Swap -1- (Basic, initialization)

2019-10-29 (http://jake.dothome.co.kr/swap-1/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

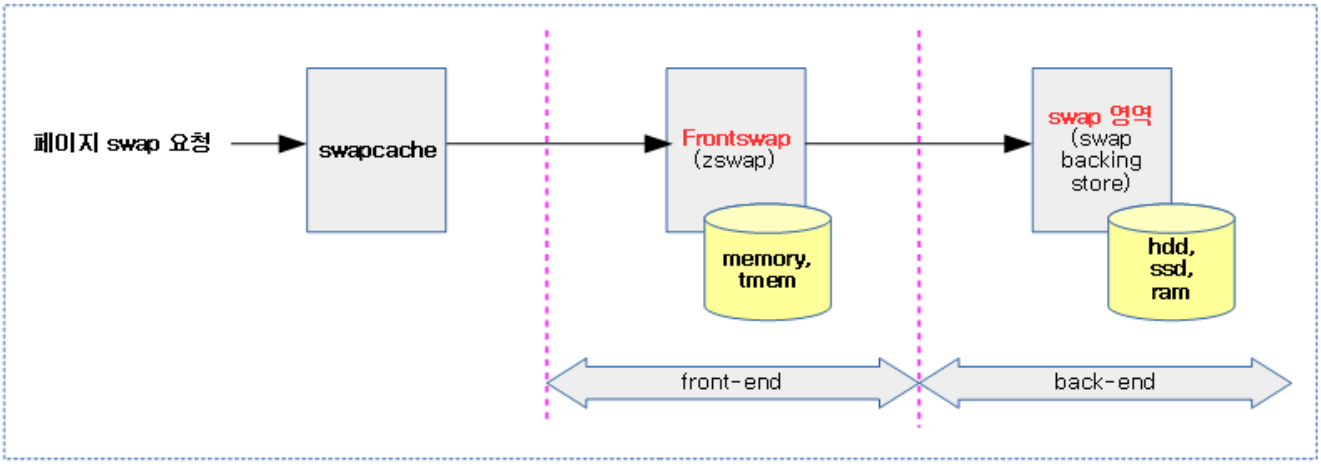
Swap

When requesting an anonymous memory allocation (e.g., malloc) and stack used by the user process, the kernel manages it by allocating an anon page. If you run out of memory, you can save it in the swap area.

The swap mechanism is swapped through a three-step configuration as follows.

- Swap Cache
- Frontswap
- Swap 영역(Swap Backing Store)

The following diagram shows the three-step swap components.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-13.png)

Clean Anon Page

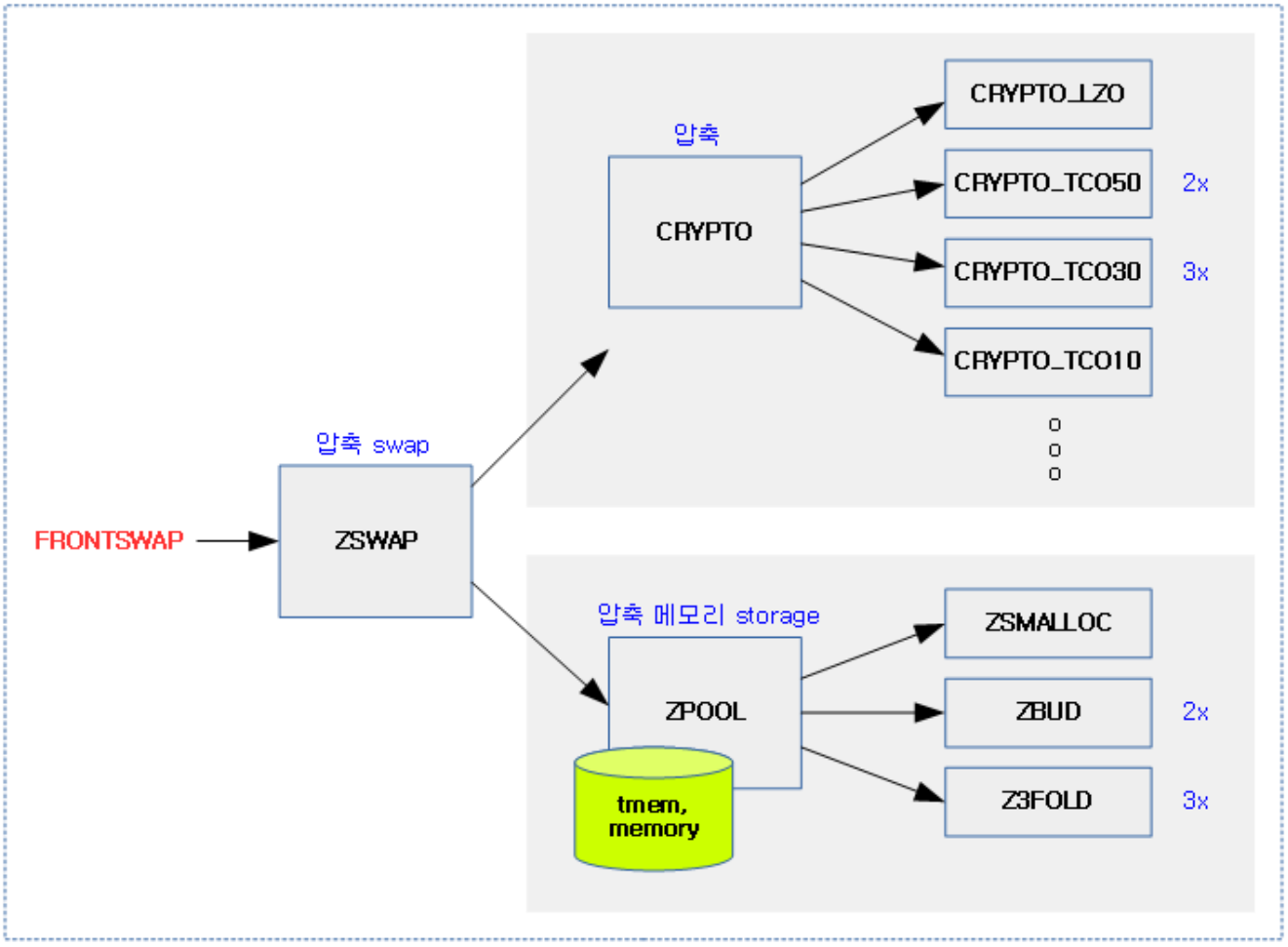
Unlike a swapable anon page, a special anon page that does not have a swap area and cannot be swapped is a clean anon page. These clean anon pages don't need to be swapped.

- PG_swapbacked missing anon page is a clean anon page.
- They are used to avoid directly unmapping to lazy free pages, which are used via the madvise API.
  - Note: MM: Support Madvise (MADV_FREE) (https://lwn.net/Articles/590693/)

FRONTSWAP

It is a swap system created at the Front-End, unlike the swap area that exists at the Back-End.

- The swap device used here must be high-speed to synchronously store the page to be swapped.
  - Back-end swap devices usually store asynchronously.
- It uses transcendent memory (tmem) and is currently available in Xen.
  - DRAM memory or persistent memory
- Reference: Frontswap (https://www.kernel.org/doc/html/latest/vm/frontswap.html) (2012) | Kernel.org



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-14.png)

Transcendent Memory

It is one of the new memory management technologies to improve insufficient RAM. Memory fast enough to be accessed synchronously by the Linux kernel, not directly addressed, but indirectly addressed, and a region is a class of memory that refers to persistent or transient memory that consists of variable sizes. The data stored in this memory can disappear without warning.

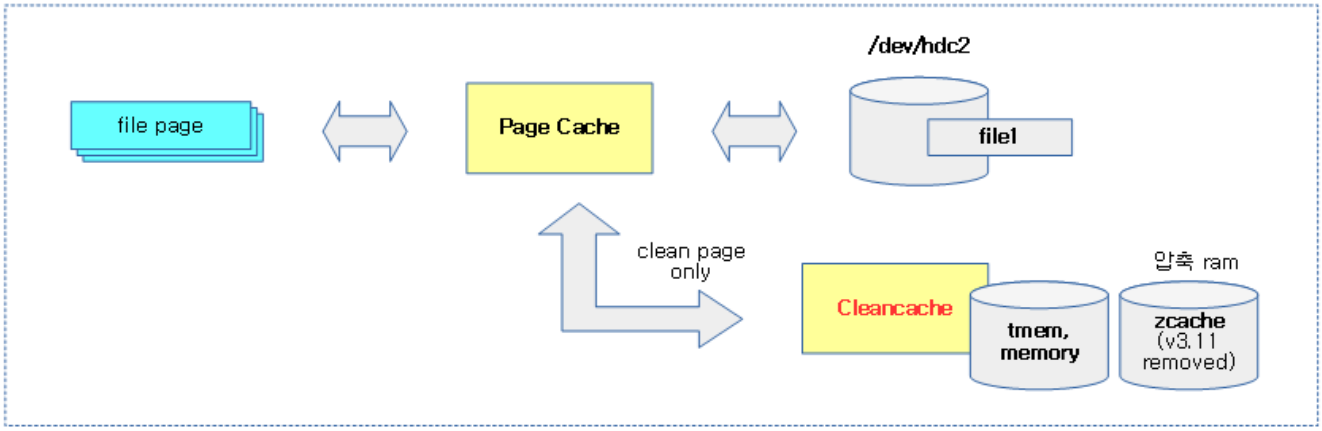
- Consultation:
  - Transcendent memory (2009) (https://lwn.net/Articles/340080/) | LWN.net
  - Transcendent memory in a nutshell (https://lwn.net/Articles/454795/) (2011) | LWN.net

- Transcendent Memory and Friends (2011) | Oracle – Download PDF (https://oss.oracle.com/projects/tmem/dist/documentation/presentations/TmemNotVirt-Linuxcon2011-110729.pdf)

Cleancache

It is a cache that can only store the clean page cache among the page caches. If the swap cache is also stored in the swap area, then this cleancache can be used because it is also a clean page cache.

- Currently, the Linux kernel uses TMEM, which is provided by Xen.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/cleancache-1a.png)

swap 영역(Swap Backing Store)

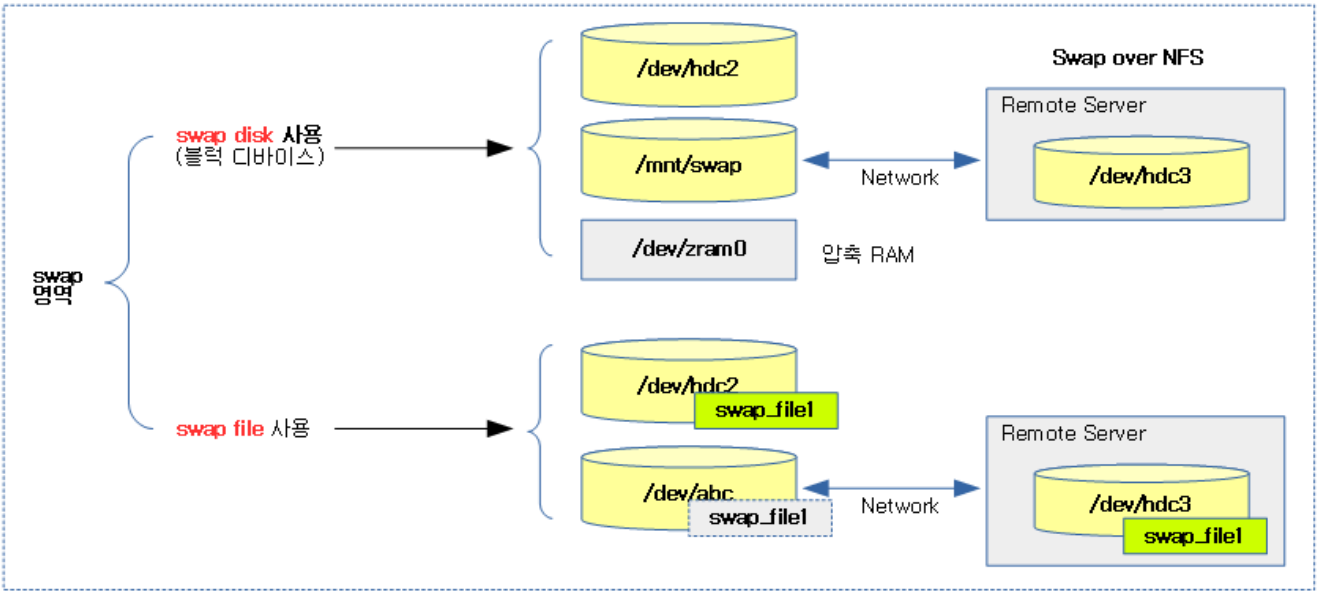
The swap area can be specified and used in the swap file and swap disk (block device) as follows.

- swap files
- swap disk

You can also configure the swap device in the following ways:

- Swap files located on a network-mounted filesystem such as Swap over NFS or sunrpc
- Other uses zram, which uses a portion of local RAM to compress and store it, which reduces IO requests.
  - By default, it is compressed using the CPU, and it is also possible to compress using HW on the server.

There are various ways to specify a swap zone, as shown in the following figure.

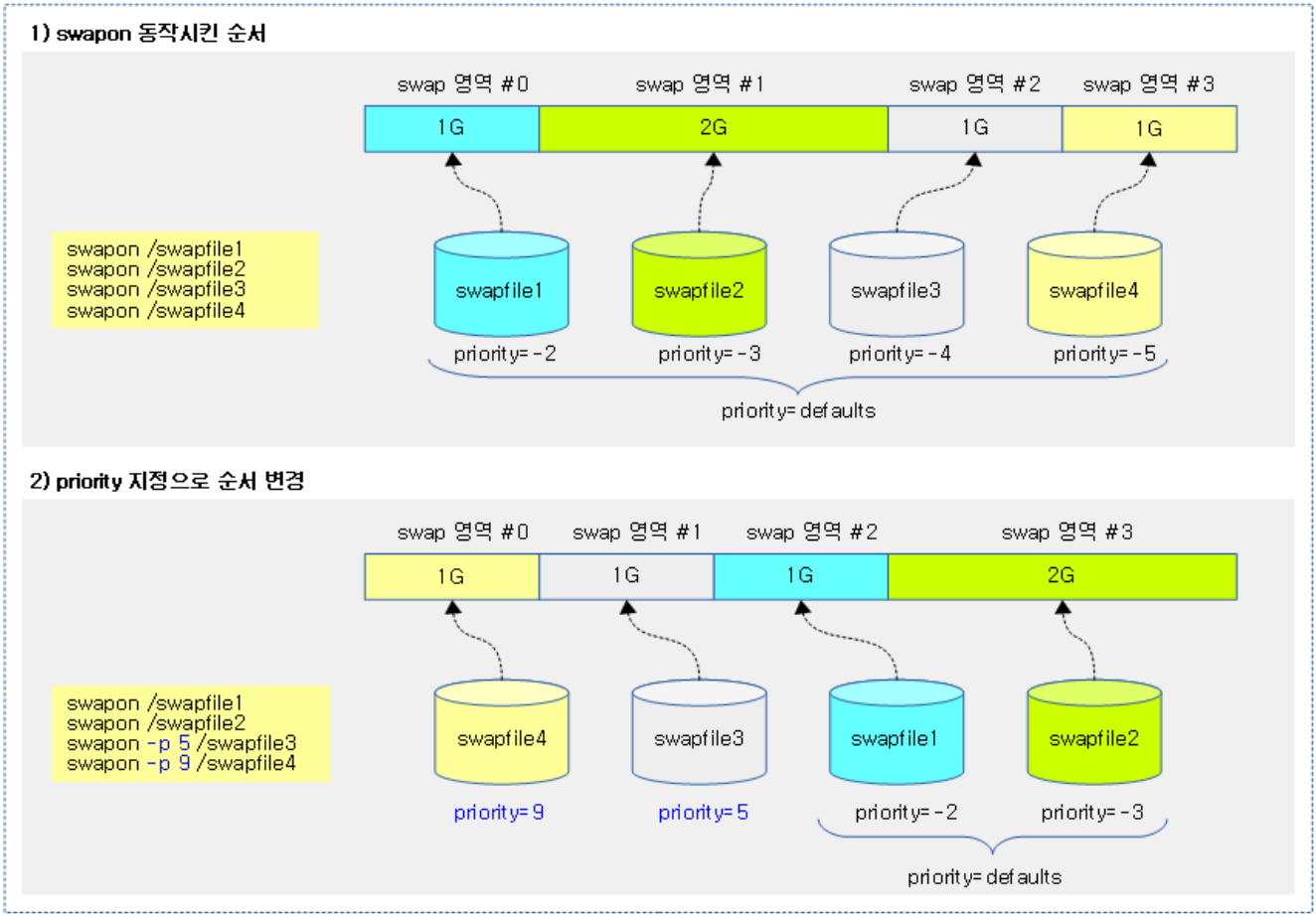


(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-8a.png)

Swap Priority

As shown in the following figure, multiple swap zones can be specified. When assigning a priority, the higher number priority is used first, and if no priority is given, it is used in the order in which it was enabled by swapon by default.

