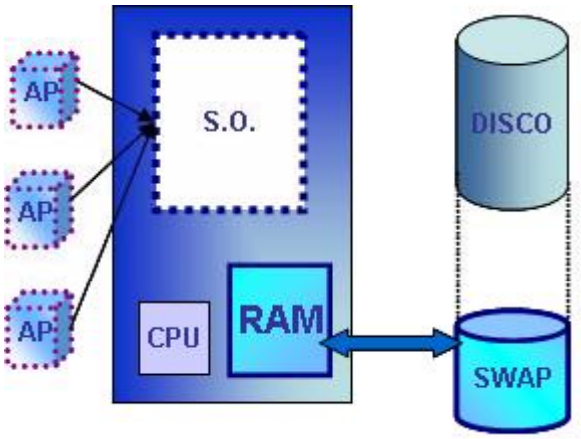
SWAP

Juan A. Fernández Sagasti
Deriman Franco García

Espacio de Intercambio "Swapping"

- SWAP: Espacio en disco (fichero o partición) para páginas no mapeadas en memoria.

- SWAPPING: Intercambio de páginas de memoria por otras que no lo están.



Tipos de páginas manejadas

- Pila (Stack): páginas pertenecientes a la pila de un proceso.
- “Dirty”: páginas de un proceso que han sido modificadas.
- Compartidas: páginas de memoria compartidas entre procesos.

Funciones principales:

- **Crear las zonas swap** en el disco, **activarlas** y **desactivarlas.**
- **Manejar el espacio swap** para almacenar y liberar marcos de página.
- **Proporcionar las funciones para el intercambio** con memoria principal (*swap in, swap out)*
- Saber **en qué parte del swap están las páginas** intercambiadas. Esto se consigue gracias a los **swapped-out page identifiers.**

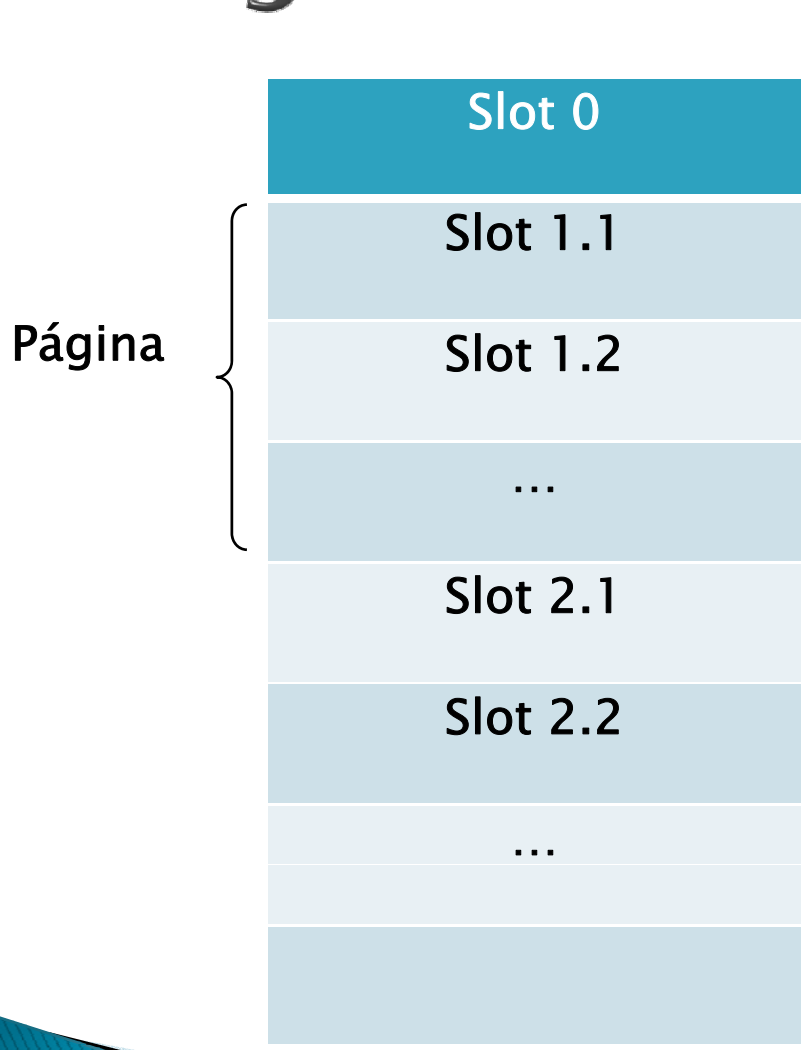
¿Es necesario el espacio SWAP?

