Slub Memory Allocator -9- (cache shrink)

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

<kernel v5.0>

kmem_cache_shrink()

mm/slab_common.c

```
1  /**
2   * kmem_cache_shrink - Shrink a cache.
3   * @cachep: The cache to shrink.
4   *
5   * Releases as many slabs as possible for a cache.
6   * To help debugging, a zero exit status indicates all slabs were releas
    ed.
7   */
```

```
01 int kmem_cache_shrink(struct kmem_cache *cachep)
02 {
03         int ret;
04
05         get_online_cpus();
06         get_online_mems();
07         ret = __kmem_cache_shrink(cachep);
08         put_online_mems();
09         put_online_cpus();
10         return ret;
11 }
12 EXPORT_SYMBOL(kmem_cache_shrink);
```

Find the revocable slab pages in the requested slab cache and cancel all of them.

__kmem_cache_shrink()

mm/slub.c

```
1  /*
2   * kmem_cache_shrink discards empty slabs and promotes the slabs filled
3   * up most to the head of the partial lists. New allocations will then
4   * fill those up and thus they can be removed from the partial lists.
5   *
6   * The slabs with the least items are placed last. This results in them
7   * being allocated from last increasing the chance that the last objects
8   * are freed in them.
9   */
```

```
01 int __kmem_cache_shrink(struct kmem_cache *s)
02 {
03         int node;
04         int i;
05         struct kmem_cache_node *n;
06         struct page *page;
07         struct page *t;
08         struct list_head discard;
```

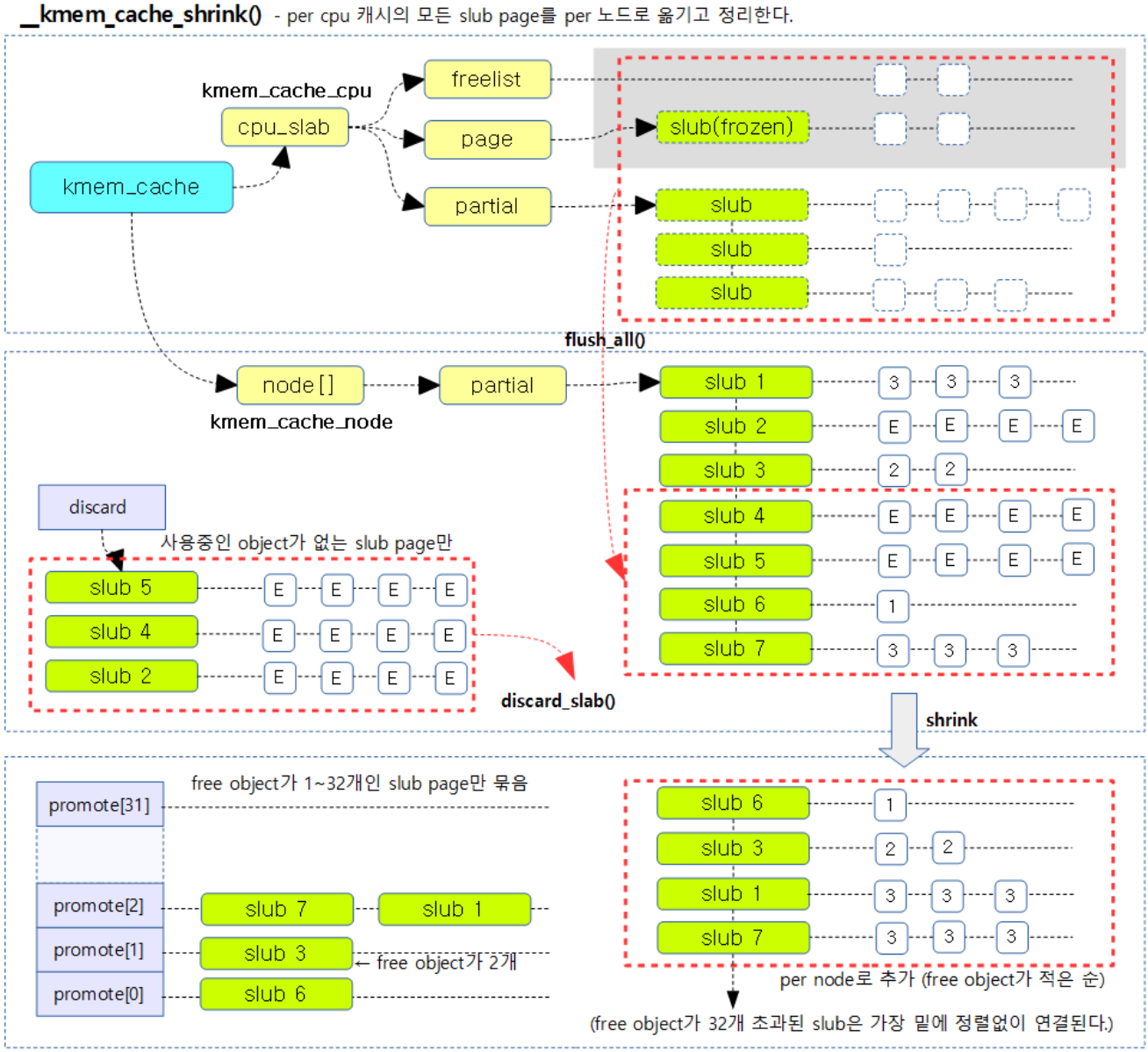
```
09          struct list_head promote[SHRINK_PROMOTE_MAX];
10          unsigned long flags;
11          int ret = 0;
12
13          flush_all(s);
14          for_each_kmem_cache_node(s, node, n) {
15                  INIT_LIST_HEAD(&discard);
16                  for (i = 0; i < SHRINK_PROMOTE_MAX; i++)
17                          INIT_LIST_HEAD(promote + i);
18
19                  spin_lock_irqsave(&n->list_lock, flags);
20
21                  /*
22                   * Build lists of slabs to discard or promote.
23                   *
24                   * Note that concurrent frees may occur while we hold th
    e
25                   * list_lock. page->inuse here is the upper limit.
26                   */
27                  list_for_each_entry_safe(page, t, &n->partial, lru) {
28                          int free = page->objects - page->inuse;
29
30                          /* Do not reread page->inuse */
31                          barrier();
32
33                          /* We do not keep full slabs on the list */
34                          BUG_ON(free <= 0);
35
36                          if (free == page->objects) {
37                                  list_move(&page->lru, &discard);
38                                  n->nr_partial--;
39                          } else if (free <= SHRINK_PROMOTE_MAX)
40                                  list_move(&page->lru, promote + free -
    1);
41                  }
42
43                  /*
44                   * Promote the slabs filled up most to the head of the
45                   * partial list.
46                   */
47                  for (i = SHRINK_PROMOTE_MAX - 1; i >= 0; i--)
48                          list_splice(promote + i, &n->partial);
49
50                  spin_unlock_irqrestore(&n->list_lock, flags);
51
52                  /* Release empty slabs */
53                  list_for_each_entry_safe(page, t, &discard, lru)
54                          discard_slab(s, page);
55
56                  if (slabs_node(s, node))
57                          ret = 1;
58          }
59
60          return ret;
61  }
```