- The default priority value decreases in order from -2 onwards.
- The size of one swap zone varies slightly from architecture to architecture.
  - ARM32
    - It can be used within the physical virtual address space (up to 3G), but is usually used up to 2G.
  - ARM64
    - It can be used within the actual virtual address space (depending on the kernel configuration).



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-9c.png)

swap cache

When an anon page swaps out and swap-in, the contents of the swap zone temporarily exist in RAM. For example, suppose that the anon page used by the user process is stored in the swap area due to insufficient memory. In this case, a fault occurs when the user process tries to access the swapped page, and if the fault handler confirms that the swapped page is the swapped page, it tries to swap-in. Then, when the swap cache is found, it is not loaded in the swap area, but immediately finds the swap cache and changes it to the anon page to use it.

- The swap cache state is expressed by a PG_swapcache flag.

The address_space used by each swap cache is managed by xarray (existing radix tree), and the size of each address_space is limited to 64M to improve the rock contention used to access it.

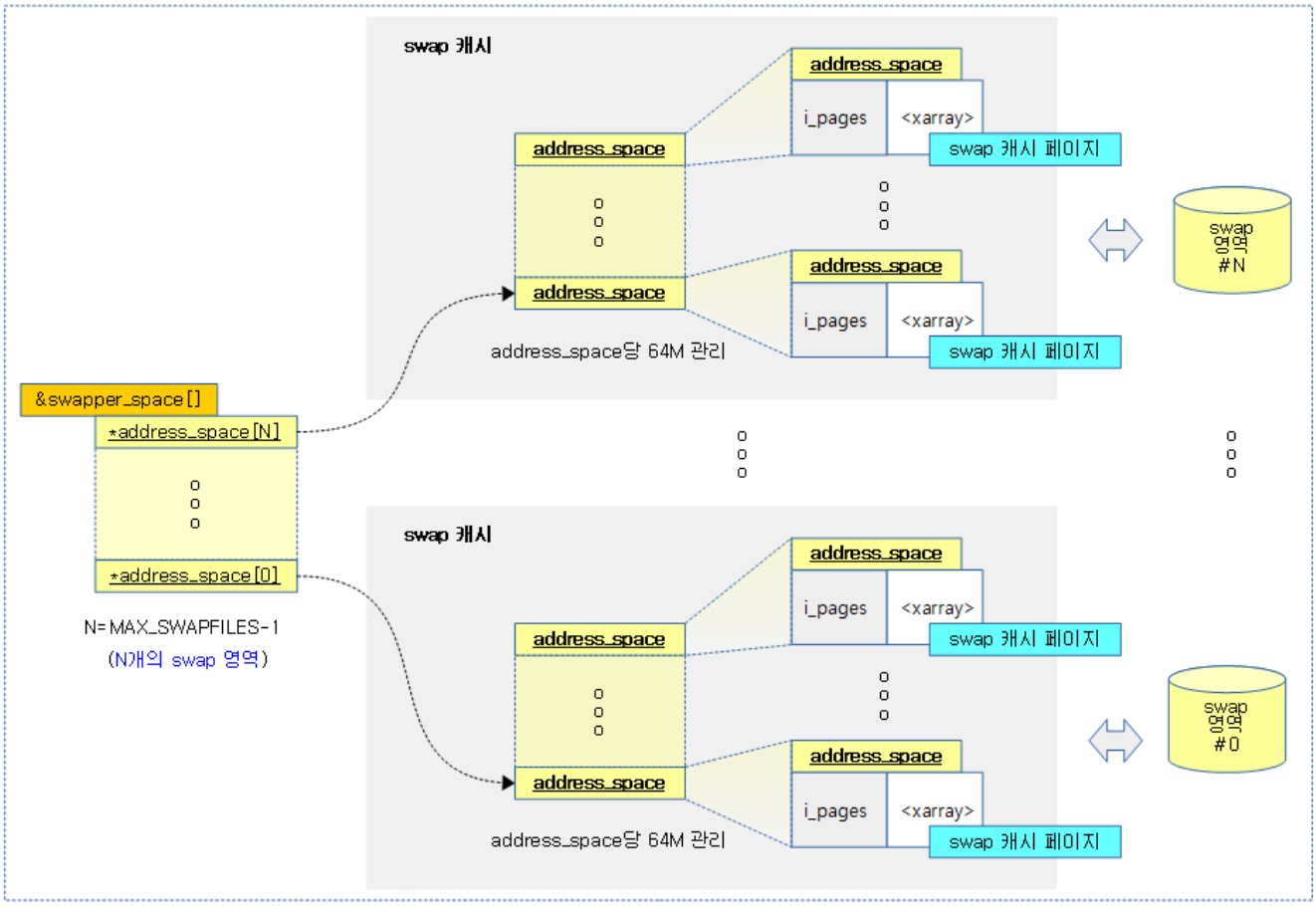
- 참고: mm/swap: split swap cache into 64MB trunks (https://github.com/torvalds/linux/commit/4b3ef9daa4fc0bba742a79faecb17fdaaead083b#diff-58353ba1b37b7ca43f95c0787f5f39bd) (4.11-rc1)

Using XArray Data Structures

The swap cache was previously managed by the kernel using the radix tree, but since kernel v4.20-rc1, it has been managed using the XArray data structure.

- address_space->i_pages used to point to a radix tree, but now it refers to an xarray.

The following figure shows the operation of a swap cache managed in 64M increments when using multiple swap zones.



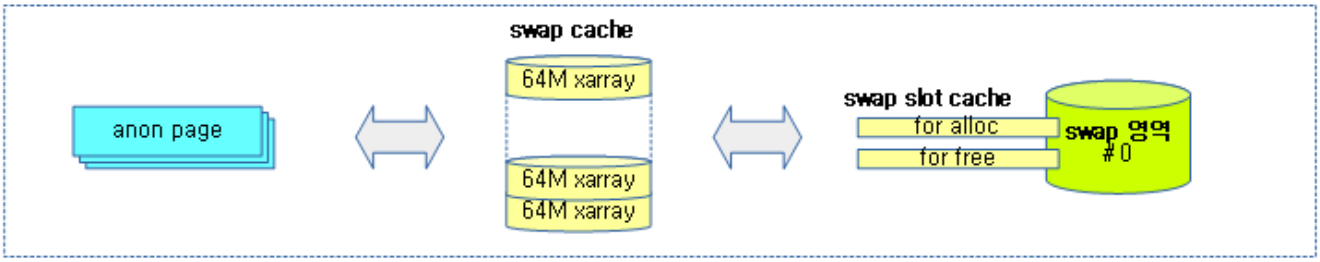
(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-7.png)

Swap Slot Cache

The swap entry is allocated via the swap_map[] of the swap area, which requires a lock in the swap area. In order to improve the performance degradation of the swap area when it is accessed frequently, the per-CPU swap slot cache is used at the front for swap entry allocation so that the swap area can be allocated/released quickly without locking.

- 참고: mm/swap: add cache for swap slots allocation (https://github.com/torvalds/linux/commit/67afa38e012e9581b9b42f2a41dfc56b1280794d#diff-ab76e5bd92ca2482619b9e9b2954f392) (v4.11-rc1)

The following figure shows a swap slot cache consisting of 64 each, which is used for quick allocation and retrieval of swap entries when storing an anon page in the swap zone.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-10a.png)

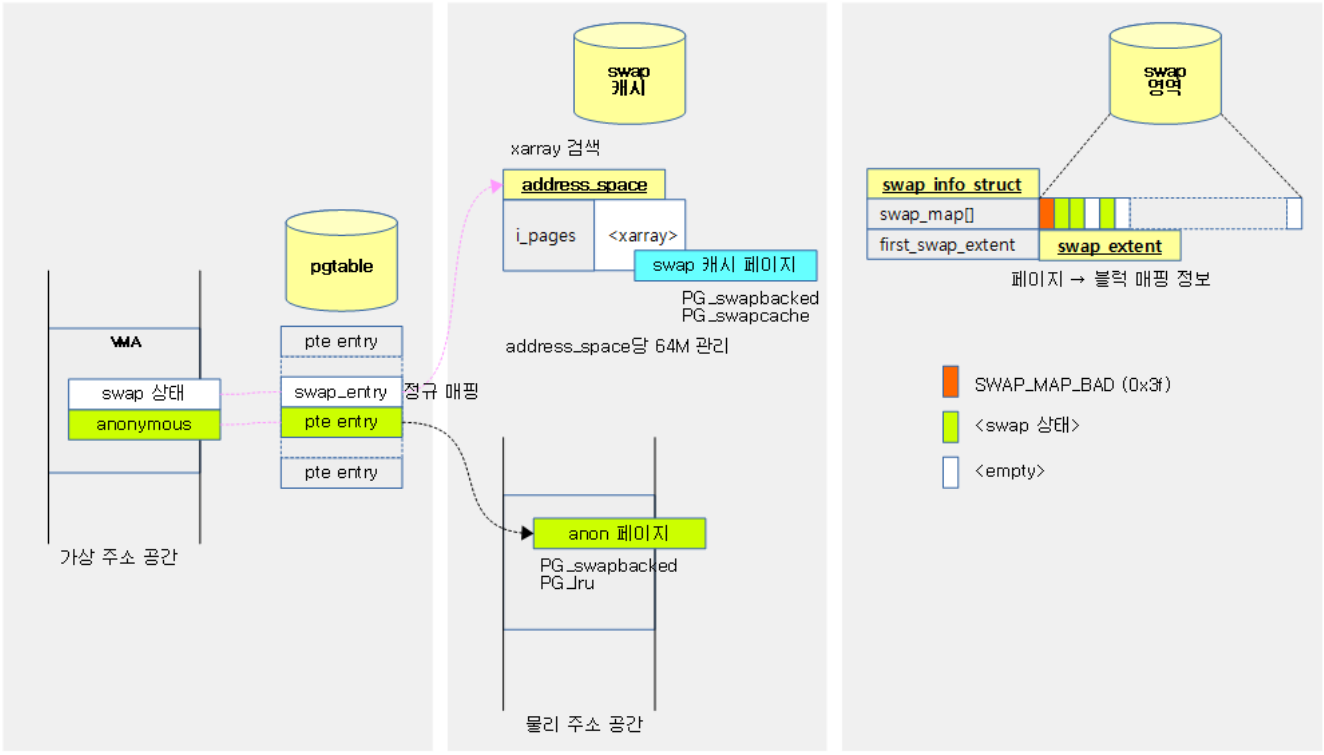
Swap Entries

The swap entry contains the minimum amount of information that can be searched on the swap page.

- When the anonymous page in the VMA area is swapped, the swap entry information is configured and recorded in the corresponding pte entry.
- On ARM, the **composition of the swap entry** is as follows:
  - offset
    - swap page offset
  - type
    - Swap Zone Demarcation
  - Bottom two bits=0b00
    - On ARM and ARM64 architectures, when accessing this page in an unmapped state, a fault error occurs.
- See: Swap Entries (http://jake.dothome.co.kr/swap-entry/) | Qc

The following figure shows the management of swapped and in-use anonymous pages in the virtual address space of the user process from a swap process perspective.

- If you exclude the first header page of one swap zone (type=1) and assign the next page (offset=0) as a swap entry.
  - The swap entry value 64x0 for ARM100 is mapped to the page table, and the xarray of the swap cache stores the page with the entry value as the key.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-6a.png)

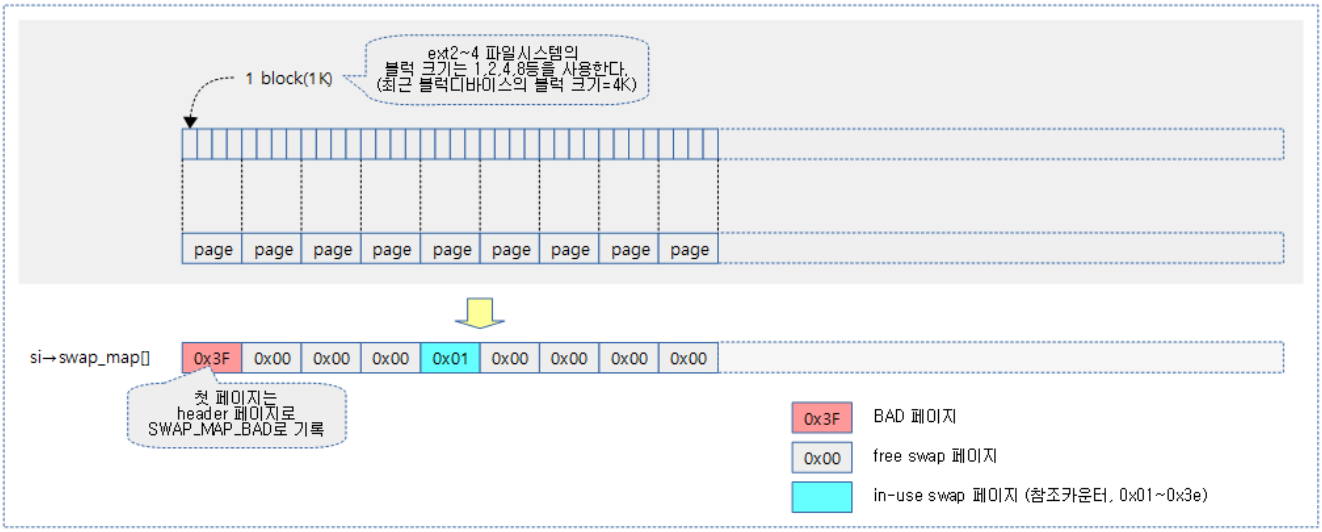
Swap Zone Management

The swap area is used by the swap_map that are members of the swap_info_struct, which are managed on a byte-by-byte basis. In this case, each byte value corresponds to whether or not the page is swapped. This value records the page's reference counter. There are two ways to manage this swap_map:

- Legacy swap map management
  - This is a common method used in legacy HDDs and swap files. In order to use the swap_map used for allocation/release management of the swap page in the SMP system, the cpu that attempts to swap holds a spin-lock on the swap area (swap_info_struct) and searches for a vacant spot from the beginning of the swap_map. This method obtains and uses a spin-lock for each page to be swapped, so lock content has a very big disadvantage.
- Per-CPU Cluster-Level Swap Map Management
  - With the advent of SSD and persistent memory, attempts were made to increase swap performance, and a method of processing on a per-CPU basis was introduced. This method was introduced in kernel v2013.3 in 12 to manage a certain cluster-level page per CPU, reducing the burden of lock contention and increasing performance.
  - consultation
    - Making swapping scalable (https://lwn.net/Articles/704478/) (2016) | LWN.net
    - Reconsidering swapping (https://lwn.net/Articles/690079/) (2016) | LWN.net
    - swap: change block allocation algorithm for SSD (https://github.com/torvalds/linux/commit/2a8f9449343260373398d59228a62a4332ea513a#diff-58353ba1b37b7ca43f95c0787f5f39bd) (v3.12-rc1)
    - swap: make cluster allocation per-cpu (https://github.com/torvalds/linux/commit/ebc2a1a69111eadfeda8487e577f1a5d42

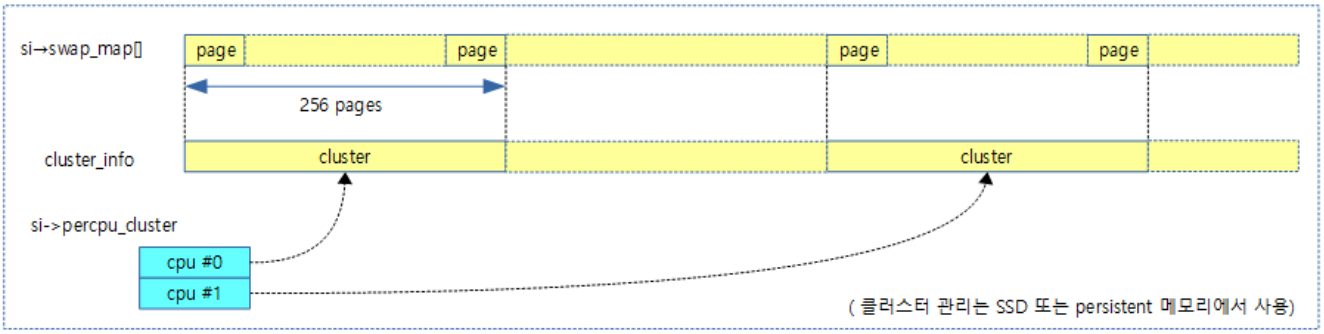
ef0dae#diff-58353ba1b37b7ca43f95c0787f5f39bd) (v3.12-rc1)

As shown in the following figure, the swap status of the swap page is managed by recording each swap page as a one-byte information in the swap_map[] array.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-11.png)

The following figure shows how swap_map is managed on a per-CPU cluster basis.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-12a.png)

Swap Utilities

mkswap

Create a swap area by assigning it to a file or partition.

```
$ mkswap -h

Usage:
 mkswap [options] device [size]

Set up a Linux swap area.

Options:
 -c, --check              check bad blocks before creating the swap area
 -f, --force              allow swap size area be larger than device
 -p, --pagesize SIZE      specify page size in bytes
 -L, --label LABEL        specify label
 -v, --swapversion NUM    specify swap-space version number
 -U, --uuid UUID          specify the uuid to use
 -V, --version            output version information and exit
 -h, --help               display this help and exit
```

e.g. create a 10M(10240K) /abc file name and specify it as a swap file.