- Espacio Swap visto como RAM adicional.
- Aún con mucha RAM es recomendable siempre crear Swap.
- Posibilita desalojar procesos poco activos.
- Regla v1.0: SWAP=2*RAM (en desuso)
- Regla v1.1: SWAP=RAM deseada-RAM real

Área SWAP

- Partición del disco o fichero.
- Máximo de áreas: MAX_SWAPFILES(generalmente a 32)
- Una zona Swap es una secuencia de slots.
- Primer slot ⇒ información sobre el Swap.

Diagrama: Área Swap



swap_header union
- Info
- Magic

Info
- Bootbits //1024 octetos
- Versión // algoritmo intercambio
- Last_page
- Nr_badpages //slots defectuosos
- Padding[125]:
- Badpages[1] //pág. defectuosas

Magic
- magic
- reserved

Proporciona una marca que indica que parte del disco se usa como zona swap. Permite al núcleo identificar un archivo o una partición como zona swap.

Crear y activar una zona swap

- Comandos: *mkswap*
  - *Inicializa los campos de swap_header.*
  - *Queda en un estado inactivo.*

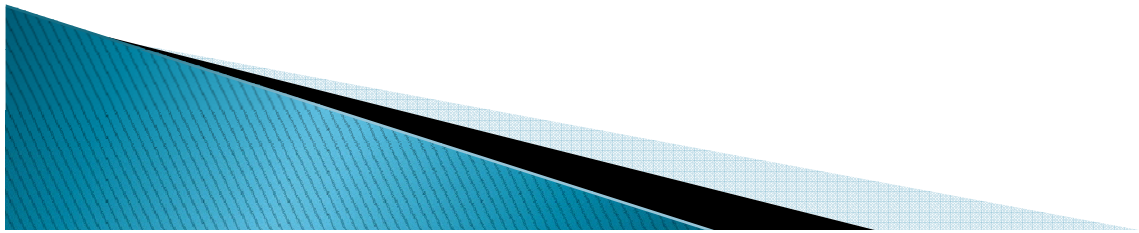
- *Una zona Swap puede activarse con un script en el arranque o dinámicamente cuando el sistema está funcionando.*

Prioridades Swap

- Cada dispositivo swap tiene asociada una prioridad.
- Cuando se desaloja una página de memoria y existen varios dispositivos swap activos se almacenará en aquel con mayor prioridad.
- Si varios dispositivos tienen la misma prioridad se distribuye usando Round-Robin.

Descriptores de zona Swap

- El kernel guarda una lista de dispositivos swap activos (*swap_info*).
- La estructura *swap_info_struct* define el formato de los descriptores almacenados en la lista *swap_info*.
- Se encuentra declarado en swap.h

