Slub Memory Allocator -5- (Slub allocation)

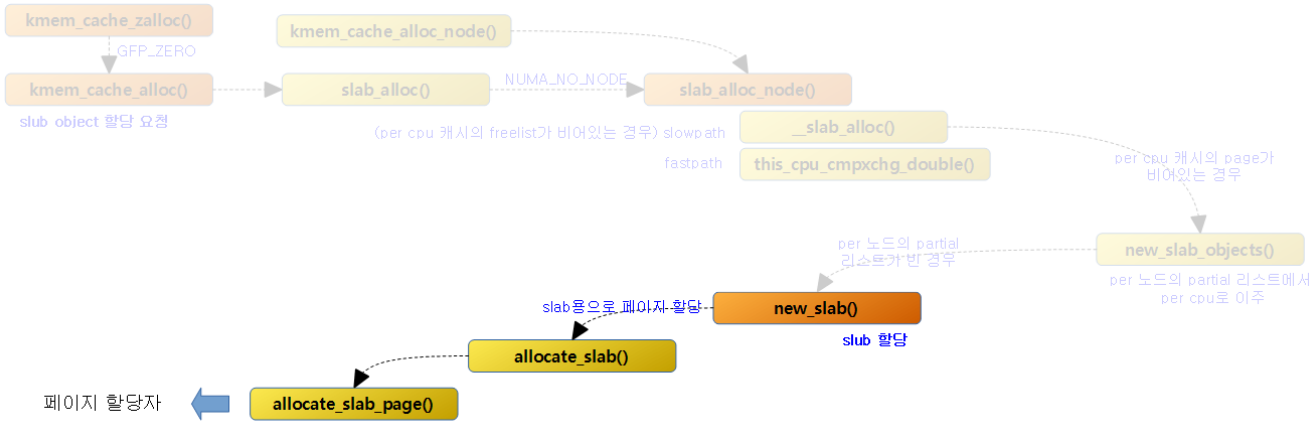
2016-06-07 (http://jake.dothome.co.kr/slub-slub-alloc/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

Slap page assignment

After receiving the number of page frames required to construct a slab page from the page allocator, it initializes it as a slab page.

The following figure shows the new_slab() function as a function flow that requests a new page allocation as the page allocator.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/new_slab-1.png)

Assign a new slap page

new_slab()

mm/slub.c

```
01 static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int nod
   e)
02 {
03         if (unlikely(flags & GFP_SLAB_BUG_MASK)) {
04                 gfp_t invalid_mask = flags & GFP_SLAB_BUG_MASK;
05                 flags &= ~GFP_SLAB_BUG_MASK;
06                 pr_warn("Unexpected gfp: %#x (%pGg). Fixing up to gfp: %
   #x (%pGg). Fix your code!\n""
07 ,
08                                 invalid_mask, &invalid_mask, flags, &fla
   gs);
09                 dump_stack();
10         }
11
12         return allocate_slab(s,
```

```
13                  flags & (GFP_RECLAIM_MASK | GFP_CONSTRAINT_MASK), node);
14  }
```

You will be assigned a new slap page by the buddy system.

- If you make a dma3 and highmem request through the slab page allocator in line 10~32 of the code, it will print out the error and call BUG().
- In code lines 12~13, a new slap page is allocated, and the allowed gfp flags are the following:

GFP_SLAB_BUG_MASK

mm/internal.h

```
1  /* Do not use these with a slab allocator */
2  #define GFP_SLAB_BUG_MASK (__GFP_DMA32|__GFP_HIGHMEM|~__GFP_BITS_MASK)
```

These are gfp flags that cannot be requested from the slab allocator.

