `proc`. (*kmem_alloc()*, *kmem_free()*)
  - páginas de usuario



- El código y los datos de kernel no se paginan.
- La disposición de la memoria física esta descrita por un array de estructuras `page_ac` llamado `pageac_table[]`. (para una disposición de memoria como la de la página siguiente el array tendría dos elementos)


struct pageac pageac_table[ ]
panext
num
endpfn
firstpfn
firstpp
endpp
panext
num
endpfn
firstpfn
firstpp
endpp
First group of user pages
Second group of user pages
Kernel text
Kernel data
User pages
User pages
Physical memory
Hole in address space

struct pageac

```
struct pageac{
   struct pageac *panext; /*siguiente*/
   unsigned num;    /*paginas controladas por esta tabla*/
   unsigned firstpfn; /*marco de la primera pagina en el area*/
   unsigned endpfn;   /*ultimo +1*/
   struct page *firstpp; /*puntero a la estructura page */
                         /*de la primera pagina del area*/
   struct page *endpp;   /*puntero a la ultima*/
};                       /*firstpp +num*/
```

- Para cada página de la memoria física hay una estructura *page* en la memoria del kernel que contiene información de esa página.

struct page

La información contenida en la estructura *page* que describe cada página es la siguiente:

- **p_lock** página ocupada, hay una operación en curso
- **p_want** hay algún proceso en espera por esa página
- **p_free** está libre, en la FREELIST
- **p_intrans** pendiente de operación e/s. No puede ser hurtada al proceso
- **p_gone** el proceso la ha liberado pero todavía no esta el la FREELIST
- **p_mod,p_ref** copia de los bits de modificada y referenciada
- **p_pagein** debe traerse de memoria secundaria
- **p_age** usado por *paged*

struct page

- **p_vnode** si la página corresponde a código o un fichero mapeado, puntero al *inodo* de este
- **p_hash** para encontrar rápidamente las páginas asoniadas a un *vnodo*
- **p_offset** *offset* en el vnodo
- **p_next,p_prev** siguiente y anterior en la FREELIST
- **p_vpnext,p_vpprev** siguiente y anterior en la lista vnodo
- **p_mapping** información de HAT
- **p_keepcont** procesos que usan esta página

`struct page`

- cada página puede estar en una de las siguientes listas
  - `lista hash vnode`: las páginas asociadas a un *vnodo* se manienen en una lista donde están encadenadas todas la asociadas a un *vnodo* y en una lista *hash* para que en caso de fallo de página el acceso sea más rápido
  - `FREE LIST`: libres y que pueden se asignadas a los procesos. Son puestas por el proceso de robo de páginas o por el kernel o cuando un proceso termina
  - `cache list`: como la FREE LIST, salvo que sus contenidos son válidos y pueden ser recuperadas. Son puestas por el proceso del robo de páginas
- las páginas de código o de un fichero mapeado tienen asociación con un *vnodo*, son *vnode pages*. El sistema no les asigna intercambio
- las páginas que no tienen asociación con un *vnodo* se llaman anónimas (p.e. datos y pila). Se asocian al *vnodo* del dispositivo de intercambio.

paged

- *paged* es el proceso que se encarga de *robar* páginas a los procesos
- la periodicidad con que se despierta depende de la cantidad de memoria libre en el sistema
  - si la memoria libre es mayor que el valor *lotsfree*, no se despierta
  - si la memoria libre es menor que *desfree*, *paged* se despierta con cada ciclo de reloj. Por defecto *desfree* suele valer 6.25% de la memoria fisica.
  - si la memoria libre esta comprendida entre *desfree* y *lotsfree*, *paged* se despierta 4 veces por segundo.

paged

- *paged* examina las páginas circularmente con 2 índices: *fronthand* y *backhand*
  - *fronthand*. Si la página examinada con este índice tiene el bit de referencia se lo quita, si no tiene el bit de referencia es liberada
  - *backhand*. Si la página examinada con este índice no tiene el bit de referencia es liberada
- las páginas que son referenciadas desde que se examinan con *fronthand* hasta que se examinan con *backhand* permanecen en memoria

paged

- el número de páginas que *paged* examina en cada ejecución tambien está controlado por unos parámetros
  - si la memoria libre es *lotsfree*, *paged* examina *slowscan* páginas cada vez
  - si la memoria libre es *desfree*, *paged* examina *fastscan* páginas cada vez
  - si la memoria libre esta comprendida entre *desfree* y *lotsfree* *paged* interpola linealmente entre *slowscan* y *fastscan* el número de páginas que tiene que examinar
- Cada vez que es liberada una página modificada es escrita a disco (dispositivo *swap*). el parámetro *maxpgio* limita el número de páginas que paged escribe a disco en cada ejecución

paged

RESUMIENDO

- la frecuencia con que se despierta es ejecuta *paged* depende de la cantidad de memoria libre en el sistema. Si mayor que *lotsfree* no se despierta, si menor que *desfree* se despierta con cada ciclo de reloj, en caso contratio 4 veces por segundo
- cada vez que se despierta *paged* escanea un número de páginas comprendido entre *slowscan* y *fastscan* dependiendo de la cantidad de memoria libre (comprendida entre *lotsfree* y *desfree*)
- las páginas que son referenciadas desde que son examinadas con *fronthand* hasta que se examinan con *backhand* permanecen en memoria
- *paged* velve a espera cuando ya ha examinado *desscan* páginas o ya ha hecho escribir a disco *maxpgio* páginas.
- *paged* se crea cuando el sistema arranca, tiene tipicamente el pid 2 y corre siempre en modo kernel

sched

```
bash-2.05$ ps -lp 0,2
 F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY       TIME CMD
19 T     0     0     0  0   0 SY        ?      0        ?        0:01 sched
19 S     0     2     0  0   0 SY        ?      0        ? ?      0:00 pageout
```

- Es posible que *paged* no pueda evitar la hiperpaginacion, en ese caso se utiliza el proceso *sched*, que tiene pid 0
- si la cantidad de memoria libre desciende por debajo de GPGSLO *paged* llama a *sched*
- *sched* selecciona utiliza la función CL_SWAPOUT() para seleccionar un proceso para intercambiar y libera todas las páginas de memoria asociadas a el. En la estructura *proc* del proceso indica que no es ejegible para ser ejecutado
- cuando la cantidad de memoria libre supera GPGSLO se vuelve a ejecutar *sched* y utiliza CL_SWAPIN() para seleccionar un proceso para traer de nuevo a memoria
- los procesos en tiempor real no son nunca seleccionados, los procesos seleccionables son
  - los de menor prioridad
  - los que están en espera o parados