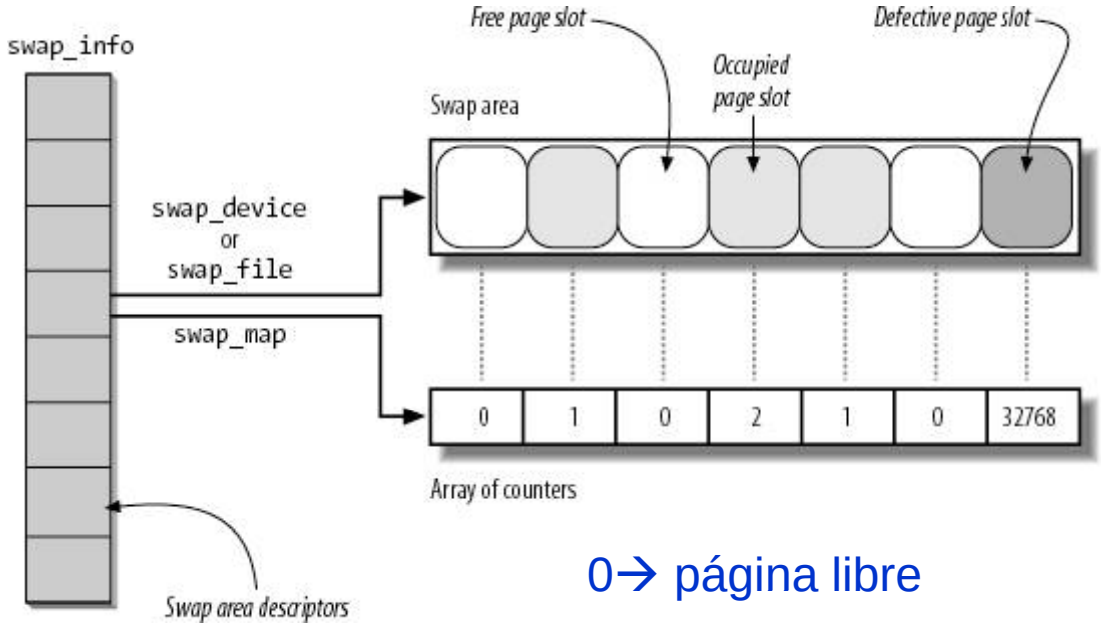


Descriptores de zona Swap

Tipo	Campo	Descripción
unsigned int	**flags**	banderas del área swap
struct file *	**swap_file**	Puntero al fichero o dispositivo que almacena el área swap
unsigned short *	**swap_map**	puntero a un vector de contadores, uno por cada slot
int	**prio**	prioridad del área swap
int	**pages**	número de slots de páginas utilizables(no cuenta primer slot, ni defectuosos)
unsigned long	**Max**	tamaño del área swap en páginas
unsigned long	**inuse_pages**	número de slots de páginas usadas en el área swap

Descriptores de zona Swap

- El vector swap_info incluye descriptores de la zona swap.



0→ página libre

>0→ página ocupada

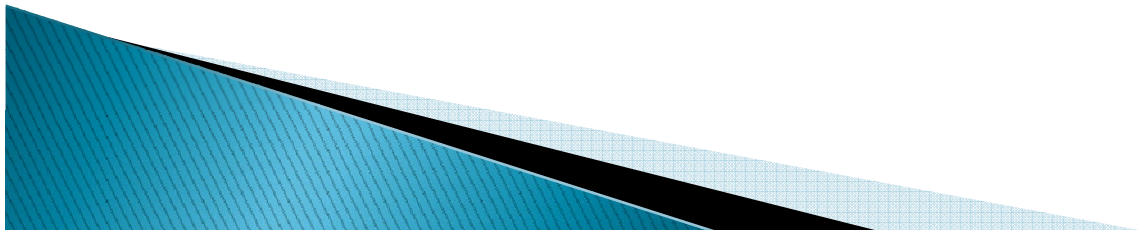
32767→ pág. permanente

32768→ pág. defectuosa

Descriptores de las zonas swap

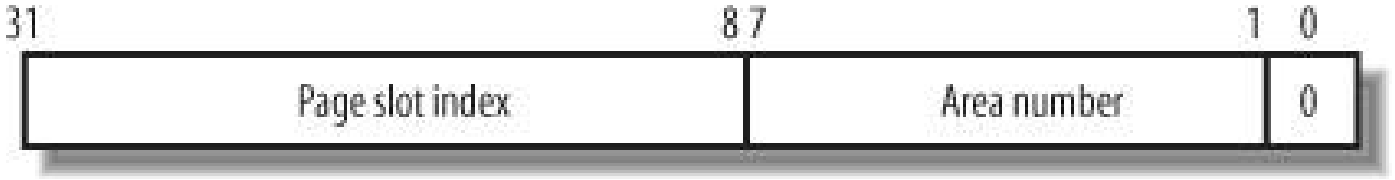
- Los descriptores de las zonas de swap activas también se insertan en **una lista ordenada por prioridad.**

- Swap_list del tipo swap_list_t contiene:
  - Head: primer elemento de la lista en el swap_info
  - Next: índice del vector swap_info de la zona swap siguiente en prioridad.

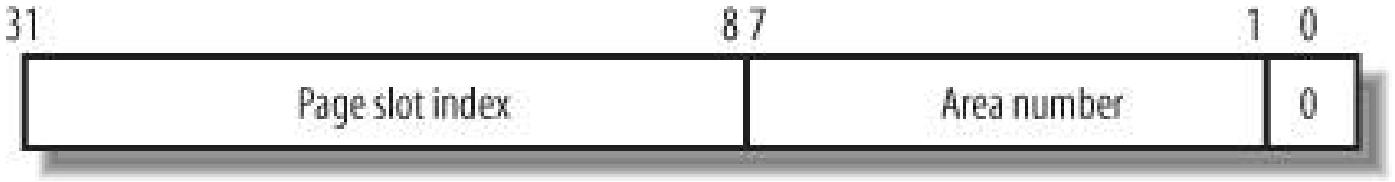


Identificador de página descargada

- El identificador de página descargada se almacena en la tabla de páginas
- Tipos de entradas en la tabla:
  - Nula
  - Almacenada en SWAP (LSB=0)
  - Contenida en RAM (LSB=1)