- $ fallocate –length 10485760 /abc
  - 또는 dd if=/dev/zero of=/abc bs=1M count=10
- $ mkswap /abc

e.g. create hdc3 partition as swap area.

- $ mkswap /dev/hdc3

swapon

Enable swap by specifying the zone on the file or partition.

```
$ swapon -h

Usage:
 swapon [options] [<spec>]

Enable devices and files for paging and swapping.

Options:
 -a, --all               enable all swaps from /etc/fstab
 -d, --discard[=<policy>] enable swap discards, if supported by device
 -e, --ifexists          silently skip devices that do not exist
 -f, --fixpgsz           reinitialize the swap space if necessary
 -o, --options <list>    comma-separated list of swap options
 -p, --priority <prio>   specify the priority of the swap device
 -s, --summary           display summary about used swap devices (DEPRECATED)
     --show[=<columns>]  display summary in definable table
     --noheadings        don't print table heading (with --show)
     --raw               use the raw output format (with --show)
     --bytes             display swap size in bytes in --show output
 -v, --verbose           verbose mode

 -h, --help     display this help and exit
 -V, --version  output version information and exit

The <spec> parameter:
 -L <label>             synonym for LABEL=<label>
 -U <uuid>              synonym for UUID=<uuid>
 LABEL=<label>          specifies device by swap area label
 UUID=<uuid>            specifies device by swap area UUID
 PARTLABEL=<label>      specifies device by partition label
 PARTUUID=<uuid>        specifies device by partition UUID
 <device>               name of device to be used
 <file>                 name of file to be used

Available discard policy types (for --discard):
 once    : only single-time area discards are issued
 pages   : freed pages are discarded before they are reused
If no policy is selected, both discard types are enabled (default).

Available columns (for --show):
 NAME   device file or partition path
 TYPE   type of the device
 SIZE   size of the swap area
 USED   bytes in use
 PRIO   swap priority
 UUID   swap uuid
 LABEL  swap label

For more details see swapon(8).
```

e.g. /abc in the file name Swap Enable swap.

- $ swapon /abc

e.g. enable swap of hdc3 partition.

- $ swapon /dev/hdc3

swapoff

Disable the swap function of the file or partition specified as the swap area.

```
$ swapoff -h

Usage:
 swapoff [options] [<spec>]

Disable devices and files for paging and swapping.

Options:
 -a, --all              disable all swaps from /proc/swaps
 -v, --verbose          verbose mode

 -h, --help     display this help and exit
 -V, --version  output version information and exit

The <spec> parameter:
 -L <label>             LABEL of device to be used
 -U <uuid>              UUID of device to be used
 LABEL=<label>          LABEL of device to be used
 UUID=<uuid>            UUID of device to be used
 <device>               name of device to be used
 <file>                 name of file to be used

For more details see swapoff(8).
```

e.g. disable swap in the /abc file.

- $ swapoff /abc

e.g. disable swap of hdc3 partition.

- $ swapoff /dev/hdc3

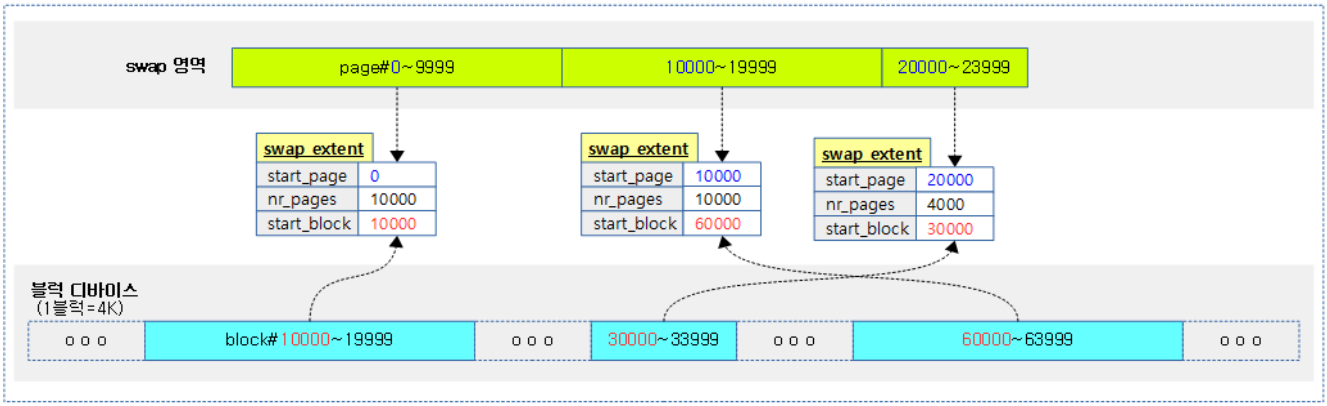
Swap Extent

This is the data structure used to map the page range of the swap area to the block of the swap device. If the swap area is used directly on the block device, only one mapping is required because each page in the swap area is used equally 1:1 consecutively for each block on the block device. However, if you

use a swap file as a swap area, you need to do multiple mappings because each page of the swap file and the actual used block on the block device are not consecutive. However, mapping a single page and a block separately consumes too much of this mapping. Therefore, the mapping is done by grouping them into scopes, and the structure used in this case is the swap_extent.

- One swap_extent includes the start page number, the number of pages, and the starting block number to be mapped.
- Swap extent, which was managed as a list, was changed to rbtree in kernel v5.3-rc1.
  - 참고: mm, swap: use rbtree for swap_extent (https://github.com/torvalds/linux/commit/4efaceb1c5f8136d5fec3f26549d294b8e898bd7#diff-58353ba1b37b7ca43f95c0787f5f39bd) (2019)

The following figure shows the three swap_extent required when the swap file is placed on the block device where it is located, divided into three consecutive blocks.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-extent-1a.png)

Swapon

Activate the swap area by specifying it on a file or block device.

sys_swapon()

mm/swapfile.c -1/3-

```
01 SYSCALL_DEFINE2(swapon, const char __user *, specialfile, int, swap_flag
   s)
02 {
03         struct swap_info_struct *p;
04         struct filename *name;
05         struct file *swap_file = NULL;
06         struct address_space *mapping;
07         int prio;
08         int error;
09         union swap_header *swap_header;
10         int nr_extents;
11         sector_t span;
12         unsigned long maxpages;
13         unsigned char *swap_map = NULL;
14         struct swap_cluster_info *cluster_info = NULL;
```

```
15          unsigned long *frontswap_map = NULL;
16          struct page *page = NULL;
17          struct inode *inode = NULL;
18          bool inced_nr_rotate_swap = false;
19
20          if (swap_flags & ~SWAP_FLAGS_VALID)
21                  return -EINVAL;
22
23          if (!capable(CAP_SYS_ADMIN))
24                  return -EPERM;
25
26          if (!swap_avail_heads)
27                  return -ENOMEM;
28
29          p = alloc_swap_info();
30          if (IS_ERR(p))
31                  return PTR_ERR(p);
32
33          INIT_WORK(&p->discard_work, swap_discard_work);
34
35          name = getname(specialfile);
36          if (IS_ERR(name)) {
37                  error = PTR_ERR(name);
38                  name = NULL;
39                  goto bad_swap;
40          }
41          swap_file = file_open_name(name, O_RDWR|O_LARGEFILE, 0);
42          if (IS_ERR(swap_file)) {
43                  error = PTR_ERR(swap_file);
44                  swap_file = NULL;
45                  goto bad_swap;
46          }
47
48          p->swap_file = swap_file;
49          mapping = swap_file->f_mapping;
50          inode = mapping->host;
51
52          /* If S_ISREG(inode->i_mode) will do inode_lock(inode); */
53          error = claim_swapfile(p, inode);
54          if (unlikely(error))
55                  goto bad_swap;
56
57          /*
58           * Read the swap header.
59           */
60          if (!mapping->a_ops->readpage) {
61                  error = -EINVAL;
62                  goto bad_swap;
63          }
64          page = read_mapping_page(mapping, 0, swap_file);
65          if (IS_ERR(page)) {
66                  error = PTR_ERR(page);
67                  goto bad_swap;
68          }
69          swap_header = kmap(page);
70
71          maxpages = read_swap_header(p, swap_header, inode);
72          if (unlikely(!maxpages)) {
73                  error = -EINVAL;
74                  goto bad_swap;
75          }
76
77          /* OK, set up the swap map and apply the bad block list */
78          swap_map = vzalloc(maxpages);
79          if (!swap_map) {
80                  error = -ENOMEM;
81                  goto bad_swap;
82          }
```

Activate the block device or file for swap in the swap area of the @type number. Returns 0 on success.

- If an invalid swap is flagged in line 20~21 of the code, it returns a -EINVAL error. Acceptable plaques include:
  - SWAP_FLAG_PREFER (0x8000)
  - SWAP_FLAG_PRIO_MASK (0x7fff)
  - SWAP_FLAG_DISCARD (0x10000)
  - SWAP_FLAG_DISCARD_ONCE (0x20000)
  - SWAP_FLAG_DISCARD_PAGES (0x40000
- If you don't have CAP_SYS_ADMIN permissions on lines 23~24 of code, return a -EPERM error.
- If the swap_avail_heads list is not initialized on lines 26~27 of the code, it returns a -ENOMEM error.
- In lines 29~31 of the code, allocate and initialize the swap_info_struct to configure the swap area.
- At line 33 of code, initialize the work so that the swap_discard_work function can be called from the worker thread.
  - You can use the discard function on disks that use SSDs.
- Open the device or file to swap in line 35~48 of the code and specify it in the swap information.
- In lines 49~55 of the code, with the address_space and inode information in the swap file, do the following:
  - In the case of a block device, specify the block device containing the inode in the member bdev of the swap information. Then, add a SWP_BLKDEV flag to the member flag.
  - If it is a file, specify inode->i_sb->s_bdev in the member bdev of the swap information. If this file is already in use as a swapfile, it returns a -EBUSY error.
- In lines 60~63 of the code, if the swap filesystem's (*readpage) hook is not specified, it returns a -INVAL error.
- In lines 64~68 of the code, the swap reads the page cache for index 0 to read the header of the file.
- Let's map the page we read from line 69 for a while using kmap to use as a swap_header.
  - In arm64 it is already mapped because it does not use highmem.
- Parse the swap header in line 71~75 to record the start and end position in the swap information, and find out the number of pages corresponding to the actual swap area.
- In lines 78~82 of code, allocate bytes corresponding to the number of pages corresponding to the actual swap area and assign them to the swap_map.

mm/swapfile.c -2/3-

```
01 .       if (bdi_cap_stable_pages_required(inode_to_bdi(inode)))
02                 p->flags |= SWP_STABLE_WRITES;
03
04         if (bdi_cap_synchronous_io(inode_to_bdi(inode)))
05                 p->flags |= SWP_SYNCHRONOUS_IO;
06
07         if (p->bdev && blk_queue_nonrot(bdev_get_queue(p->bdev))) {
08                 int cpu;
09                 unsigned long ci, nr_cluster;
10
11                 p->flags |= SWP_SOLIDSTATE;
12                 /*
```

```
13                  * select a random position to start with to help wear l
    eveling
14                  * SSD
15                  */
16                 p->cluster_next = 1 + (prandom_u32() % p->highest_bit);
17                 nr_cluster = DIV_ROUND_UP(maxpages, SWAPFILE_CLUSTER);
18
19                 cluster_info = kvcalloc(nr_cluster, sizeof(*cluster_inf
    o),
20                                         GFP_KERNEL);
21                 if (!cluster_info) {
22                         error = -ENOMEM;
23                         goto bad_swap;
24                 }
25
26                 for (ci = 0; ci < nr_cluster; ci++)
27                         spin_lock_init(&((cluster_info + ci)->lock));
28
29                 p->percpu_cluster = alloc_percpu(struct percpu_cluster);
30                 if (!p->percpu_cluster) {
31                         error = -ENOMEM;
32                         goto bad_swap;
33                 }
34                 for_each_possible_cpu(cpu) {
35                         struct percpu_cluster *cluster;
36                         cluster = per_cpu_ptr(p->percpu_cluster, cpu);
37                         cluster_set_null(&cluster->index);
38                 }
39         } else {
40                 atomic_inc(&nr_rotate_swap);
41                 inced_nr_rotate_swap = true;
42         }
43
44         error = swap_cgroup_swapon(p->type, maxpages);
45         if (error)
46                 goto bad_swap;
47
48         nr_extents = setup_swap_map_and_extents(p, swap_header, swap_ma
    p,
49                 cluster_info, maxpages, &span);
50         if (unlikely(nr_extents < 0)) {
51                 error = nr_extents;
52                 goto bad_swap;
53         }
54         /* frontswap enabled? set up bit-per-page map for frontswap */
55         if (IS_ENABLED(CONFIG_FRONTSWAP))
56                 frontswap_map = kvcalloc(BITS_TO_LONGS(maxpages),
57                                          sizeof(long),
58                                          GFP_KERNEL);
59
60         if (p->bdev &&(swap_flags & SWAP_FLAG_DISCARD) && swap_discardab
    le(p)) {
61                 /*
62                  * When discard is enabled for swap with no particular
63                  * policy flagged, we set all swap discard flags here in
64                  * order to sustain backward compatibility with older
65                  * swapon(8) releases.
66                  */
67                 p->flags |= (SWP_DISCARDABLE | SWP_AREA_DISCARD |
68                              SWP_PAGE_DISCARD);
69
70                 /*
71                  * By flagging sys_swapon, a sysadmin can tell us to
72                  * either do single-time area discards only, or to just
73                  * perform discards for released swap page-clusters.
74                  * Now it's time to adjust the p->flags accordingly.
75                  */
76                 if (swap_flags & SWAP_FLAG_DISCARD_ONCE)
```

```
77                      p->flags &= ~SWP_PAGE_DISCARD;
78              else if (swap_flags & SWAP_FLAG_DISCARD_PAGES)
79                      p->flags &= ~SWP_AREA_DISCARD;
80
81              /* issue a swapon-time discard if it's still required */
82              if (p->flags & SWP_AREA_DISCARD) {
83                      int err = discard_swap(p);
84                      if (unlikely(err))
85                              pr_err("swapon: discard_swap(%p): %d\n",
86                                      p, err);
87              }
88      }
```