GFP_RECLAIM_MASK

mm/internal.h

```
1  /*
2   * The set of flags that only affect watermark checking and reclaim
3   * behaviour. This is used by the MM to obey the caller constraints
4   * about IO, FS and watermark checking while ignoring placement
5   * hints such as HIGHMEM usage.
6   */
```

```
1  #define GFP_RECLAIM_MASK (__GFP_RECLAIM|__GFP_HIGH|__GFP_IO|__GFP_FS|\
2                            __GFP_NOWARN|__GFP_RETRY_MAYFAIL|__GFP_NOFAIL|\
3                            __GFP_NORETRY|__GFP_MEMALLOC|__GFP_NOMEMALLOC|\
4                            __GFP_ATOMIC)
```

These are the reclaim-related gfp flags that can be requested from the slab allocator.

GFP_CONSTRAINT_MASK

mm/internal.h

```
1  /* Control allocation cpuset and node placement constraints */
2  #define GFP_CONSTRAINT_MASK (__GFP_HARDWALL|__GFP_THISNODE)
```

These are the gfp flags for cpuset and node that can be requested from the slab allocator.

allocate_slab()

mm/slub.c

```
01  static struct page *allocate_slab(struct kmem_cache *s, gfp_t flags, int
    node)
02  {
03          struct page *page;
04          struct kmem_cache_order_objects oo = s->oo;
05          gfp_t alloc_gfp;
06          void *start, *p, *next;
07          int idx, order;
```

```
08          bool shuffle;
09
10          flags &= gfp_allowed_mask;
11
12          if (gfpflags_allow_blocking(flags))
13                  local_irq_enable();
14
15          flags |= s->allocflags;
16
17          /*
18           * Let the initial higher-order allocation fail under memory pre
    ssure
19           * so we fall-back to the minimum order allocation.
20           */
21          alloc_gfp = (flags | __GFP_NOWARN | __GFP_NORETRY) & ~__GFP_NOFA
    IL;
22          if ((alloc_gfp & __GFP_DIRECT_RECLAIM) && oo_order(oo) > oo_orde
    r(s->min))
23                  alloc_gfp = (alloc_gfp | __GFP_NOMEMALLOC) & ~(__GFP_REC
    LAIM|__GFP_NOFAIL);
24
25          page = alloc_slab_page(s, alloc_gfp, node, oo);
26          if (unlikely(!page)) {
27                  oo = s->min;
28                  alloc_gfp = flags;
29                  /*
30                   * Allocation may have failed due to fragmentation.
31                   * Try a lower order alloc if possible
32                   */
33                  page = alloc_slab_page(s, alloc_gfp, node, oo);
34                  if (unlikely(!page))
35                          goto out;
36                  stat(s, ORDER_FALLBACK);
37          }
38
39          page->objects = oo_objects(oo);
40
41          order = compound_order(page);
42          page->slab_cache = s;
43          __SetPageSlab(page);
44          if (page_is_pfmemalloc(page))
45                  SetPageSlabPfmemalloc(page);
46
47          kasan_poison_slab(page);
48
49          start = page_address(page);
50
51          setup_page_debug(s, start, order);
52
53          shuffle = shuffle_freelist(s, page);
54
55          if (!shuffle) {
56                  start = fixup_red_left(s, start);
57                  start = setup_object(s, page, start);
58                  page->freelist = start;
59                  for (idx = 0, p = start; idx < page->objects - 1; idx++)
    {
60                          next = p + s->size;
61                          next = setup_object(s, page, next);
62                          set_freepointer(s, p, next);
63                          p = next;
64                  }
65                  set_freepointer(s, p, NULL);
66          }
67
68          page->inuse = page->objects;
69          page->frozen = 1;
70
```

```
71  out:
72          if (gfpflags_allow_blocking(flags))
73                  local_irq_disable();
74          if (!page)
75                  return NULL;
76
77          mod_lruvec_page_state(page,
78                  (s->flags & SLAB_RECLAIM_ACCOUNT) ?
79                  NR_SLAB_RECLAIMABLE : NR_SLAB_UNRECLAIMABLE,
80                  1 << oo_order(oo));
81
82          inc_slabs_node(s, page_to_nid(page), page->objects);
83
84          return page;
85  }
```