Identificador de página descargada



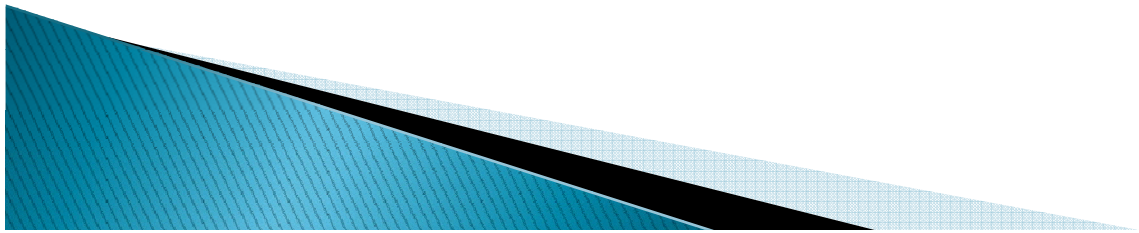
- Slot: Primer slot de página (excepto slot 0)
- Área swap
- LSB: Presencia o no en RAM

Identificador de página descargada

- Swp_entry(type, offset)
  - Crea un identificador
- Swp_type
  - Extrae el área swap del identificador
- Swp_offset
  - Extrae el primer slot del identificador

Activar y desactivar una zona swap

- sys_swapon()
  - Activa una zona swap.
- sys_swapoff()
  - Desactiva una zona swap. Cuando se desactiva todas las páginas se vuelven a cargar en memoria.



sys_swapon(specialfile,swap_flags)

- Busca un hueco libre.

```
1495        p = swap_info;
1496        for (type = 0 ; type < nr_swapfiles ; type++,p++)
1497                if (!(p->flags & SWP_USED))
1498                        break;
1499        error = -EPERM;
1500        if (type >= MAX_SWAPFILES) {
1501                spin_unlock(&swap_lock);
1502                goto out;
1503        }
1504        if (type >= nr_swapfiles)
1505                nr_swapfiles = type+1;
```

sys_swapon(specialfile,swap_flags)

- En la estructura swap_info_struct se marca la bandera que indica que la zona swap está activa y se inicializa el descriptor de la zona swap.

- 1508 p->flags = SWP_USED;
- 1509 p->next = -1;

sys_swapon(specialfile,swap_flags)

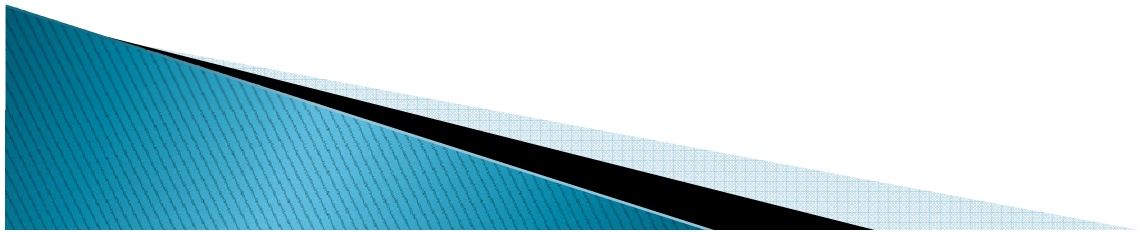
- La función, tras comprobar posibles errores del dispositivo pasado en specialfile, rellena la estructura swap_header almacenada en el slot 0 del dispositivo swap cuando se creó (mkswap).

```
1599        if (swab32(swap_header->info.version) == 1) {
1600                swab32s(&swap_header->info.version);
1601                swab32s(&swap_header->info.last_page);
1602                swab32s(&swap_header->info.nr_badpages);
1603                for (i = 0; i < swap_header->info.nr_badpages; i++)
1604                        swab32s(&swap_header->info.badpages[i]);
1605        }
```

sys_swapon(specialfile,swap_flags)

- Finalmente se inserta debidamente en swap_list (lista por prioridades).

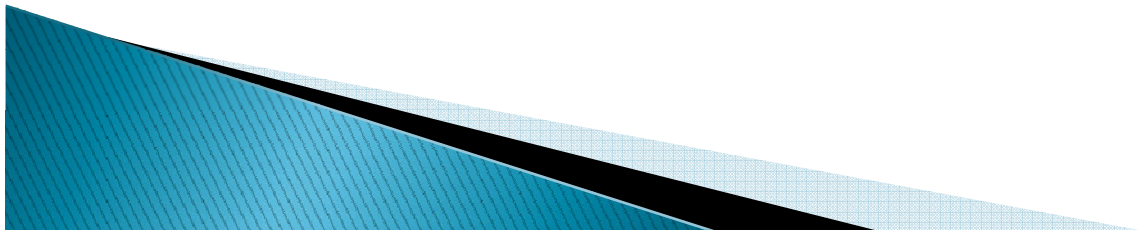
```
1710   for (i = swap_list.head; i >= 0; i = swap_info[i].next) {
1711        if (p->prio >= swap_info[i].prio) {
1712             break;
1713        }
1714             prev = i;
1715        }
1716        p->next = i;
```



sys_swapoff(specialfile)