- In line 1~2 of the code, add the SWP_STABLE_WRITES flag if the device can reliably record swaps.
- In line 4~5 of the code, if the swap recording is a fast device (zram, pmem, etc.), there is no need to process it asynchronously. Add a SWP_SYNCHRONOUS_IO flag at this point.
- In code lines 7~11, add the SWP_SOLIDSTATE flag if it is a non-rotational block device such as an SSD.
- In line 16 of code, we randomly select the next cluster location in the swap area.
- In line 17 of the code, swap uses the number of available pages to determine the number of clusters.
  - One cluster manages as many pages as SWAPFILE_CLUSTER (1).
  - Currently, only x86_64 architectures support THP_SWAP, and instead of 256 pages, HPAGE_PMD_NR pages are managed.
- Allocate and initialize the cluster_info number of clusters determined in line 19~27 of the code.
- In lines 29~38 of code, assign a per-cpu percpu_cluster structure to the member percpu_cluster of the swap information and initialize it.
- In code lines 39~42, increment the nr_rotate_swap and set the inced_nr_rotate_swap to true if the device is not an SSD.
  - If the nr_rotate_swap value is not zero, do not use VMA-based readahead.
- In lines 44~46 of code, allocate and prepare swap_cgroup arrays for swapon for cgroup.
- On lines 48~53 of code, allocate and prepare the swap map and swap_extent.
- In line 55~58 of code, if the kernel supports frontswap, it allocates a map for frontswap.
  - Each bit of the map corresponds to one page.
- Handle SWAP_FLAG_DISCARD request on lines 60~88 of code.

mm/swapfile.c -3/3-

```
01      error = init_swap_address_space(p->type, maxpages);
02      if (error)
03              goto bad_swap;
04
05      mutex_lock(&swapon_mutex);
06      prio = -1;
07      if (swap_flags & SWAP_FLAG_PREFER)
08              prio =
09                (swap_flags & SWAP_FLAG_PRIO_MASK) >> SWAP_FLAG_PRIO_S
    HIFT;
10      enable_swap_info(p, prio, swap_map, cluster_info, frontswap_ma
    p);
11
12      pr_info("Adding %uk swap on %s.  Priority:%d extents:%d across:%
    lluk %s%s%s%s%s\n",
13              p->pages<<(PAGE_SHIFT-10), name->name, p->prio,
```

```
14                nr_extents, (unsigned long long)span<<(PAGE_SHIFT-10),
15                (p->flags & SWP_SOLIDSTATE) ? "SS" : "",
16                (p->flags & SWP_DISCARDABLE) ? "D" : "",
17                (p->flags & SWP_AREA_DISCARD) ? "s" : "",
18                (p->flags & SWP_PAGE_DISCARD) ? "c" : "",
19                (frontswap_map) ? "FS" : "");
20
21        mutex_unlock(&swapon_mutex);
22        atomic_inc(&proc_poll_event);
23        wake_up_interruptible(&proc_poll_wait);
24
25        if (S_ISREG(inode->i_mode))
26                inode->i_flags |= S_SWAPFILE;
27        error = 0;
28        goto out;
29 bad_swap:
30        free_percpu(p->percpu_cluster);
31        p->percpu_cluster = NULL;
32        if (inode && S_ISBLK(inode->i_mode) && p->bdev) {
33                set_blocksize(p->bdev, p->old_block_size);
34                blkdev_put(p->bdev, FMODE_READ | FMODE_WRITE | FMODE_EXC
   L);
35        }
36        destroy_swap_extents(p);
37        swap_cgroup_swapoff(p->type);
38        spin_lock(&swap_lock);
39        p->swap_file = NULL;
40        p->flags = 0;
41        spin_unlock(&swap_lock);
42        vfree(swap_map);
43        kvfree(cluster_info);
44        kvfree(frontswap_map);
45        if (inced_nr_rotate_swap)
46                atomic_dec(&nr_rotate_swap);
47        if (swap_file) {
48                if (inode && S_ISREG(inode->i_mode)) {
49                        inode_unlock(inode);
50                        inode = NULL;
51                }
52                filp_close(swap_file, NULL);
53        }
54 out:
55        if (page && !IS_ERR(page)) {
56                kunmap(page);
57                put_page(page);
58        }
59        if (name)
60                putname(name);
61        if (inode && S_ISREG(inode->i_mode))
62                inode_unlock(inode);
63        if (!error)
64                enable_swap_slots_cache();
65        return error;
66 }
```

- In line 1~3 of the code, initialize the swap area for @type.
  - Prepare by dividing the size of the swap area by the number of units in SWAP_ADDRESS_SPACE_PAGES (2^14=16K pages=64M) for swapper_spaces[type], and assigning address_space array.
- If a SWAP_FLAG_PREFER flag is requested in codelines 6~9, the priority value is added to the flag. In this case, only the priority value is separated and assigned to the prio. In all other cases, it is -1.
- Activate the swap area on lines 10~19 of the code and send the message.

    - "Adding <pages> swap on <file/block device name>. Priority:<prio> extents:<extent mappings> across:<span size> [SS][D][s][c][FS]"
        - SS: SSD
        - D: Discardable
        - s: swap zone discard
        - c: swap page discard
        - FS: FrontSwap Map
- In line 25~26 of the code, if the swap area uses a swap file, add the S_SWAPFILE flag to the inode.
- On lines 27~28 of the code, move to the out label to successfully process without any errors.
- In code lines 29~53, bad_swap: Label. If the swap zone fails to activate, the memory that was allocated will be reclaimed.
- In code lines 54~65, the out: label is. If the initialization of the swap area is successful, the swap slot cache is enabled.

swap_info_struct Initialize after assignment

alloc_swap_info()

mm/swapfile.c

```
01 static struct swap_info_struct *alloc_swap_info(void)
02 {
03         struct swap_info_struct *p;
04         unsigned int type;
05         int i;
06         int size = sizeof(*p) + nr_node_ids * sizeof(struct plist_node);
07
08         p = kvzalloc(size, GFP_KERNEL);
09         if (!p)
10                 return ERR_PTR(-ENOMEM);
11
12         spin_lock(&swap_lock);
13         for (type = 0; type < nr_swapfiles; type++) {
14                 if (!(swap_info[type]->flags & SWP_USED))
15                         break;
16         }
17         if (type >= MAX_SWAPFILES) {
18                 spin_unlock(&swap_lock);
19                 kvfree(p);
20                 return ERR_PTR(-EPERM);
21         }
22         if (type >= nr_swapfiles) {
23                 p->type = type;
24                 swap_info[type] = p;
25                 /*
26                  * Write swap_info[type] before nr_swapfiles, in case a
27                  * racing procfs swap_start() or swap_next() is reading
    them.
28                  * (We never shrink nr_swapfiles, we never free this ent
    ry.)
29                  */
30                 smp_wmb();
31                 nr_swapfiles++;
32         } else {
33                 kvfree(p);
34                 p = swap_info[type];
35                 /*
36                  * Do not memset this entry: a racing procfs swap_next()
```

```
37                  * would be relying on p->type to remain valid.
38                  */
39         }
40         INIT_LIST_HEAD(&p->first_swap_extent.list);
41         plist_node_init(&p->list, 0);
42         for_each_node(i)
43                 plist_node_init(&p->avail_lists[i], 0);
44         p->flags = SWP_USED;
45         spin_unlock(&swap_lock);
46         spin_lock_init(&p->lock);
47         spin_lock_init(&p->cont_lock);
48
49         return p;
50 }
```

Swap area information. (Returns the swap_info_struct pointer you assign.)

- Connect with swap_info_struct struct in code lines 6~10 and allocate an array of plist_node structs equal to the number of nodes.
- In lines 13~16 of the code, look for empty spaces in the swap_info[] array for the number of swap files.
- If the number of swap files created in lines 17~21 is already more than MAX_SWAPFILES, it will unallocate and return a -EPERM error.
- If there are no empty spaces in the swap_info[] array in lines 22~31, specify the memory allocated at the end, and increase the number of swap files.
- In lines 32~39 of code, if there is a vacant space in the swap_info[] array, the memory already allocated will be canceled and the previously allocated memory will be used.
- Initialize the relevant members of the swap_info_struct array assigned in lines 40~47 and set the SWP_USED flag.

Swap Header

swap_header Structs

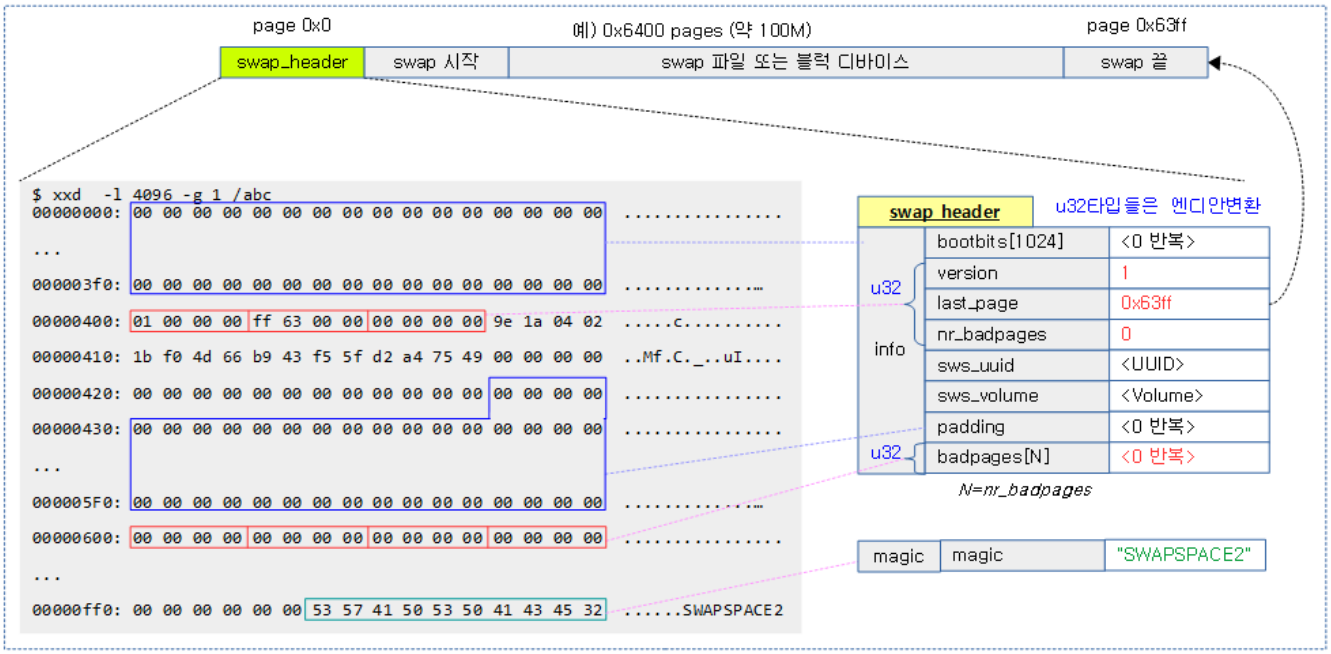
include/linux/swap.h

```
01 /*
02  * Magic header for a swap area. The first part of the union is
03  * what the swap magic looks like for the old (limited to 128MB)
04  * swap area format, the second part of the union adds - in the
05  * old reserved area - some extra information. Note that the first
06  * kilobyte is reserved for boot loader or disk label stuff...
07  *
08  * Having the magic at the end of the PAGE_SIZE makes detecting swap
09  * areas somewhat tricky on machines that support multiple page sizes.
10  * For 2.5 we'll probably want to move the magic to just beyond the
11  * bootbits...
12  */
```

```
01 union swap_header {
02         struct {
03                 char reserved[PAGE_SIZE - 10];
04                 char magic[10];                 /* SWAP-SPACE or SWAPSPA
   CE2 */
05         } magic;
06         struct {
```

```
                char            bootbits[1024]; /* Space for disklabel e
tc. */
                __u32           version;
                __u32           last_page;
                __u32           nr_badpages;
                unsigned char   sws_uuid[16];
                unsigned char   sws_volume[16];
                __u32           padding[117];
                __u32           badpages[1];
        } info;
};
```

The following image shows the header configuration of the swap file.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap_header-1c.png)

read_swap_header()

mm/swapfile.c

```
static unsigned long read_swap_header(struct swap_info_struct *p,
                                      union swap_header *swap_header,
                                      struct inode *inode)
{
        int i;
        unsigned long maxpages;
        unsigned long swapfilepages;
        unsigned long last_page;

        if (memcmp("SWAPSPACE2", swap_header->magic.magic, 10)) {
                pr_err("Unable to find swap-space signature\n");
                return 0;
        }

        /* swap partition endianess hack... */
        if (swab32(swap_header->info.version) == 1) {
                swab32s(&swap_header->info.version);
                swab32s(&swap_header->info.last_page);
                swab32s(&swap_header->info.nr_badpages);
                if (swap_header->info.nr_badpages > MAX_SWAP_BADPAGES)
                        return 0;
                for (i = 0; i < swap_header->info.nr_badpages; i++)
                        swab32s(&swap_header->info.badpages[i]);
```

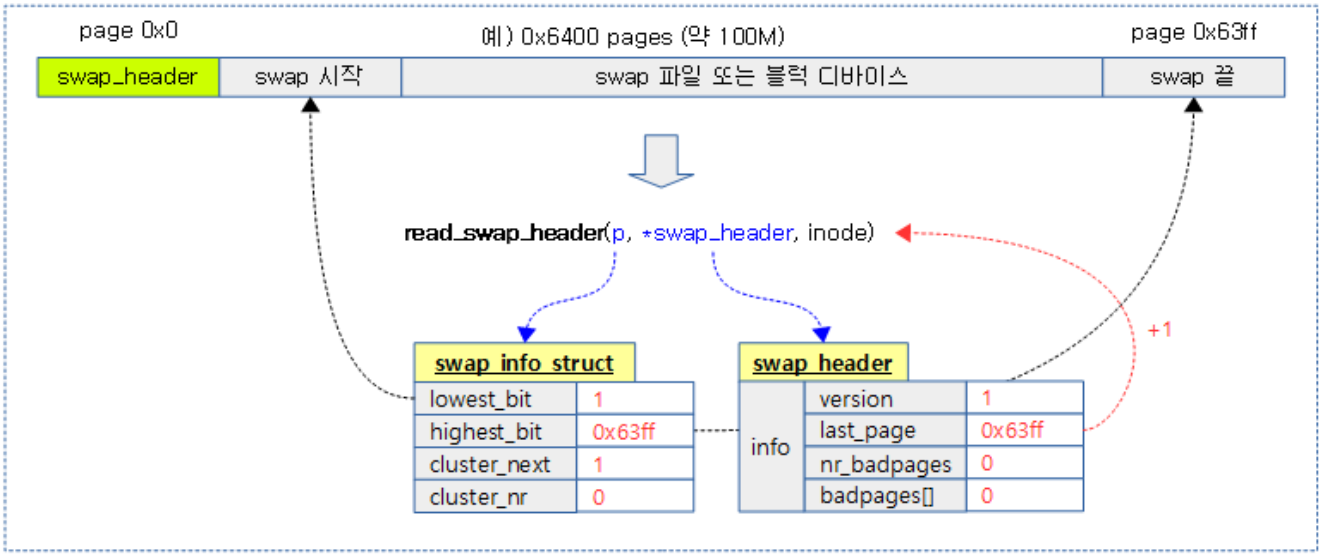
```
24          }
25          /* Check the swap header's sub-version */
26          if (swap_header->info.version != 1) {
27                  pr_warn("Unable to handle swap header version %d\n",
28                          swap_header->info.version);
29                  return 0;
30          }
31
32          p->lowest_bit  = 1;
33          p->cluster_next = 1;
34          p->cluster_nr = 0;
35
36          maxpages = max_swapfile_size();
37          last_page = swap_header->info.last_page;
38          if (!last_page) {
39                  pr_warn("Empty swap-file\n");
40                  return 0;
41          }
42          if (last_page > maxpages) {
43                  pr_warn("Truncating oversized swap area, only using %luk
    out of %luk\n",
44                          maxpages << (PAGE_SHIFT - 10),
45                          last_page << (PAGE_SHIFT - 10));
46          }
47          if (maxpages > last_page) {
48                  maxpages = last_page + 1;
49                  /* p->max is an unsigned int: don't overflow it */
50                  if ((unsigned int)maxpages == 0)
51                          maxpages = UINT_MAX;
52          }
53          p->highest_bit = maxpages - 1;
54
55          if (!maxpages)
56                  return 0;
57          swapfilepages = i_size_read(inode) >> PAGE_SHIFT;
58          if (swapfilepages && maxpages > swapfilepages) {
59                  pr_warn("Swap area shorter than signature indicates\n");
60                  return 0;
61          }
62          if (swap_header->info.nr_badpages && S_ISREG(inode->i_mode))
63                  return 0;
64          if (swap_header->info.nr_badpages > MAX_SWAP_BADPAGES)
65                  return 0;
66
67          return maxpages;
68  }
```

Parse the swap header to get the swap information about its start and end positions.