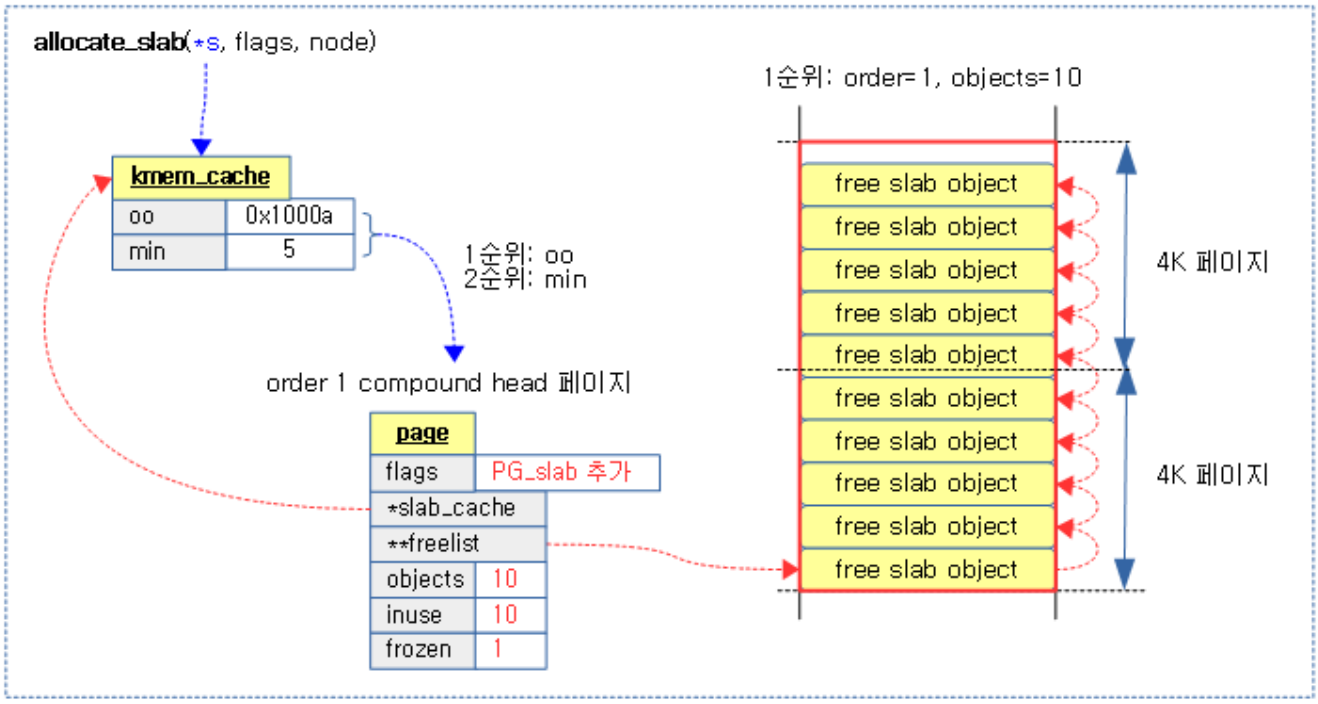
It takes the slub page assignment and initializes the free object contained in the page.

- Connect all the free objects in order, and null the connection of the last object.
  - In the slub, free objects are connected in one direction.
  - When a slub is first allocated, all obects on it are free, so the entire object is concatenated in order.