- Comprueba si el specialfile se puede abrir. En ese caso busca descriptor de esa zona swap.

```
1254        for (type = swap_list.head; type >= 0;
type = swap_info[type].next) {
1255                p = swap_info + type;
1256                if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
1257                        if (p->swap_file->f_mapping == mapping)
1258                                break;
1259                }
1260                prev = type;
1261        }
```



sys_swapoff(specialfile)

- Si no lo encuentra devuelve un error.

```
1262        if (type < 0) {
1263                err = -EINVAL;
1264                spin_unlock(&swap_lock);
1265                goto out_dput;
1266        }
```

sys_swapoff(specialfile)

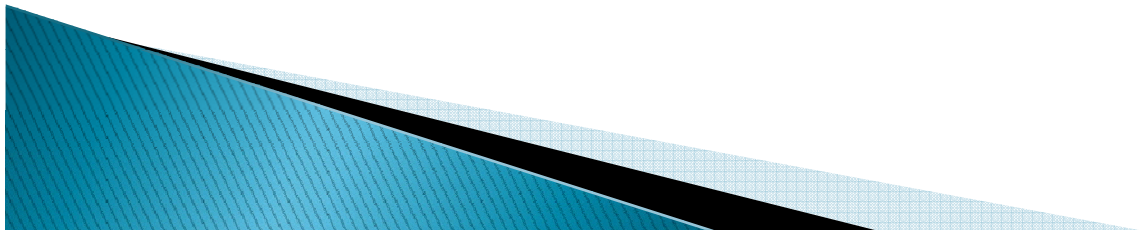
- Comprueba si se pueden almacenar todas las páginas en memoria.

```
1267        if (!security_vm_enough_memory(p->pages))
1268                vm_unacct_memory(p->pages);
1269        else {
1270                err = -ENOMEM;
1271                spin_unlock(&swap_lock);
1272                goto out_dput;
1273        }
```

sys_swapoff(specialfile)

- Se elimina de la lista swap_list la zona swap desactivada.

```
1274        if (prev < 0) {
1275                swap_list.head = p->next;
1276        } else {
1277                swap_info[prev].next = p->next;
1278        }
```



sys_swapoff(specialfile)

- Se restauran las páginas a memoria. De ello se encarga la función try_to_unuse.
- La función *TRy_to_unuse()* explora el vector *swap_map* de la zona swap. Cuando la función encuentra un slot de página en uso, intercambia (de swap a RAM) la página.

- 1294 err = try_to_unuse(type);

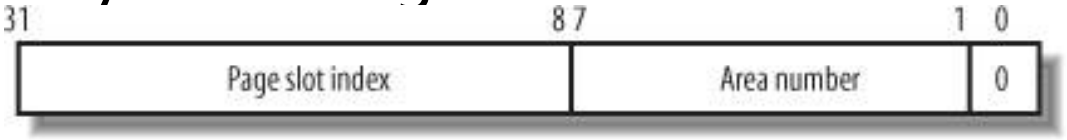
sys_swapoff(specialfile)

- Si hubo algún error en la elección del dispositivo a eliminar (estaba en uso por ejemplo), se vuelve a insertar en la lista en la posición que le correspondía.
- En caso de que todo sea correcto, se elimina ese descriptor.

```
1337        swap_file = p->swap_file;
1338        p->swap_file = NULL;
1339        p->max = 0;
1340        swap_map = p->swap_map;
1341        p->swap_map = NULL;
1342        p->flags = 0;
1343        spin_unlock(&swap_lock);
1344        mutex_unlock(&swapon_mutex);
1345        vfree(swap_map);
```

Swap_duplicate(entry)

- La función *swap_duplicate()* se invoca generalmente cuando se intenta intercambiar una página ya descargada al almacenamiento swap



- Validación del identificador de página

```
1787    type = swp_type(entry);     //extraer el número de zona swap
1788    if (type >= nr_swapfiles)
1789            goto bad_file;
1790    p = type + swap_info;
1791    offset = swp_offset(entry); // extraer el índice de slots de página
                                    para el parámetro
```