- In line 10~13 of the code, check the last 10 bytes of the page for a magic string called "SWAPSPACE2", and if it doesn't exist, it prints an error message and returns 0.
  - “Unable to find swap-space signature\n”
- If the version is resolved to 16 in code lines 30~1, all unsigned int values are read by byte swapping. If the version is not 1, it displays the following warning message and returns 0:
  - “Unable to handle swap header version %d\n”
- In line 32~34 of the code, initialize the cluster number by specifying 0 and the next cluster number as 1. Then, specify 1 as the first page for the swap start (lowestbit) page.
- In line 36 of code, we find out the limit of swap offset pages that the architecture supports, and assign them to maxpages.
- In line 37~41 of the code, if the number of last_page written in the swap header is zero, it prints an "Empty swap-file" message and returns 0.

- In line 42~46 of the code, it prints a warning message if the last_page exceeds maxpages.
- In lines 47~52 of code, if maxpages is greater than last_page, maxpages substitutes last_page+1.
- In line 53 of code, specify maxpages-1 as the swap highestbit page.
- In line 57~61 of the code, the number of pages in the swap file or block device is determined and if it is less than maxpages, it prints the following warning message and returns 0:
  - “Swap area shorter than signature indicates\n”
- In line 62~63 of code, the swap file returns 0 if a bad page exists.
- In lines 64~65 of code, if the number of bad pages exceeds MAX_SWAP_BADPAGES, it returns 0.
- Returns maxpages at line 67 of the code.

The following figure shows the swap header being read by a swap file or block device and specifying the start (lowest_bit) and end (highest_bit) positions for the first time in the swap information.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/read_swap_header-1.png)

swap for Cgroup

swap_cgroup_swapon()

mm/swap_cgroup.c

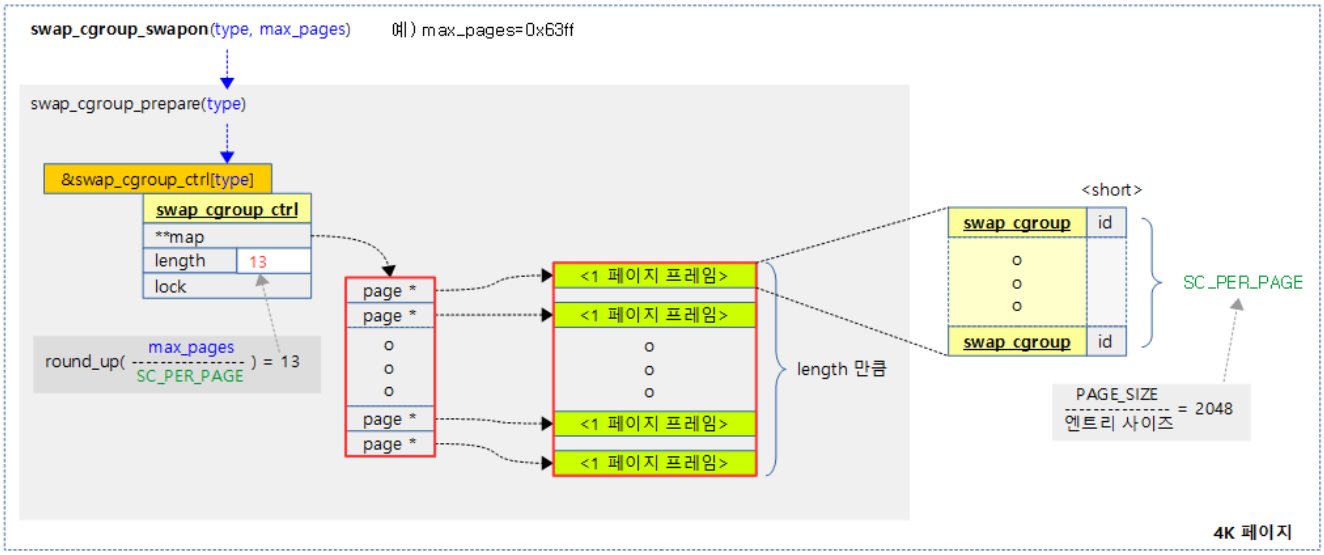
```
01 int swap_cgroup_swapon(int type, unsigned long max_pages)
02 {
03         void *array;
04         unsigned long array_size;
05         unsigned long length;
06         struct swap_cgroup_ctrl *ctrl;
07
08         if (!do_swap_account)
09                 return 0;
10
11         length = DIV_ROUND_UP(max_pages, SC_PER_PAGE);
12         array_size = length * sizeof(void *);
13
14         array = vzalloc(array_size);
15         if (!array)
16                 goto nomem;
```

```
17
18          ctrl = &swap_cgroup_ctrl[type];
19          mutex_lock(&swap_cgroup_mutex);
20          ctrl->length = length;
21          ctrl->map = array;
22          spin_lock_init(&ctrl->lock);
23          if (swap_cgroup_prepare(type)) {
24                  /* memory shortage */
25                  ctrl->map = NULL;
26                  ctrl->length = 0;
27                  mutex_unlock(&swap_cgroup_mutex);
28                  vfree(array);
29                  goto nomem;
30          }
31          mutex_unlock(&swap_cgroup_mutex);
32
33          return 0;
34  nomem:
35          pr_info("couldn't allocate enough memory for swap_cgroup\n");
36          pr_info("swap_cgroup can be disabled by swapaccount=0 boot optio
    n\n");
37          return -ENOMEM;
38  }
```

Allocate and prepare swap cgroups. Returns 0 on success.

- If the kernel doesn't support memcg swap in line 8~9 of code, the function will be left out.
  - CONFIG_MEMCG_SWAP & CONFIG_MEMCG_SWAP_ENABLED kernel options.
- In lines 11~16 of the code, calculate the number of swap_cgroup structs required for the number of pages required in the swap area, and allocate an array of page pointers for that number.
- In lines 18~22 of code, concatenate the array of page pointers assigned to the swap_cgroup_ctrl corresponding to @type in the global swap_cgroup_ctrl[] array, and contain the number of swap_cgroup structs calculated by length.
- In lines 23~30 of the code, assign and link the pages to be used for the struct array to swap_cgroup the page pointer array prepared by assigning them to the swap_cgroup_ctrl.
- On line 33 of the code, when the normal allocation completes, it returns 0.
- In code lines 34~37, nomem: is the label. Outputs a message saying that there is not enough memory to allocate the swap cgroup. And if you run out of memory, you can use "swapaccount=0" as a kernel option to disable the swap cgroup.

The following diagram shows the process of allocating and preparing a swap cgroup.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap_cgroup_swapon-1.png)

swap_cgroup_prepare()

mm/swap_cgroup.c

```
1  /*
2   * allocate buffer for swap_cgroup.
3   */
```

```
01 static int swap_cgroup_prepare(int type)
02 {
03         struct page *page;
04         struct swap_cgroup_ctrl *ctrl;
05         unsigned long idx, max;
06
07         ctrl = &swap_cgroup_ctrl[type];
08
09         for (idx = 0; idx < ctrl->length; idx++) {
10                 page = alloc_page(GFP_KERNEL | __GFP_ZERO);
11                 if (!page)
12                         goto not_enough_page;
13                 ctrl->map[idx] = page;
14
15                 if (!(idx % SWAP_CLUSTER_MAX))
16                         cond_resched();
17         }
18         return 0;
19 not_enough_page:
20         max = idx;
21         for (idx = 0; idx < max; idx++)
22                 __free_page(ctrl->map[idx]);
23
24         return -ENOMEM;
25 }
```

For each page pointer array that you have prepared by assigning it to the swap_cgroup_ctrl, it assigns and links the pages that you want to use for the swap_cgroup struct array.

- Specify the swap_cgroup_ctrl corresponding to @type on line 7 of the code.
- Travers by ctrl->length on lines 9~17 of code and link to ctrl->map[idx] by allocating pages for an array of swap_cgroup structs.
- On line 18 of the code, it returns 0 on success.

- In code lines 19~24, not_enough_page: Labels. If you run out of memory, it deallocates the allocated pages and returns a -ENOMEM error.

Initialize the swap map

setup_swap_map_and_extents()

mm/swapfile.c

```
01 static int setup_swap_map_and_extents(struct swap_info_struct *p,
02                                         union swap_header *swap_header,
03                                         unsigned char *swap_map,
04                                         struct swap_cluster_info *cluste
   r_info,
05                                         unsigned long maxpages,
06                                         sector_t *span)
07 {
08         unsigned int j, k;
09         unsigned int nr_good_pages;
10         int nr_extents;
11         unsigned long nr_clusters = DIV_ROUND_UP(maxpages, SWAPFILE_CLUS
   TER);
12         unsigned long col = p->cluster_next / SWAPFILE_CLUSTER % SWAP_CL
   USTER_COLS;
13         unsigned long i, idx;
14
15         nr_good_pages = maxpages - 1;   /* omit header page */
16
17         cluster_list_init(&p->free_clusters);
18         cluster_list_init(&p->discard_clusters);
19
20         for (i = 0; i < swap_header->info.nr_badpages; i++) {
21                 unsigned int page_nr = swap_header->info.badpages[i];
22                 if (page_nr == 0 || page_nr > swap_header->info.last_pag
   e)
23                         return -EINVAL;
24                 if (page_nr < maxpages) {
25                         swap_map[page_nr] = SWAP_MAP_BAD;
26                         nr_good_pages--;
27                         /*
28                          * Haven't marked the cluster free yet, no list
29                          * operation involved
30                          */
31                         inc_cluster_info_page(p, cluster_info, page_nr);
32                 }
33         }
34
35         /* Haven't marked the cluster free yet, no list operation involv
   ed */
36         for (i = maxpages; i < round_up(maxpages, SWAPFILE_CLUSTER); i+
   +)
37                 inc_cluster_info_page(p, cluster_info, i);
38
39         if (nr_good_pages) {
40                 swap_map[0] = SWAP_MAP_BAD;
41                 /*
42                  * Not mark the cluster free yet, no list
43                  * operation involved
44                  */
45                 inc_cluster_info_page(p, cluster_info, 0);
46                 p->max = maxpages;
47                 p->pages = nr_good_pages;
48                 nr_extents = setup_swap_extents(p, span);
```

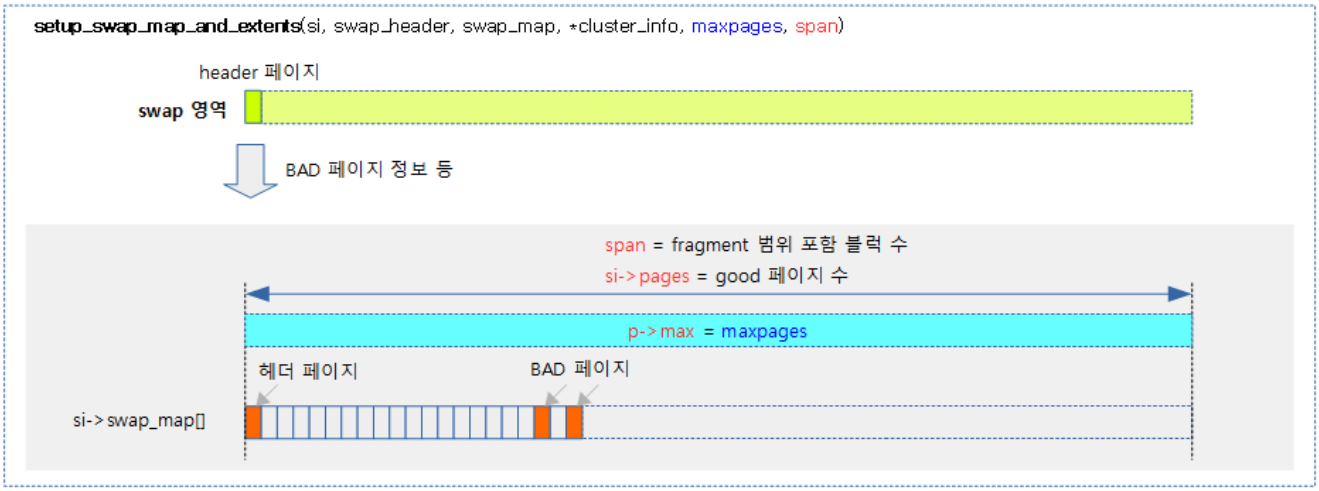
```
49                  if (nr_extents < 0)
50                          return nr_extents;
51                  nr_good_pages = p->pages;
52          }
53          if (!nr_good_pages) {
54                  pr_warn("Empty swap-file\n");
55                  return -EINVAL;
56          }
57
58          if (!cluster_info)
59                  return nr_extents;
60
61          /*
62           * Reduce false cache line sharing between cluster_info and
63           * sharing same address space.
64           */
65          for (k = 0; k < SWAP_CLUSTER_COLS; k++) {
66                  j = (k + col) % SWAP_CLUSTER_COLS;
67                  for (i = 0; i < DIV_ROUND_UP(nr_clusters, SWAP_CLUSTER_C
   OLS); i++) {
68                          idx = i * SWAP_CLUSTER_COLS + j;
69                          if (idx >= nr_clusters)
70                                  continue;
71                          if (cluster_count(&cluster_info[idx]))
72                                  continue;
73                          cluster_set_flag(&cluster_info[idx], CLUSTER_FLA
   G_FREE);
74                          cluster_list_add_tail(&p->free_clusters, cluster
   _info,
75                                                idx);
76                  }
77          }
78          return nr_extents;
79  }
```

It allocates and prepares swap maps and swap_extent, and returns the number of swap_extent allocated.

- In line 11 of the code, swap to calculate the required number of clusters with the maximum number of pages.
- On line 12 of code, we find the cluster number for the next step.
  - swap files can be up to 64M increments, and the maximum cluster number is
- When calculating the number of good pages in line 15 of the code, we first use 1 page as the header page before subtracting the bad pages, so subtract 1 from the swap maximum number of pages.
- Clear the free_clusters and discard_clusters lists on lines 17~18 of the code.
- In code lines 20~33, traverse the number of bad pages recorded on the header page, get the bad page number, mark the corresponding swap_map[] SWAP_MAP_BAD, and decrease the number of good pages. The cluster with the marked bad page is also set to in use.
- In line 36~37 of the code, the swap area is managed on a cluster-by-cluster basis, and all the pages in the remaining area are set to the cluster in use.
- If the good page exists in line 39~52 of the code, the first page of the swap area should be set to SWAP_MAP_BAD, and the first cluster should be set to In Use. and configure the swap extent.
- If there are no good pages in lines 53~56, it will warn that the swap file is empty and return a -EINVAL error.
- If no @cluster_info is specified in code lines 58~59, returns the number of swap extents.

- In line 65~77 of the code, in order to reduce false cache line sharing, the cluster_info and address_space were separated by each CPU. Add the available free clusters to the fre_clusters list, and add a CLUSTER_FLAG_FREE to the flag of the cluster_info information.
- Returns the number of swap extent at line 78 of code.