- On lines 10~15 of code, if direct-reclaim is allowed outside of boot time, you should enable irq.
  - It doesn't allow io, fs, or reclaim at boot time.
- In lines 21~23 of code, add nowarn and noretry to the alloc_gfp flag and remove the nofail flag. If direct-reclaim is allowed, add nomemalloc, and remove reclaim and nofail.
- In lines 25~37 of code, we assign a slap page with s->oo. If it fails, s->min is used to allocate the slab page.
- In line 39~69 of the code, initialize the page descriptor of the slab page as follows, and concatenate all the FPs of the free objects. One of the security options, CONFIG_SLAB_FREELIST_RANDOM kernel options, allows you to randomly reorder free objects. In addition, CONFIG_SLAB_FREELIST_HARDENED kernel options can be used to encaptualize the contents of FPs and hide them from prying eyes.
  - The fixup_red_left() function returns the object address by moving the object address by red_left_pad amount if the red-zone was used.
  - page->objects stores the number of objects corresponding to the determined order.
  - page->slab_cache to specify the slab cache.
  - Add PG_slab flag.
  - If you used the pfmemalloc page, add the PG_active flag.
  - page->inuse.
  - Assign 1 to page->frozen. (This will be a partial list of per-cpu caches.)
  - If poison debugging is enabled, POISON_INUSE (0x55) is recorded throughout the slab page to indicate that the first slab object was not used.
- In code lines 71~75, the out: label. If the function initially enabled the irq, disable it.
- In line 77~80 of the code, increment the NR_SLAB_RECLAIMABLE counter if you allocated a reclaimable slap page, otherwise increment the NR_SLAB_UNRECLAIMABLE counter.
- In line 82 of code, increment s->node[]->nr_slabs by 1 and add the number of objects assigned to the s->node[]->total_objects counter.

- Returns the slap page assigned in line 84 of code.

The following image shows how the order 1 slab page has been allocated and initialized.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/allocate_slab-1.png)

Slub with PFMEMALLOC

- If you use the PF_MEMALLOC or TIF_MEMDIE flags, you can use the ALLOC_NO_WATERMARKS flag to secure pages even if you don't have enough free pages and they fall below the low watermark criteria.
  - The page is set to page->pfmemalloc.
    - get_page_from_freelist() -> after the buddy assignment -> prep_new_page() is set as follows in the function.
      - page->pfmemalloc = !! (alloc_flags & ALLOC_NO_WATERMARKS);
  - Where to use PF_MEMALLOC in Slub
    - Swap over the NBD(Network Block Device)
      - It can be swapped via TCP communication with the NBD (Network Block Device) driver.
        - SOCK_MEMALLOC flag uses PF_MEMALLOC.
      - If you are using a network-based swap instead of a disk-based swap, you will be able to use a lot of small buffers related to SKB when you run out of memory, and you can use this pfmemalloc configured page to facilitate the allocation.
      - Set the Active flag of the slub page to distinguish which slub pages are assigned with this pfmemalloc situation.
        - This active slub page would not be available in normal cases.

set_freepointer()