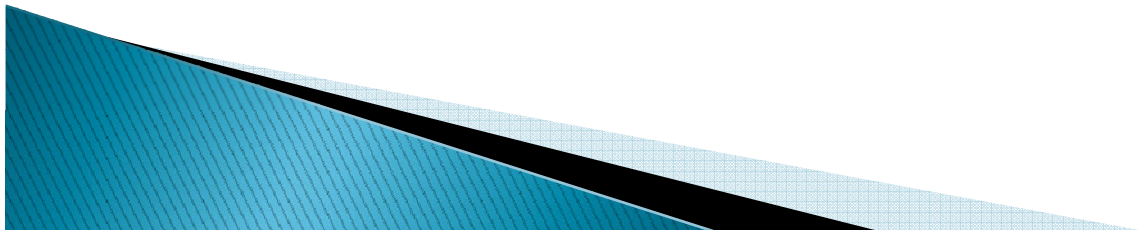
Swap_duplicate(entry)

- Comprueba si el slot de página es válido y no está libre (su contador *swap_map* es mayor de 0 y menor que *SWAP_MAP_BAD*);
- Si no, el identificador de página descargada al almacenamiento swap establece una página válida. Y aumenta el contador *swap_map* del slot de página si no ha alcanzado ya el valor *SWAP_MAP_MAX*.

```
1794        if (offset < p->max && p->swap_map[offset]) {
1795                if (p->swap_map[offset] < SWAP_MAP_MAX - 1) {
1796                        p->swap_map[offset]++;
1797                        result = 1;
1798                } else if (p->swap_map[offset] <= SWAP_MAP_MAX) {
1799                        if (swap_overflow++ < 5)
1800                                printk(KERN_WARNING "swap_dup: swap entry
overflow\n");
1801                        p->swap_map[offset] = SWAP_MAP_MAX;
1802                        result = 1;
1803                }
1804        }
```

Asignar y lanzar un slot de página

- Linux comienza una búsqueda de slot libre empezando siempre por el último que se asignó.
- La función scan_swap_map() se utiliza para buscar un slot libre en una zona swap dada.
- Devolverá el índice del slot libre ó 0 en el caso que no haya slots libres.



Scan_swap_map (si)

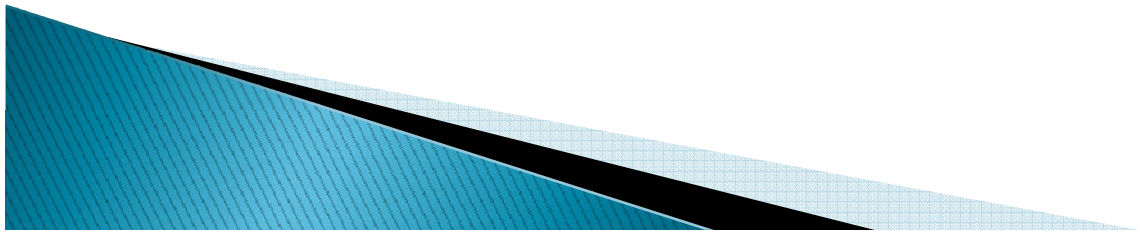
- Se intenta en el cluster actual y se comprueba si el cluster tiene suficiente capacidad.

```
105     if (unlikely(!si->cluster_nr)) {
106         si->cluster_nr = SWAPFILE_CLUSTER - 1;
107         if (si->pages - si->inuse_pages < SWAPFILE_CLUSTER)
108             goto lowest;
```

Scan_swap_map (si)

- Se busca el primer slot libre

```
115     for (; last_in_cluster <= si->highest_bit; offset++) {
116             if (si->swap_map[offset])
117                     last_in_cluster = offset + SWAPFILE_CLUSTER;
118             else if (offset == last_in_cluster) {
119                     spin_lock(&swap_lock);
120                     si->cluster_next = offset-SWAPFILE_CLUSTER+1;
121                     goto cluster;
122             }
```



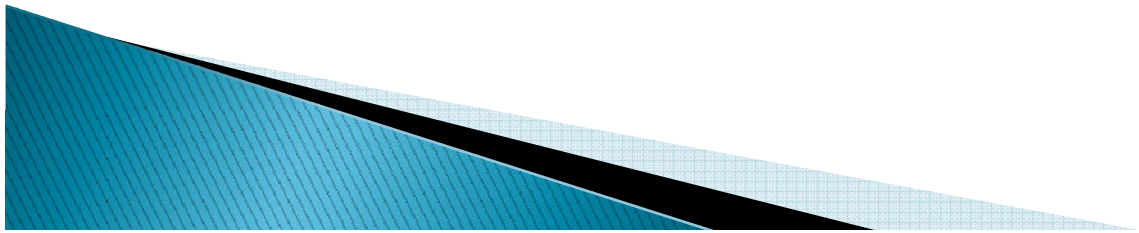
Scan_swap_map (si)

- Se decrementa el número de slots libres.