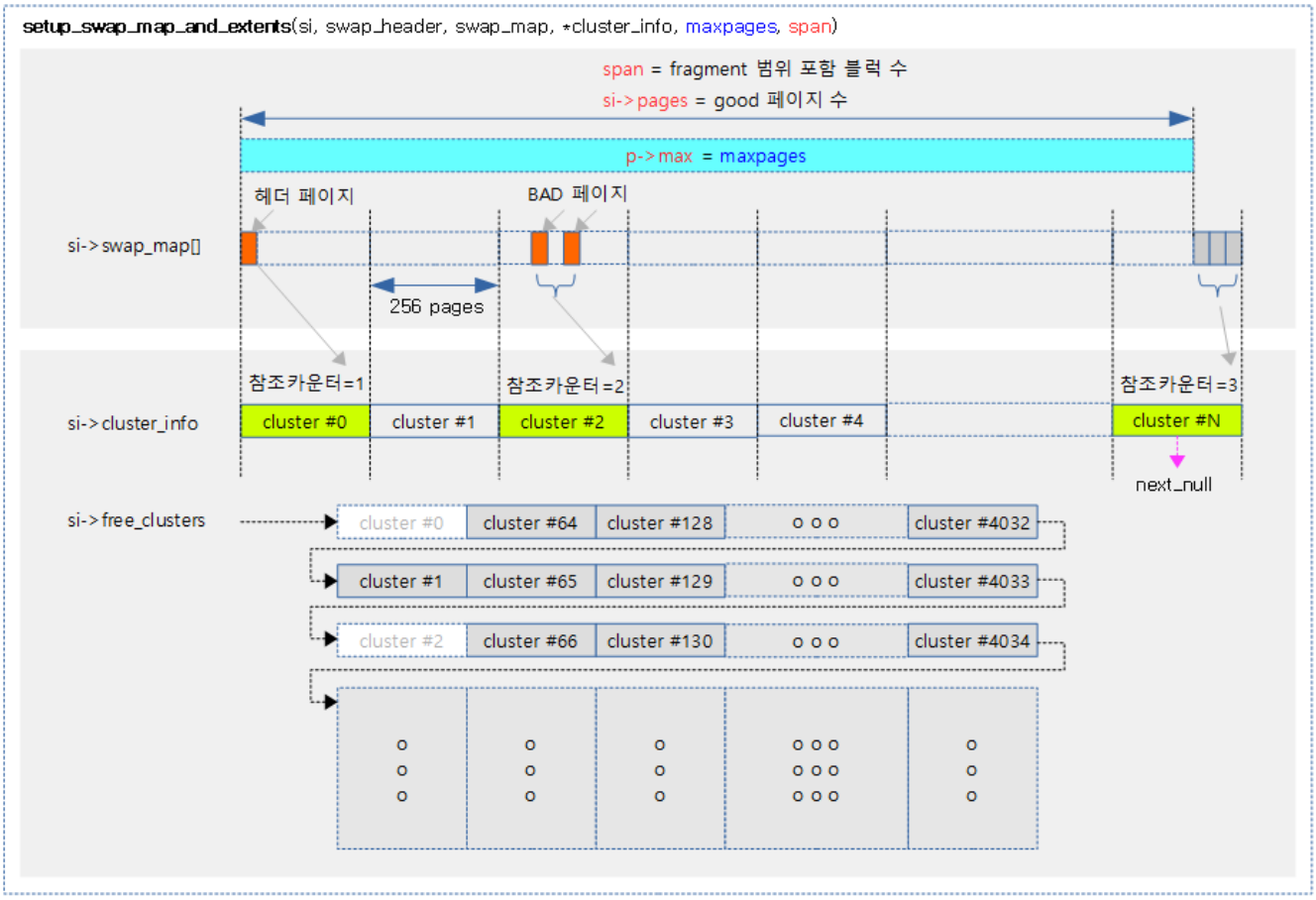
The following figure shows how the swap_map is constructed with the information analyzed from the header page of the swap area.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/setup_swap_map_and_extents-1a.png)

The following figure shows the process of preparing a list of free clusters for cluster configuration when using SSDs.

- Clusters to which header pages and BAD pages belong are to be made in use and removed from the list of free clusters.
- When adding a free cluster, it shows that the clusters are not put in order, but are added separately in groups of 64.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/setup_swap_map_and_extents-2.png)

Swap Extents

The swap area is used to map the range to the block device. Depending on the type of swap area, there are three ways to prepare the swap extent.

- When using a block device, the swap area and the block device are all mapped using one swap extent at a time.
- In the mounted filesystem, the swap area where the swap file with (*swap_activate) support is located is also mapped all at once, so only one swap extent is required.
  - nfs, xfs, btrfs, sunrpc, ...
- Generic Swap Files
  - This is the case when you use a swap file as a swap area. In this case, the empty space of the block device is fragmented in several places, so multiple swap_extent are required.

setup_swap_extents()

mm/swapfile.c

```
01 /*
02  * A `swap extent' is a simple thing which maps a contiguous range of pa
   ges
03  * onto a contiguous range of disk blocks.  An ordered list of swap exte
   nts
```

```
 * is built at swapon time and is then used at swap_writepage/swap_readpage
 * time for locating where on disk a page belongs.
 *
 * If the swapfile is an S_ISBLK block device, a single extent is installed.
 * This is done so that the main operating code can treat S_ISBLK and S_ISREG
 * swap files identically.
 *
 * Whether the swapdev is an S_ISREG file or an S_ISBLK blockdev, the swap
 * extent list operates in PAGE_SIZE disk blocks.  Both S_ISREG and S_ISBLK
 * swapfiles are handled *identically* after swapon time.
 *
 * For S_ISREG swapfiles, setup_swap_extents() will walk all the file's blocks
 * and will parse them into an ordered extent list, in PAGE_SIZE chunks.  If
 * some stray blocks are found which do not fall within the PAGE_SIZE alignment
 * requirements, they are simply tossed out - we will never use those blocks
 * for swapping.
 *
 * For S_ISREG swapfiles we set S_SWAPFILE across the life of the swapon.  This
 * prevents root from shooting her foot off by ftruncating an in-use swapfile,
 * which will scribble on the fs.
 *
 * The amount of disk space which a single swap extent represents varies.
 * Typically it is in the 1-4 megabyte range.  So we can have hundreds of
 * extents in the list.  To avoid much list walking, we cache the previous
 * search location in `curr_swap_extent', and start new searches from there.
 * This is extremely effective.  The average number of iterations in
 * map_swap_page() has been measured at about 0.3 per page.  - akpm.
 */
```

```
static int setup_swap_extents(struct swap_info_struct *sis, sector_t *span)
{
        struct file *swap_file = sis->swap_file;
        struct address_space *mapping = swap_file->f_mapping;
        struct inode *inode = mapping->host;
        int ret;

        if (S_ISBLK(inode->i_mode)) {
                ret = add_swap_extent(sis, 0, sis->max, 0);
                *span = sis->pages;
                return ret;
        }

        if (mapping->a_ops->swap_activate) {
                ret = mapping->a_ops->swap_activate(sis, swap_file, span);
                if (ret >= 0)
                        sis->flags |= SWP_ACTIVATED;
                if (!ret) {
                        sis->flags |= SWP_FS;
                        ret = add_swap_extent(sis, 0, sis->max, 0);
                        *span = sis->pages;
                }
```

```
23                  return ret;
24          }
25
26          return generic_swapfile_activate(sis, swap_file, span);
27  }
```

Prepare swap extents. Specify the number of pages in the output factor @span. If the result is 0, it is newly activated.

- In line 8~12 of the code, if the swap area is a block device, the entire swap area (0 ~ sis->max page) can be mapped to the specified block device at once. To do this, add 0 swap_extent to map the entire starting block 1.
- If the (*swap_activate) hook of the mapped operation is supported on lines 14~24 of code, add the SWP_ACTIVATED flag if it is already enabled after calling it. Also, if it is newly enabled, set the SWP_FS flag and configure and add 0 swap_extent to map the entire swap area (1 ~ sis->max page) at once.
- In line 26 of code, if the swap area is a generic swap file, add and enable swap_extent for one or more mappings.

generic_swapfile_activate()

mm/page_io.c

```
01  int generic_swapfile_activate(struct swap_info_struct *sis,
02                                struct file *swap_file,
03                                sector_t *span)
04  {
05          struct address_space *mapping = swap_file->f_mapping;
06          struct inode *inode = mapping->host;
07          unsigned blocks_per_page;
08          unsigned long page_no;
09          unsigned blkbits;
10          sector_t probe_block;
11          sector_t last_block;
12          sector_t lowest_block = -1;
13          sector_t highest_block = 0;
14          int nr_extents = 0;
15          int ret;
16
17          blkbits = inode->i_blkbits;
18          blocks_per_page = PAGE_SIZE >> blkbits;
19
20          /*
21           * Map all the blocks into the extent list.  This code doesn't t
    ry
22           * to be very smart.
23           */
24          probe_block = 0;
25          page_no = 0;
26          last_block = i_size_read(inode) >> blkbits;
27          while ((probe_block + blocks_per_page) <= last_block &&
28                          page_no < sis->max) {
29                  unsigned block_in_page;
30                  sector_t first_block;
31
32                  cond_resched();
33
34                  first_block = bmap(inode, probe_block);
35                  if (first_block == 0)
36                          goto bad_bmap;
```

```
37
38                  /*
39                   * It must be PAGE_SIZE aligned on-disk
40                   */
41                  if (first_block & (blocks_per_page - 1)) {
42                          probe_block++;
43                          goto reprobe;
44                  }
45
46                  for (block_in_page = 1; block_in_page < blocks_per_page;
47                                          block_in_page++) {
48                          sector_t block;
49
50                          block = bmap(inode, probe_block + block_in_pag
    e);
51                          if (block == 0)
52                                  goto bad_bmap;
53                          if (block != first_block + block_in_page) {
54                                  /* Discontiguity */
55                                  probe_block++;
56                                  goto reprobe;
57                          }
58                  }
59
60                  first_block >>= (PAGE_SHIFT - blkbits);
61                  if (page_no) {  /* exclude the header page */
62                          if (first_block < lowest_block)
63                                  lowest_block = first_block;
64                          if (first_block > highest_block)
65                                  highest_block = first_block;
66                  }
67
68                  /*
69                   * We found a PAGE_SIZE-length, PAGE_SIZE-aligned run of
    blocks
70                   */
71                  ret = add_swap_extent(sis, page_no, 1, first_block);
72                  if (ret < 0)
73                          goto out;
74                  nr_extents += ret;
75                  page_no++;
76                  probe_block += blocks_per_page;
77  reprobe:
78                  continue;
79          }
80          ret = nr_extents;
81          *span = 1 + highest_block - lowest_block;
82          if (page_no == 0)
83                  page_no = 1;    /* force Empty message */
84          sis->max = page_no;
85          sis->pages = page_no - 1;
86          sis->highest_bit = page_no - 1;
87  out:
88          return ret;
89  bad_bmap:
90          pr_err("swapon: swapfile has holes\n");
91          ret = -EINVAL;
92          goto out;
93  }
```

The swap area enables this in the generic swap file. Returns the number of extents added on success.

- From lines 17~18 of the code, find the number of block bits that can fit in a page and assign it to the blocks_per_page.
  - The block size is 512 bytes. However, the block referred to here refers to a virtual block that can be processed on a per-IO basis. If the swap file runs on an ext2, ext3, or ext4

filesystem, the virtual block size supports 1K, 2K, 4K, and 8K, and uses 4K by default.
    - e.g. PAGE_SIZE(4096) >> inodes->iblkbits(12) = 1 block
- In lines 24~26 of the code, prepare the start page (page_no) of the file for traversal from 0 to the end page (sis->max). In this case, the probe_block also starts with 0, and the last_block is assigned the end block number of the file.
- From code lines 27~28, it increases from probe_block to blocks_per_page and traverses to last_block.
    - If the swap file runs on an ext2, ext3, or ext4 filesystem, using the default block size setting will result in one page being the same as one block. Therefore, in the case of blocks_per_page, it is 1.
- In lines 34~36 of the code, get the disk block number for page probe_block of the swap file and assign it to the first_block.
- If the disk block number (first_block) you learned from lines 41~44 of code is not sorted by blocks_per_page, increment the block number (probe_block) for the swap file until it is sorted and move to the reprobe label.
- In code lines 46~58, if there are more than 2 blocks in the page, it will traverse from the second page in the block to the end page in the block. probe_block + the number of pages you are traversing plus the number to make sure that it is linked to the block number of the corresponding block device. If they don't match, move them to the reprobe label.
- In line 60~66 of the code, the smallest lowest_block and the largest highest_block are updated with the number (first_block) of the block device that you have learned, excluding the header page.
- In lines 71~74 of code, map 1 page corresponding to the page_no of the swap file to the known block device number (first_block). If the actual swap extent is added at the time of mapping, 1 is returned. If it is not added and merges into an existing swap extent, 0 is returned. The number returned is added to the nr_extents.
- Continue to process the next page at lines 75~76 of code. The probe_block should be sorted by page, so increase it by blocks_per_page.
- In code lines 77~79, the reprobe: label. While loop continues.
- Substitute the number of extent added as the value to return on line 80 of the code.
- At line 81 of code, the output argument @span is assigned a number from smallest to largest.
- If the page_no is never incremented in code lines 82~83, it is a blank page. Substitute 1 for page_no.
- In code lines 84~86, update max, pages, highest_bit, etc.
- In code lines 87~88, the out: label is. Returns the number of extents added.
- In code lines 89~92, bad_bmap: Label. Prints the following error message and returns an error of -EINVAL:
    - “swapon: swapfile has holes\n”

swap_extent Structure

include/linux/swap.h

```
1 /*
```

```
 * A swap extent maps a range of a swapfile's PAGE_SIZE pages onto a range of
 * disk blocks.  A list of swap extents maps the entire swapfile.  (Where the
 * term `swapfile' refers to either a blockdevice or an IS_REG file.  Apart
 * from setup, they're handled identically.
 *
 * We always assume that blocks are of size PAGE_SIZE.
 */
```

```
struct swap_extent {
        struct list_head list;
        pgoff_t start_page;
        pgoff_t nr_pages;
        sector_t start_block;
};
```

The swap area is used to map the range to the block device.