Perform the following procedure to find revocable slab pages in the requested slab cache and delete all of them.

- If none of the slab pages in the n->partial list have an object in use, the slab pages are returned to the buddy system.
- Fewer than 32 free objects in the slab pages of an n->partial list are asscending sorted.

- In line 13 of code, move all the per cpu slab pages in the slab cache to an n->partial list.
- In lines 14~17 of the code, traverse the nodes in the slab cache and initialize the discard list, and initialize the promote[] list by SHRINK_PROMOTE_MAX(32).
- In code lines 27~41, traverse the slab pages of the n->partial list, adding the slab pages with all the objects of free objects to the discard list, and if the number of free objects is less than SHRINK_PROMOTE_MAX(32), add them to the promote[free-1] list.
- In lines 47~48 of code, add the promote[i] list to the beginning of the n->partial list.
    - In the end, the free objects are sorted in the order of 1~32 and added to the lead.
    - Place the slab page with the fewest free objects at the top so that the slab page can be quickly removed from the list management when the object is assigned.
- In code lines 53~54, unload all the slap pages in the discard list and return them to the buddy system.
- In line 56~57 of the code, if there is a slap left on the node during slub debugging, the function will return 1 at the exit of the function.

The following illustration shows all the slab pages in the current slab cache being moved to an n->partial list and organized.



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

kick_all_cpus_sync()

kernel/smp.c

```
/**
 * kick_all_cpus_sync - Force all cpus out of idle
 *
 * Used to synchronize the update of pm_idle function pointer. It's
 * called after the pointer is updated and returns after the dummy
 * callback function has been executed on all cpus. The execution of
 * the function can only happen on the remote cpus after they have
 * left the idle function which had been called via pm_idle function
 * pointer. So it's guaranteed that nothing uses the previous pointer
 * anymore.
 */
```

```
void kick_all_cpus_sync(void)
{
        /* Make sure the change is visible before we kick the cpus */
        smp_mb();
        smp_call_function(do_nothing, NULL, 1);
}
EXPORT_SYMBOL_GPL(kick_all_cpus_sync);
```

Use an IPI call to call each cpu to perform a dummy callback.

- pm_idle used to synchronize the update of the function pointer.
- do_nothing() is an empty function.

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) Qc
- 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) | Sentence C – Current post
- 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 -11- (Clear Cache) (http://jake.dothome.co.kr/slub-cache-destroy/)

ZONE Type › (http://jake.dothome.co.kr/zone-types/)

Munc Blog (2015 ~ 2024)