- 132 si->cluster_nr--;

- Si al buscar el slot, no se encontró ninguno libre probaremos empezando desde el lowest_bit.

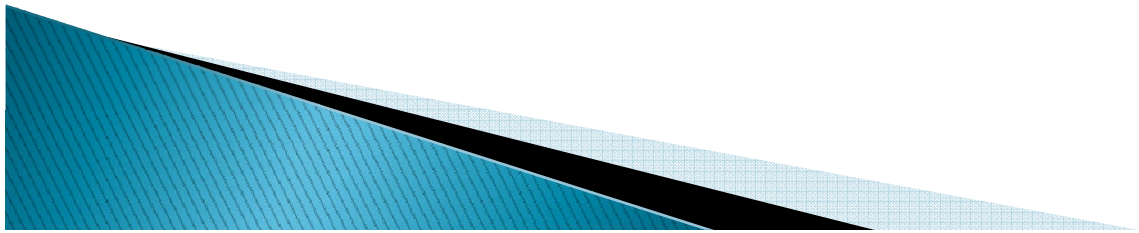
- 134 offset = si->cluster_next;
- 135 if (offset > si->highest_bit)
- 136lowest: offset = si->lowest_bit;



Scan_swap_map (si)

- Cuando se encuentre un slot libre, se actualizará la estructura swap_info_struct.

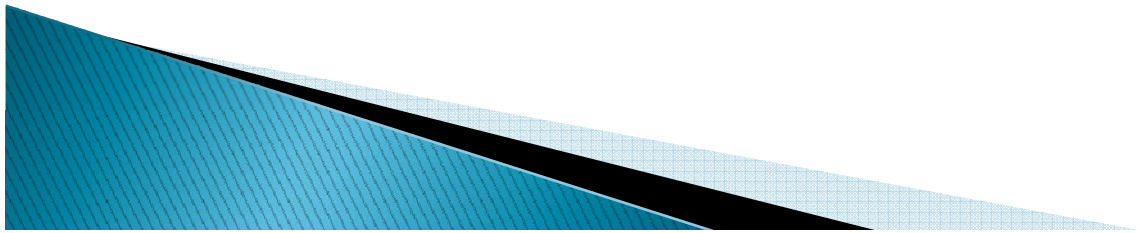
```
151     si->swap_map[offset] = 1;
152     si->cluster_next = offset + 1;
153     si->flags -= SWP_SCANNING;
154     return offset;
```



get_swap_page()

- La función *get_swap_page()* se utiliza para encontrar un slot de página libre buscando en todas las zonas de swap activas.

- Se tendrá en consideración las prioridades



get_swap_page()

- Se comprueba que existen áreas swap activas

```
184         if (nr_swap_pages <= 0)
185                 goto noswap;
```

get_swap_page()

- Se entra en un bucle que recorre la lista por prioridad en busca de un slot libre. Si no se encontrara, se devolverá cero.

```
188     for (type = swap_list.next; type >= 0 && wrapped < 2; type = next)
{
  [...]
202             swap_list.next = next;
203             offset = scan_swap_map(si);
204             if (offset) {
205                     spin_unlock(&swap_lock);
206                     return swp_entry(type, offset);
207             }
208             next = swap_list.next;
209     }
```

swap_free(entry)

- La función *swap_free()* es invocada cuando se intercambia una página para disminuir en el *swap_map* el contador correspondiente.

```
302     p = swap_info_get(entry);
303     if (p) {
304         swap_entry_free(p, swp_offset(entry));
```

RAM -> Swap

- Cache swap: Su función es controlar los accesos concurrentes para realizar una migración de una página.

- Se utilizan distintas funciones para manejarla.

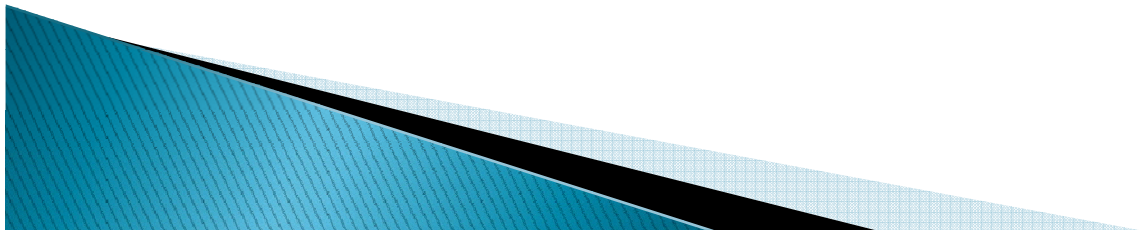
RAM -> Swap

- 1) Insertar el marco de página en la cache swap.
- 2) Actualizar las entradas de la tabla de páginas.
- 3) Escribir la página en el área de intercambio (swap área).
- 4) Borrado del marco de página de la cache swap.