- list
  - There are two ways to use it:
    - In the case of a list of swap_extent built into the swap_info_struct, it is used as the list head.
    - The list of swap_extent that is added to the head is used as the node used to add it.
- start_page
  - This is the start page to map in the swap area.
- nr_pages
  - The number of pages to be mapped consecutively from the above start_page in the swap area.
- start_block
  - It is mapped from the start_block of the block device as much as it nr_pages from the start_page of the swap area.

add_swap_extent()

mm/swapfile.c

```
/*
 * Add a block range (and the corresponding page range) into this swapdev's
 * extent list.  The extent list is kept sorted in page order.
 *
 * This function rather assumes that it is called in ascending page order.
 */
```

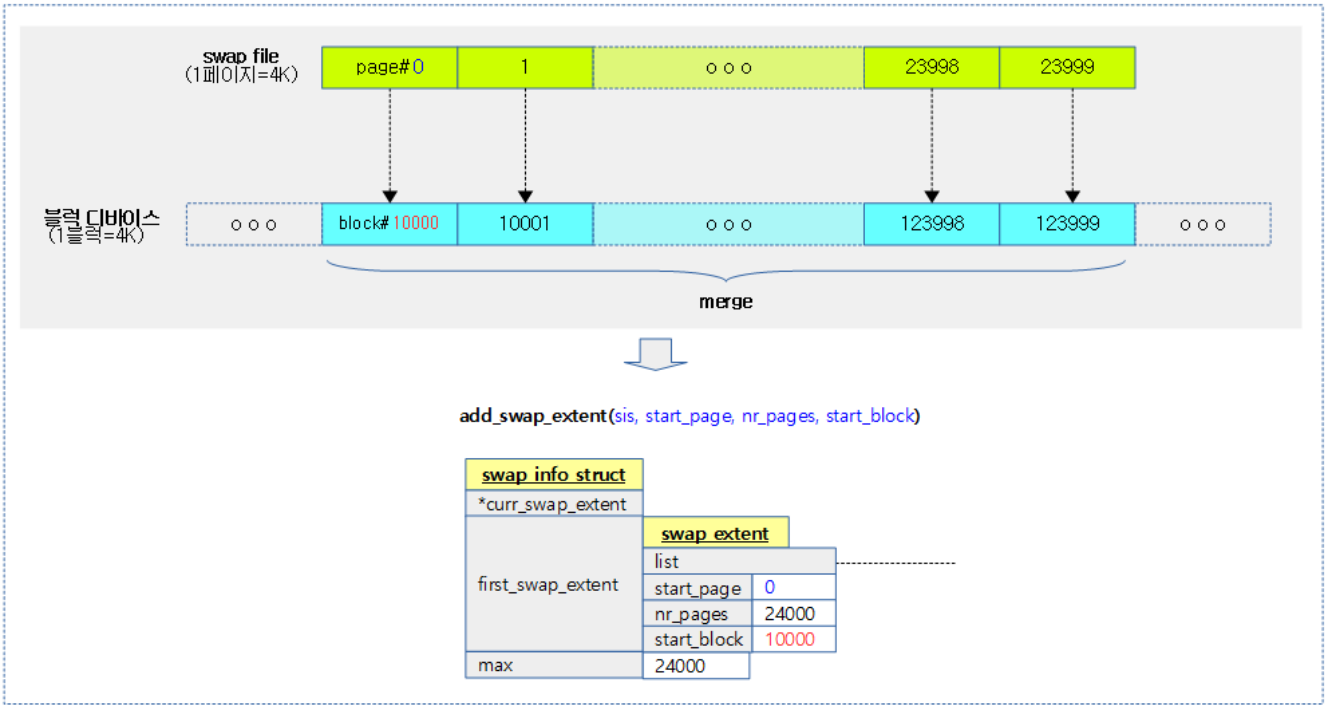
```
int
add_swap_extent(struct swap_info_struct *sis, unsigned long start_page,
                unsigned long nr_pages, sector_t start_block)
{
        struct swap_extent *se;
        struct swap_extent *new_se;
        struct list_head *lh;

        if (start_page == 0) {
                se = &sis->first_swap_extent;
                sis->curr_swap_extent = se;
                se->start_page = 0;
```

```
13                 se->nr_pages = nr_pages;
14                 se->start_block = start_block;
15                 return 1;
16         } else {
17                 lh = sis->first_swap_extent.list.prev;  /* Highest exten
   t */
18                 se = list_entry(lh, struct swap_extent, list);
19                 BUG_ON(se->start_page + se->nr_pages != start_page);
20                 if (se->start_block + se->nr_pages == start_block) {
21                         /* Merge it */
22                         se->nr_pages += nr_pages;
23                         return 0;
24                 }
25         }
26
27         /*
28          * No merge.  Insert a new extent, preserving ordering.
29          */
30         new_se = kmalloc(sizeof(*se), GFP_KERNEL);
31         if (new_se == NULL)
32                 return -ENOMEM;
33         new_se->start_page = start_page;
34         new_se->nr_pages = nr_pages;
35         new_se->start_block = start_block;
36
37         list_add_tail(&new_se->list, &sis->first_swap_extent.list);
38         return 1;
39 }
40 EXPORT_SYMBOL_GPL(add_swap_extent);
```

From the @start_page of the swap area, the @nr_pages is mapped to the @start_block of the block device. If a new swap extent is assigned, it returns 1.

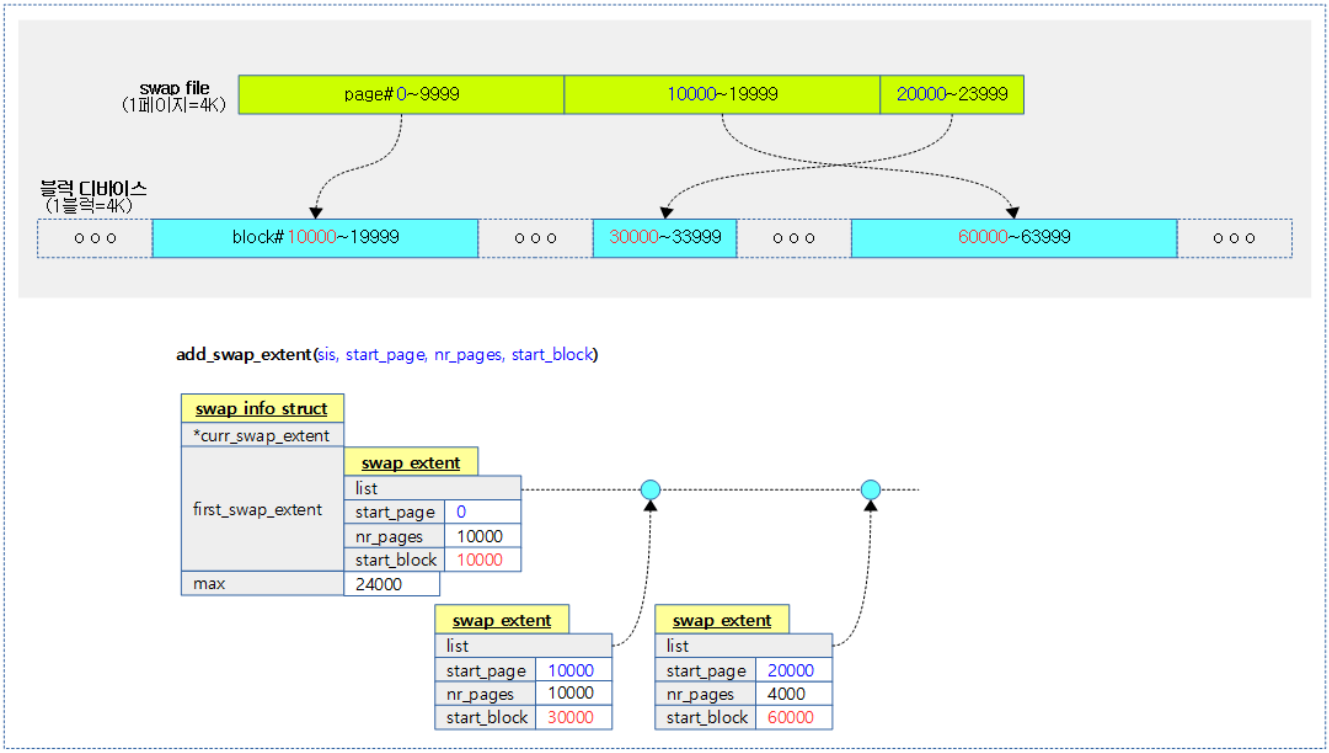
- In line 9~15 of the code, when we first try to map the swap area, we start from page 0, and at this time, we map to @start_block as much as we @nr_pages. The swap_extent used for mapping uses the first_swap_extent that is used as the default inside the swap_info_struct structure. Even if you used the built-in swap extent, it will return 1 to mean that it has been added.
  - When using a block device as a swap area, only one swap_extent is used, so this condition is called only once for one mapping.
- If contiguous blocks are available in code lines 16~25, this is the case used by merging the existing mappings. Since it has been merged, no swap extent is added, so it returns 0.
- In line 30~38 of the code, if the block device number you just mapped is not consecutive, assign a new swap extent and record the new mapping information. Then add it to sis->first_swap_extent.list and return 1.

The following figure shows how a swap file uses only one swap extent when it is used contiguously, rather than fragmenting across mounted block devices.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/add_swap_extent-1.png)

The following figure shows the use of three swap extents when the swap file is used in three fragments on a mounted block device.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/add_swap_extent-2.png)

destroy_swap_extents()

mm/swapfile.c

```
1 /*
2  * Free all of a swapdev's extent information
3  */
```

```
01 static void destroy_swap_extents(struct swap_info_struct *sis)
02 {
03         while (!list_empty(&sis->first_swap_extent.list)) {
04                 struct swap_extent *se;
05
06                 se = list_first_entry(&sis->first_swap_extent.list,
07                                 struct swap_extent, list);
08                 list_del(&se->list);
09                 kfree(se);
10         }
11
12         if (sis->flags & SWP_ACTIVATED) {
13                 struct file *swap_file = sis->swap_file;
14                 struct address_space *mapping = swap_file->f_mapping;
15
16                 sis->flags &= ~SWP_ACTIVATED;
17                 if (mapping->a_ops->swap_deactivate)
18                         mapping->a_ops->swap_deactivate(swap_file);
19         }
20 }
```

Remove all swap extents from the list and deallocate them.

---

address_space management for swap entries

Creating and destructing address_space for swap

address_space structs are created and destroyed by the swapon/swapoff command.

- swapon -> sys_swapon() -> init_swap_address_space()
- swapoff -> sys_swapoff() -> exit_swap_address_space()

init_swap_address_space()

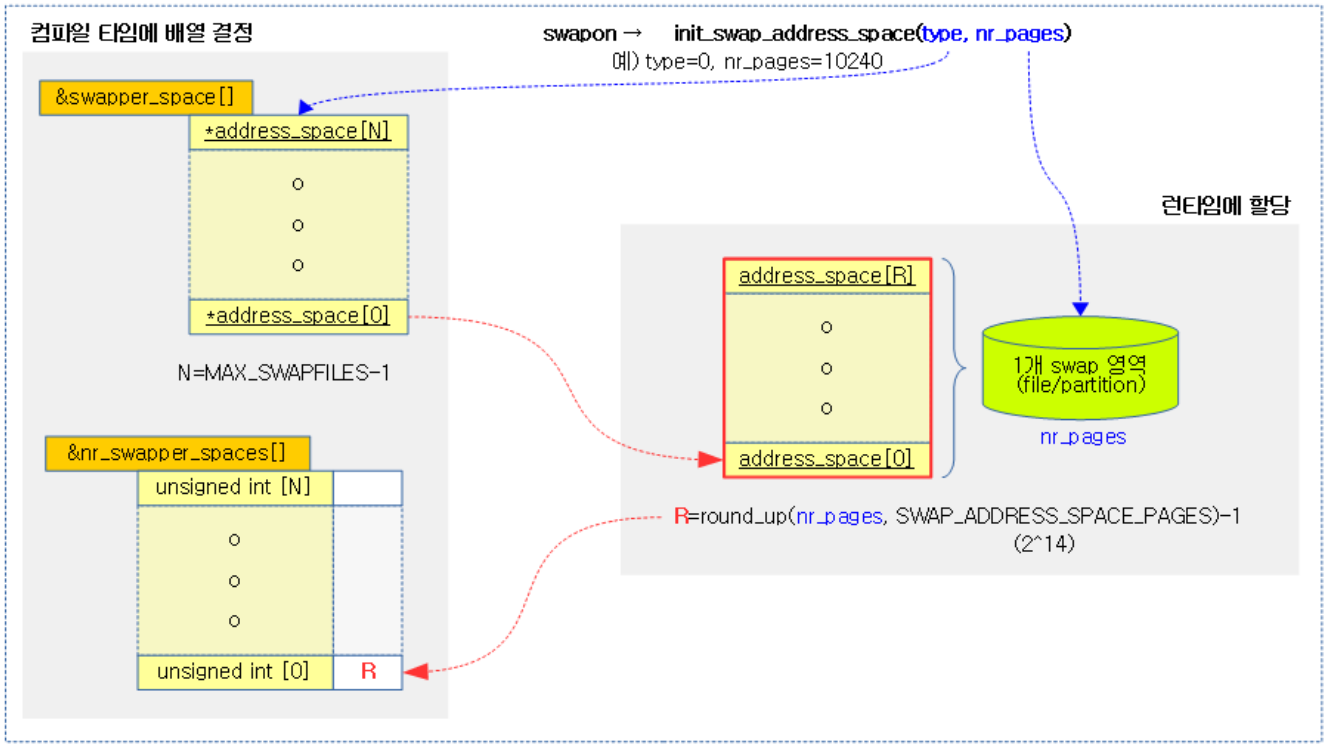
mm/swap_state.c

```
01 int init_swap_address_space(unsigned int type, unsigned long nr_pages)
02 {
03         struct address_space *spaces, *space;
04         unsigned int i, nr;
05
06         nr = DIV_ROUND_UP(nr_pages, SWAP_ADDRESS_SPACE_PAGES);
07         spaces = kvcalloc(nr, sizeof(struct address_space), GFP_KERNEL);
08         if (!spaces)
09                 return -ENOMEM;
10         for (i = 0; i < nr; i++) {
11                 space = spaces + i;
12                 xa_init_flags(&space->i_pages, XA_FLAGS_LOCK_IRQ);
13                 atomic_set(&space->i_mmap_writable, 0);
14                 space->a_ops = &swap_aops;
15                 /* swap cache doesn't use writeback related tags */
16                 mapping_set_no_writeback_tags(space);
17         }
18         nr_swapper_spaces[type] = nr;
19         rcu_assign_pointer(swapper_spaces[type], spaces);
20
21         return 0;
22 }
```

Prepare by allocating a address_space structure for swap.

- Swapon is called to allocate a address_space struct that has been incised in SWAP_ADDRESS_SPACE_PAGES (2^14) increments of the nr_pages to initialize it, and then assign it to swapper_space[@type].
- The swap cache used in the swap area was managed as a single radix tree, but in order to reduce the frequency of rock use and improve performance, only a maximum of 64M was managed for each address_space that manages the swap cache. Therefore, instead of using the swapper_space[] array, it was changed to using the swapper_space[][] double array.

The following figure shows how an array of address_space structures for swaps is created on swapon.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/init_swap_address_space-1.png)

exit_swap_address_space()

mm/swap_state.c

```
01 void exit_swap_address_space(unsigned int type)
02 {
03         struct address_space *spaces;
04
05         spaces = swapper_spaces[type];
06         nr_swapper_spaces[type] = 0;
07         rcu_assign_pointer(swapper_spaces[type], NULL);
08         synchronize_rcu();
09         kvfree(spaces);
10 }
```

swapoff to deallocate the address_space array stored in the swapper_space[@type].

Finding address_space with swap entries

swap_address_space()

include/linux/swap.h

```
#define swap_address_space(entry)                           \
        (&swapper_spaces[swp_type(entry)][swp_offset(entry) \
                >> SWAP_ADDRESS_SPACE_SHIFT])
```

Returns a address_space pointer to the swap entry.

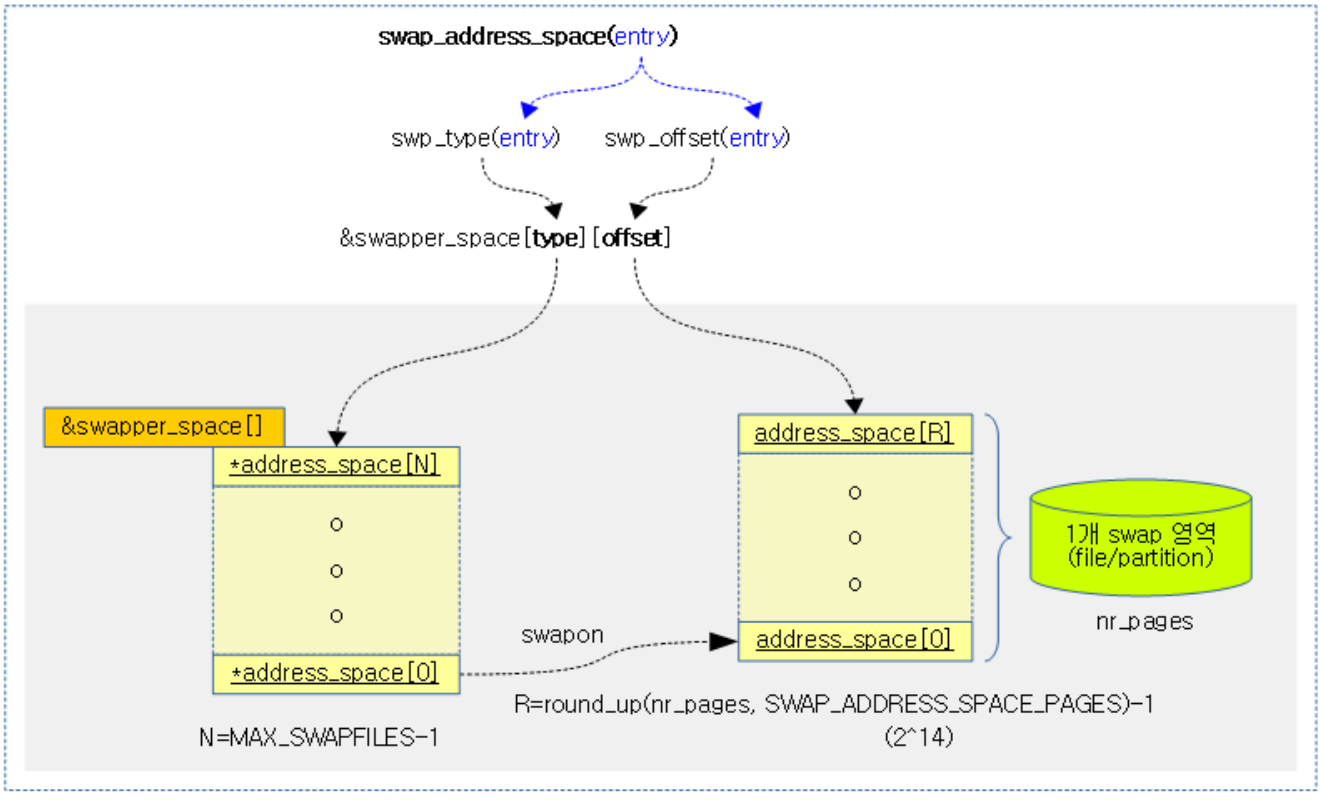
- Returns the address_space pointer specified by the type and offset of the swap entry.

```
/* linux/mm/swap_state.c */
/* One swap address space for each 64M swap space */
```

```
#define SWAP_ADDRESS_SPACE_SHIFT        14
#define SWAP_ADDRESS_SPACE_PAGES        (1 << SWAP_ADDRESS_SPACE_SHIFT)
```

The swap size managed by 1 address_space is 64M.