mm/slub.c

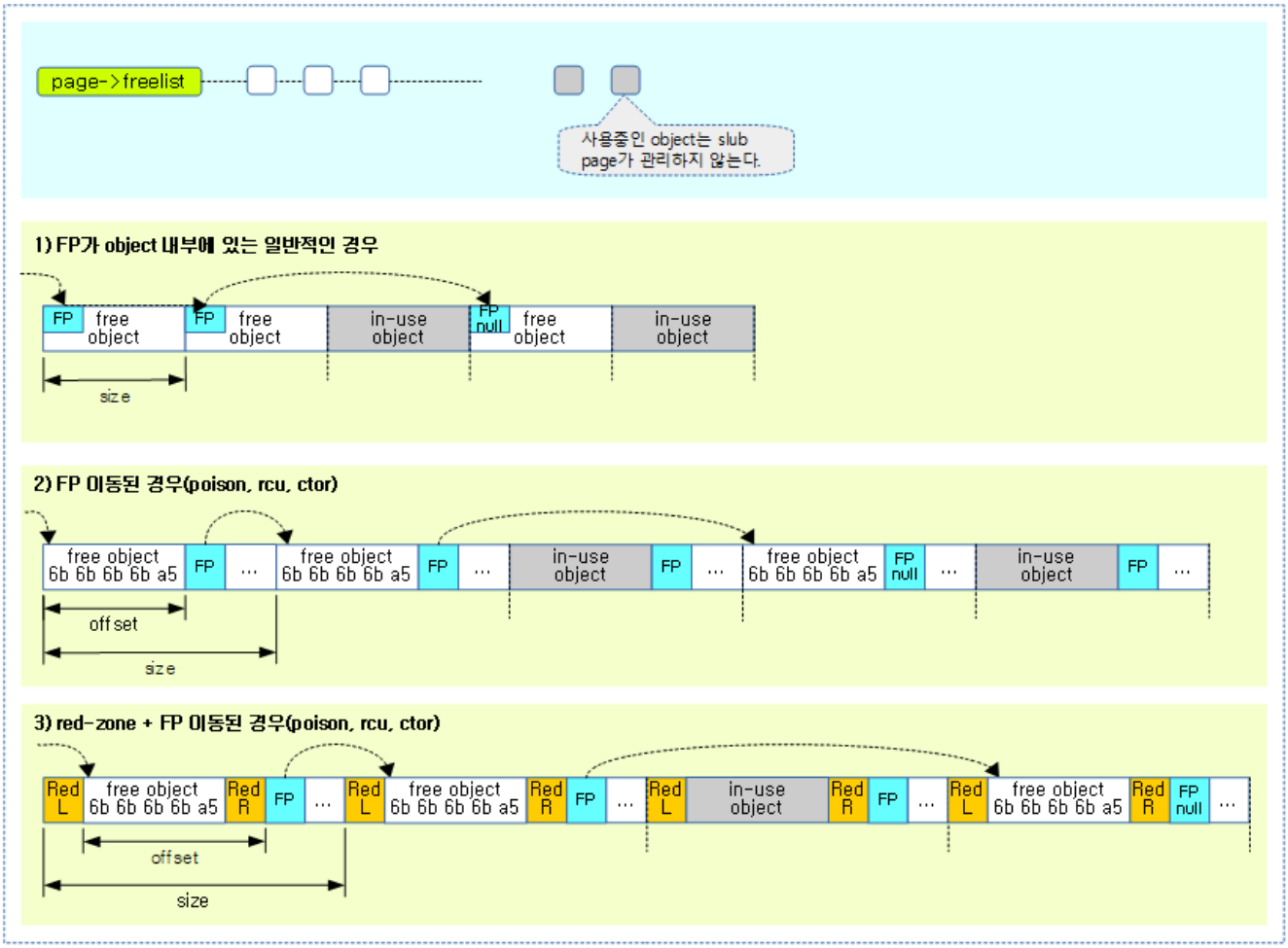
```
01 static inline void set_freepointer(struct kmem_cache *s, void *object, v
   oid *fp)
02 {
03         unsigned long freeptr_addr = (unsigned long)object + s->offset;
04
05 #ifdef CONFIG_SLAB_FREELIST_HARDENED
06         BUG_ON(object == fp); /* naive detection of double free or corru
   ption */
07 #endif
08
09         *(void **)freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
10 }
```

Write the address of the next object at the FP location of the slub object. If CONFIG_SLAB_FREELIST_HARDENED kernel options are used, the FP contents are encaptualized unrecognizable.

- In line 3 of code, the Free Pointer (FP) always skips as much as s->offset on the object.
- In line 9 of code, write the following free object address at the FP location:

The following figure shows the FP(Free Pointer) position and offset values in the general case where the FP is inside the object and the FP is placed after the object in a slub page with a maximum of 5 objects.

- SLAB_POISON, SLAB_DESTROY_BY_RCU, and a slap cache with separate constructors will move the FP behind the object as shown in 2).



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/set_freepointer-1f.png)

gfp_allowed_mask

mm/page_alloc.c

```
1 gfp_t gfp_allowed_mask __read_mostly = GFP_BOOT_MASK;
```

It has all the bits except for the flag, which should not be used at boot time.