Swap -> RAM

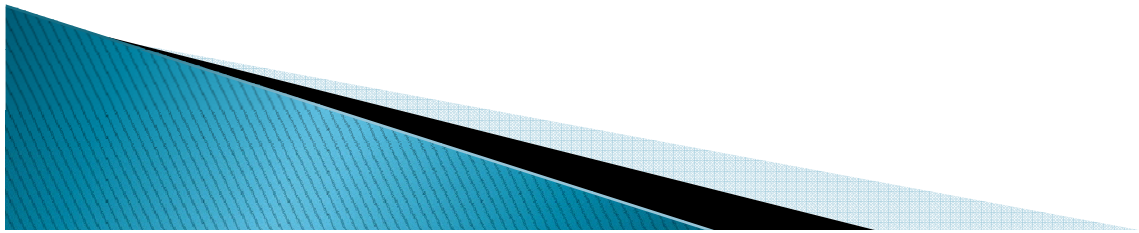
- El proceso de migración de las páginas del espacio swap a RAM ocurre cuando un proceso hace referencia a una página que ha sido intercambiada fuera del disco. El manejador de la excepción de fallo de página comienza la migración (de swap a RAM) .

- Esto se realiza con la función **do_swap_page()**



Swap->RAM

- 1)Se produce un fallo de página.
- 2)Se mira la tabla de página.
- 3)Se introduce en cache swap
- 4)Se trae de swap a RAM



Creación de un archivo de intercambio

- Creación del fichero de 64 MB

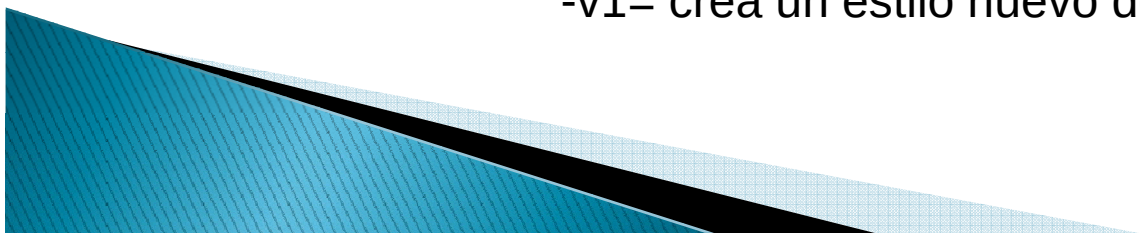
dd if=/dev/zero of=swapfile bs=1024 count=65536

if = input file
of = output file
bs = block size
count = nº de veces a multiplicar al valor de "bs"

- Dar formato al fichero

mkswap -v1 /mnt/w95/swapfile

-v1= crea un estilo nuevo de área swap



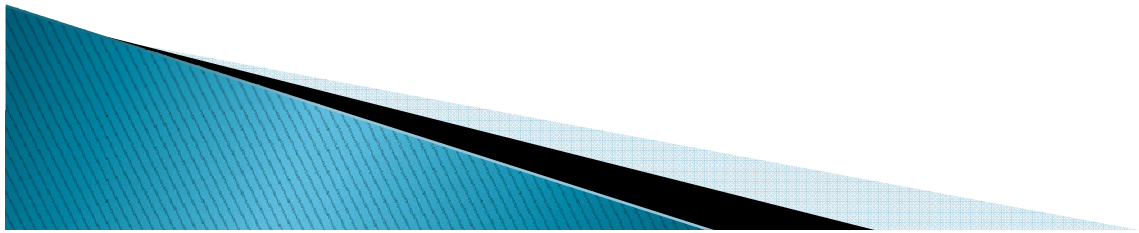
Creación de un archivo de intercambio:

- Activación del fichero ⇒ ***sys_swapon swapfile***
- Comprobación de funcionamiento

swapon -s

Filename	*Type*	*Size*	*Used*	*Priority*
/dev/sda1	*partition*	*40156*	*0*	*-1*
/root/Desktop/swapfile	*file*	*65532*	*0*	*-2*

- Desactivación del fichero ⇒ ***sys_swapoff swapfile***



Referencia Bibliográficas

- Maxvell(cap 8)
- Card (cap 8)
- Understanding Linux Kernel 3ª edición
- http://lxr.linux.no/