The following figure shows the process of getting the address_space from the swap_address_space() function to the swap entry.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap_address_space-1.png)

swapper_spaces[]

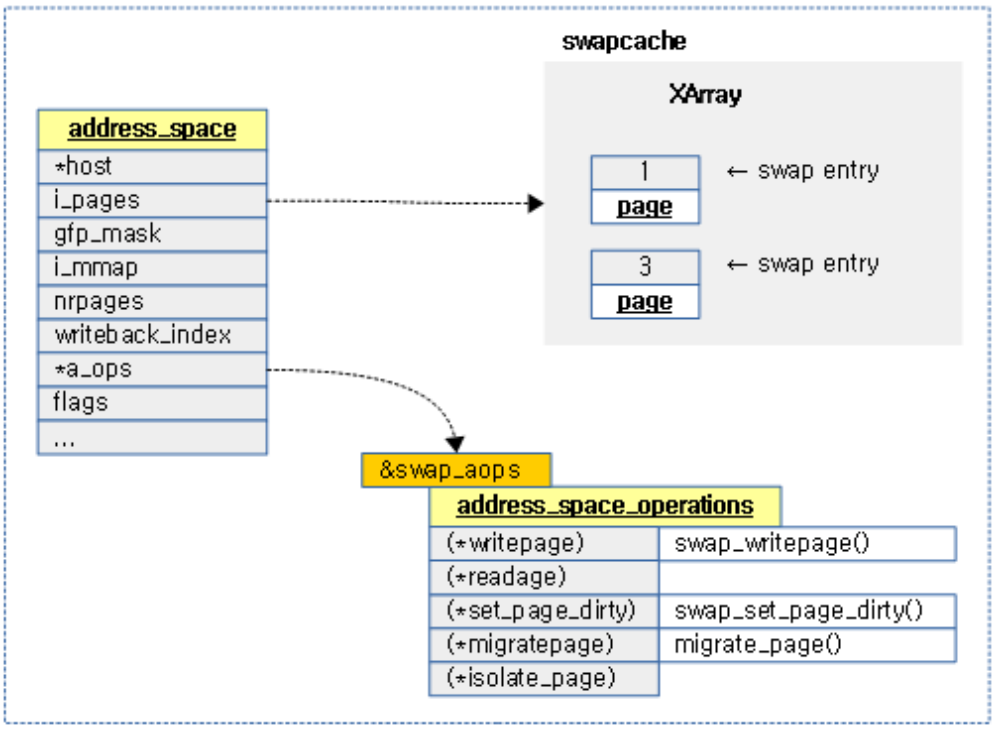
mm/swap_state.c

```
struct address_space *swapper_spaces[MAX_SWAPFILES] __read_mostly;
```

MAX_SWAPFILES (27~32 depending on kernel options, ARM64 default=29) are associated with a number of files, each of which is assigned a address_space array.

- This array is staticly generated as a one-dimensional array, but the actual operation uses it as a two-dimensional array as follows:
  - address_space[type][offset]

The following figure shows the xarray and swap opearations managed by the address_space of the swap cache.



(http://jake.dothome.co.kr/wp-content/uploads/2019/10/swap-18.png)

Structure

swap_info_struct Struct

```
1  /*
2   * The in-memory structure used to track swap areas.
3   */
```

```
01 struct swap_info_struct {
02         unsigned long   flags;          /* SWP_USED etc: see above */
03         signed short    prio;           /* swap priority of this type */
04         struct plist_node list;         /* entry in swap_active_head */
05         signed char     type;           /* strange name for an index */
06         unsigned int    max;            /* extent of the swap_map */
07         unsigned char *swap_map;        /* vmalloc'ed array of usage cou
   nts */
08         struct swap_cluster_info *cluster_info; /* cluster info. Only fo
   r SSD */
09         struct swap_cluster_list free_clusters; /* free clusters list */
10         unsigned int lowest_bit;        /* index of first free in swap_m
   ap */
```

```
11          unsigned int highest_bit;       /* index of last free in swap_ma
    p */
12          unsigned int pages;             /* total of usable pages of swap
    */
13          unsigned int inuse_pages;       /* number of those currently in
    use */
14          unsigned int cluster_next;      /* likely index for next allocat
    ion */
15          unsigned int cluster_nr;        /* countdown to next cluster sea
    rch */
16          struct percpu_cluster __percpu *percpu_cluster; /* per cpu's swa
    p location */
17          struct swap_extent *curr_swap_extent;
18          struct swap_extent first_swap_extent;
19          struct block_device *bdev;      /* swap device or bdev of swap f
    ile */
20          struct file *swap_file;         /* seldom referenced */
21          unsigned int old_block_size;    /* seldom referenced */
22  #ifdef CONFIG_FRONTSWAP
23          unsigned long *frontswap_map;   /* frontswap in-use, one bit per
    page */
24          atomic_t frontswap_pages;       /* frontswap pages in-use counte
    r */
25  #endif
26          spinlock_t lock;                /*
27                                           * protect map scan related fiel
    ds like
28                                           * swap_map, lowest_bit, highest
    _bit,
29                                           * inuse_pages, cluster_next,
30                                           * cluster_nr, lowest_alloc,
31                                           * highest_alloc, free/discard c
    luster
32                                           * list. other fields are only c
    hanged
33                                           * at swapon/swapoff, so are pro
    tected
34                                           * by swap_lock. changing flags
    need
35                                           * hold this lock and swap_lock.
    If
36                                           * both locks need hold, hold sw
    ap_lock
37                                           * first.
38                                           */
39          spinlock_t cont_lock;           /*
40                                           * protect swap count continuati
    on page
41                                           * list.
42                                           */
43          struct work_struct discard_work; /* discard worker */
44          struct swap_cluster_list discard_clusters; /* discard clusters l
    ist */
45          struct plist_node avail_lists[0]; /*
46                                             * entries in swap_avail_head
    s, one
47                                             * entry per node.
48                                             * Must be last as the number
    of the
49                                             * array is nr_node_ids, which
    is not
50                                             * a fixed value so have to al
    locate
51                                             * dynamically.
52                                             * And it has to be an array s
    o that
53                                             * plist_for_each_* can work.
54                                             */
```

```
55 };
```

One is used for each swap area.

- flags
  - These are the flag values used in the swap area. (see below)
- Prio
  - A priority value for ordering the use of zones.
  - The default value is decremented in the order in which it is generated, starting from -2.
  - Positive numbers can be used if specified by the user.
- list
  - swap_active_head node used for the list.
- type
  - The index number for the swap zone. (0~)
- Max
  - Total number of pages in the swap area (including bad pages)
- *swam_map
  - The swap map for the swap area uses a value of 1 byte per page.
  - e.g. 0=free, 1~0x3e: usage counter, 0x3f: bad
- *cluster_info
  - Refers to all the clusters used in the swap zone.
- free_clusters
  - This is a list of available free clusters.
- lowest_bit
  - The lowest offset value of the free page in the swap area.
- highest_bit
  - The highest offset value of the free page in the swap area.
- pages
  - Total number of available pages in the swap area (excluding bad pages)
- inuse_pages
  - The number of pages allocated and in use in the swap area
- cluster_next
  - It points to a page offset that will most likely be used for the next assignment.
- cluster_nr
  - Countdown value to retrieve the following clusters
- *percpu_cluster
  - Refers to the currently specified cluster per CPU.
  - If there is no cluster specified on that CPU, the value of that CPU is null.
- *curr_swap_extent
  - Point to the swap extent you are currently using.
- first_swap_extent
  - This is the first swap extent built into the swap area.
- *bdev
  - The block device specified in the swap area or the block device in the swap file.
- *swap_file

    - swap file.
- old_block_size
    - Size of the swap block device
- *frontswap_map
    - The in-use state of the frontswap is expressed in bits, and 1 bit corresponds to 1 page.
- frontswap_pages
    - FronSwap's In-Use Counter
- lock
    - It is a lock on the swap area.
- cont_lock
    - If the usage counter of the swap_map[] exceeds 0x3e, the map management of the swap area is managed in the swap count continuation mode, which is the lock used to access the swap count continuation page list.
- discard_work
    - This is the workpiece to be used on SSDs that support discard.
- discard_clusters
    - This is a list of clusters to discard.
    - After discarding, move it to the list of free clusters.
- avail_lists[0]
    - It is used as a node in a swap_avail_heads[] list managed by each node.

These are the values used in the flag.

```
01 enum {
02         SWP_USED        = (1 << 0),     /* is slot in swap_info[] used?
   */
03         SWP_WRITEOK     = (1 << 1),     /* ok to write to this swap?
   */
04         SWP_DISCARDABLE = (1 << 2),     /* blkdev support discard */
05         SWP_DISCARDING  = (1 << 3),     /* now discarding a free cluster
   */
06         SWP_SOLIDSTATE  = (1 << 4),     /* blkdev seeks are cheap */
07         SWP_CONTINUED   = (1 << 5),     /* swap_map has count continuati
   on */
08         SWP_BLKDEV      = (1 << 6),     /* its a block device */
09         SWP_ACTIVATED   = (1 << 7),     /* set after swap_activate succe
   ss */
10         SWP_FS          = (1 << 8),     /* swap file goes through fs */
11         SWP_AREA_DISCARD = (1 << 9),    /* single-time swap area discard
   s */
12         SWP_PAGE_DISCARD = (1 << 10),   /* freed swap page-cluster disca
   rds */
13         SWP_STABLE_WRITES = (1 << 11),  /* no overwrite PG_writeback pag
   es */
14         SWP_SYNCHRONOUS_IO = (1 << 12), /* synchronous IO is efficient
   */
15                                         /* add others here before... */
16         SWP_SCANNING    = (1 << 13),    /* refcount in scan_swap_map */
17 };
```

- SWP_USED
    - The swapon is the swap area that is being used.
- SWP_WRITEOK

    - swap area.
- SWP_DISCARDABLE
    - swap area supports discard. (SSD)
- SWP_DISCARDING
    - I'm doing a discard operation on the swap area.
- SWP_SOLIDSTATE
    - The swap area is SSD.
- SWP_CONTINUED
    - In the swap area, the usage counter is in use for more than 0x3e.
- SWP_BLKDEV
    - The swap area is the block device.
- SWP_ACTIVATED
    - The swap zone is set if the swap_activate hook succeeds.
- SWP_FS
    - It is a swap file through the file system.
- SWP_AREA_DISCARD
    - It supports discard only once.
- SWP_PAGE_DISCARD
    - You can freely use the discard on a cluster-by-cluster basis.
- SWP_STABLE_WRITES
    - It is given to an integrity storage device using checksums, etc.
    - SCSI Data Integrity Block Device
- SWP_SYNCHRONOUS_IO
    - This is a swap area where synchronous storage is possible. (RAM, etc.)
- SWP_SCANNING
    - This is set when the swap area is being searched.

consultation

- Swap -1- (Basic, initialization) (http://jake.dothome.co.kr/swap-1) | Question C – Current Article
- Swap -2- (Swapin & Swapout) (http://jake.dothome.co.kr/swap-2) | 문c
- Swap -3- (allocate/unallocate swap area) (http://jake.dothome.co.kr/swap-3) | Qc
- Swap -4- (Swap Entry) (http://jake.dothome.co.kr/swap-entry) | Qc

- Memory Resource Controller (2009) | Kame – Download PDF (https://events.static.linuxfound.org/images/stories/slides/jls09/jls09_kamezawa.pdf?source=post_page----------)
- How Block Device I/O Works (1) (http://egloos.zum.com/studyfoss/v/5575220) | F/OSS
- How Block Device I/O Works (2) (http://studyfoss.egloos.com/5576850) | F/OSS
- How Block Device I/O Works (3) (http://studyfoss.egloos.com/5583458) | F/OSS
- SWAP Management (https://htst.tistory.com/32) | DrakeOh
- Frontswap (https://www.kernel.org/doc/html/latest/vm/frontswap.html) (2012) | Kernel.org

- Cleancache (https://www.kernel.org/doc/html/latest/vm/cleancache.html) (2011) | Kernel.org
- Automatically bind swap device to numa node (https://www.kernel.org/doc/html/latest/vm/swap_numa.html) | Kernel.org
- zsmalloc (https://www.kernel.org/doc/html/latest/vm/zsmalloc.html) | Kernel.org
- zswap (https://www.kernel.org/doc/html/latest/vm/zswap.html) | Kernel.org
- Transcendent memory (2009) (https://lwn.net/Articles/340080/) | LWN.net
- Transcendent memory in a nutshell (https://lwn.net/Articles/454795/) (2011) | LWN.net
- Transcendent Memory and Friends (2011) | Oracle – Download PDF (https://oss.oracle.com/projects/tmem/dist/documentation/presentations/TmemNotVirt-Linuxcon2011-110729.pdf)
- Low RAM on Ubuntu? Using ZRAM (https://muritzy.tistory.com/1653) | Eddy lab

---

4 thoughts to "Swap -1- (Basic, initialization)"

IPARAN (HTTPS://PARANLEE.GITHUB.IO/)
2021-03-13 22:41 (http://jake.dothome.co.kr/swap-1/#comment-304898)

Cleancache entries

The picture below is

cleancache-1.png -I don't see >

cleancache-1a-1.png -> renamed to this image?

Swap Extent Items

swap_extent In the struct image,

In the picture, the start_block on the right intersect

30000 ~ 60000 ~

Is it an arrow that goes up?

RESPONSE (/SWAP-1/?REPLYTOCOM=304898#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2021-03-15 07:23 (http://jake.dothome.co.kr/swap-1/#comment-304903)

Hello?

The cleancache-1.png and cleancache-1a-.png 1 images are both identical and have been removed and renamed .png cleancache-1a.

In the swap extent section, the start_block in the swap_extent structure (30000, 60000) are reversed and corrected.

Thanks for the proofreading. ^^

RESPONSE (/SWAP-1/?REPLYTOCOM=304903#RESPOND)

IPARAN (HTTPS://PARANLEE.GITHUB.IO/)
2021-03-15 10:35 (http://jake.dothome.co.kr/swap-1/#comment-304904)

Yes, thank you as always.

There are a lot of concepts that work in conjunction with other subsystems in the swap content, so I'm going to report it again and again!

RESPONSE (/SWAP-1/?REPLYTOCOM=304904#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2021-03-15 12:43 (http://jake.dothome.co.kr/swap-1/#comment-304905)

네. swap 시스템도 만만치가 않지만 잘 준비하시기 바랍니다.

감사합니다. ^^

응답 (/SWAP-1/?REPLYTOCOM=304905#RESPOND)

댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

VMPressure (http://jake.dothome.co.kr/vmpressure/)

Swap -2- (Swapin & Swapout) (http://jake.dothome.co.kr/swap-2/)

문c 블로그 (2015 ~ 2023)