GFP_BOOT_MASK

mm/internal.h

```
1 /* Control slab gfp mask during early boot */
2 #define GFP_BOOT_MASK (__GFP_BITS_MASK & ~(__GFP_RECLAIM|__GFP_IO|__GFP_
  FS))
```

- GFP_BOOT_MASK
  - Requests for the reclaim, io, and fs flags should not be used during the kernel boot process, so they should be removed from the GFP standard mask.

alloc_slab_page()

mm/slub.c

```
1 /*
```

```
 * Slab allocation and freeing
 */
```

```
01 static inline struct page *alloc_slab_page(struct kmem_cache *s,
02                 gfp_t flags, int node, struct kmem_cache_order_objects o
   o)
03 {
04         struct page *page;
05         unsigned int order = oo_order(oo);
06
07         if (node == NUMA_NO_NODE)
08                 page = alloc_pages(flags, order);
09         else
10                 page = __alloc_pages_node(node, flags, order);
11
12         if (page && memcg_charge_slab(page, flags, order, s)) {
13                 __free_pages(page, order);
14                 page = NULL;
15         }
16
17         return page;
18 }
```

Under the control of the Memory Control Group, you are assigned a page to use in the slab. If it fails, it returns null.

- If no node is specified in code lines 7~10, the appropriate node handles the order page assignment, and if a node is specified, the order page is assigned from that node.
- In line 12~15 of the code, the memory control group is asked to assign the order page, and the slab can be deassigned through the control.
- Returns the page assigned in line 17 of code.

inc_slabs_node()

mm/slub.c

```
01 static inline void inc_slabs_node(struct kmem_cache *s, int node, int ob
   jects)
02 {
03         struct kmem_cache_node *n = get_node(s, node);
04
05         /*
06          * May be called early in order to allocate a slab for the
07          * kmem_cache_node structure. Solve the chicken-egg
08          * dilemma by deferring the increment of the count during
09          * bootstrap (see early_kmem_cache_node_alloc).
10          */
11         if (likely(n)) {
12                 atomic_long_inc(&n->nr_slabs);
13                 atomic_long_add(objects, &n->total_objects);
14         }
15 }
```

Only if you use the CONFIG_SLUB_DEBUG kernel option will you increase the nr_slabs for the per node with a high probability and add as much as objects to the total_objects.

SetPageSlabPfmemalloc()

include/linux/page-flags.h

```
static inline void SetPageSlabPfmemalloc(struct page *page)
{
        VM_BUG_ON_PAGE(!PageSlab(page), page);
        SetPageActive(page);
}
```

Set the PG_Active flag on the page.

consultation

- Slab Memory Allocator -1- (Structure) (http://jake.dothome.co.kr/slub/) | Qc
- Slab Memory Allocator -2- (Initialize Cache) (http://jake.dothome.co.kr/kmem_cache_init) | Qc
- Slub Memory Allocator -3- (Create Cache) (http://jake.dothome.co.kr/slub-cache-create) | Qc
- Slub Memory Allocator -4- (Calculate Order) (http://jake.dothome.co.kr/slub-order) | Qc
- Slub Memory Allocator -5- | (http://jake.dothome.co.kr/slub-slub-alloc) Sentence C – Current post
- Slub Memory Allocator -6- (Assign Object) (http://jake.dothome.co.kr/slub-object-alloc) | Qc
- Slub Memory Allocator -7- (Object Unlocked) (http://jake.dothome.co.kr/slub-object-free) | Qc
- Slub Memory Allocator -8- (Drain/Flash Cache) (http://jake.dothome.co.kr/slub-drain-flush-cache) | Qc
- Slub Memory Allocator -9- (Cache Shrink) (http://jake.dothome.co.kr/slub-cache-shrink) | Qc
- Slub Memory Allocator -10- | (http://jake.dothome.co.kr/slub-slub-free) Qc
- Slub Memory Allocator -11- (Clear Cache (http://jake.dothome.co.kr/slub-cache-destroy)) | Qc
- Slub Memory Allocator -12- (Debugging Slub) (http://jake.dothome.co.kr/slub-debug) | Qc
- Slub Memory Allocator -13- (slabinfo) (http://jake.dothome.co.kr/slub-slabinfo) | 문c

---

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

‹ Slub Memory Allocator -7- (Object Unlocked) (http://jake.dothome.co.kr/slub-object-free/)

Slub Memory Allocator -11- (Clear Cache) › (http://jake.dothome.co.kr/slub-cache-destroy/)

Munc Blog (2015 ~ 2024)