Zoned Allocator -4- (Buddy page terminated)

2016-05-12 (http://jake.dothome.co.kr/buddy-free/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

Related kernel options

CONFIG_KMEMCHECK

- It is only supported on the x86 architecture and is a kernel option that allows you to dynamically tracing the allocated kernel memory.
- You can enable/disable it using the "kmemcheck=0" or "kmemcheck=1" early kernel parameters in cmdline.

CONFIG_MEMORY_ISOLATION

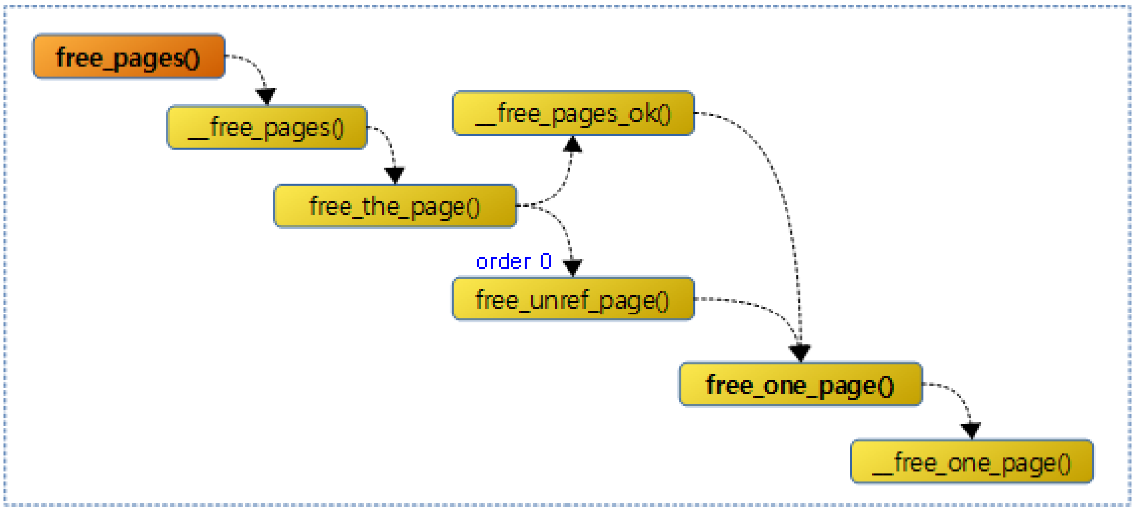
- Recent kernels have been CONFIG_{CMA | MEMORY_HOTPLUG | MEMORY_FAILURE} If you do not use the kernel option and use the CONFIG_MEMORY_ISOLATION kernel option, you can use the isolation function of memory.
- rpi2: Use this kernel option.

CONFIG_KASAN

- Kernel Address Sanitizer (KASAN) – Runtime memory debugger for SLUB
- Using this feature degrades performance by up to 3 times and consumes about 1/8 of free memory.
- For better error detection, use "slub_debug=U" in CONFIG_STACKTRACE and cmdline.

Page Cancellation (Free)

The following figure shows the process of calling after the free_pages() function.



(http://jake.dothome.co.kr/wp-content/uploads/2016/05/free_pages-1b.png)

free_pages()

mm/page_alloc.c

```
void free_pages(unsigned long addr, unsigned int order)
{
        if (addr != 0) {
                VM_BUG_ON(!virt_addr_valid((void *)addr));
                __free_pages(virt_to_page((void *)addr), order);
        }
}

EXPORT_SYMBOL(free_pages);
```

The Buddy system unassigns @order pages from the @addr address.

__free_pages()

mm/page_alloc.c

```
void __free_pages(struct page *page, unsigned int order)
{
        if (put_page_testzero(page))
                free_the_page(page, order);
}
EXPORT_SYMBOL(__free_pages);
```

The reference counter decrements, and when it reaches zero, the buddy system deallocates the @order pages.

free_the_page()

mm/page_alloc.c

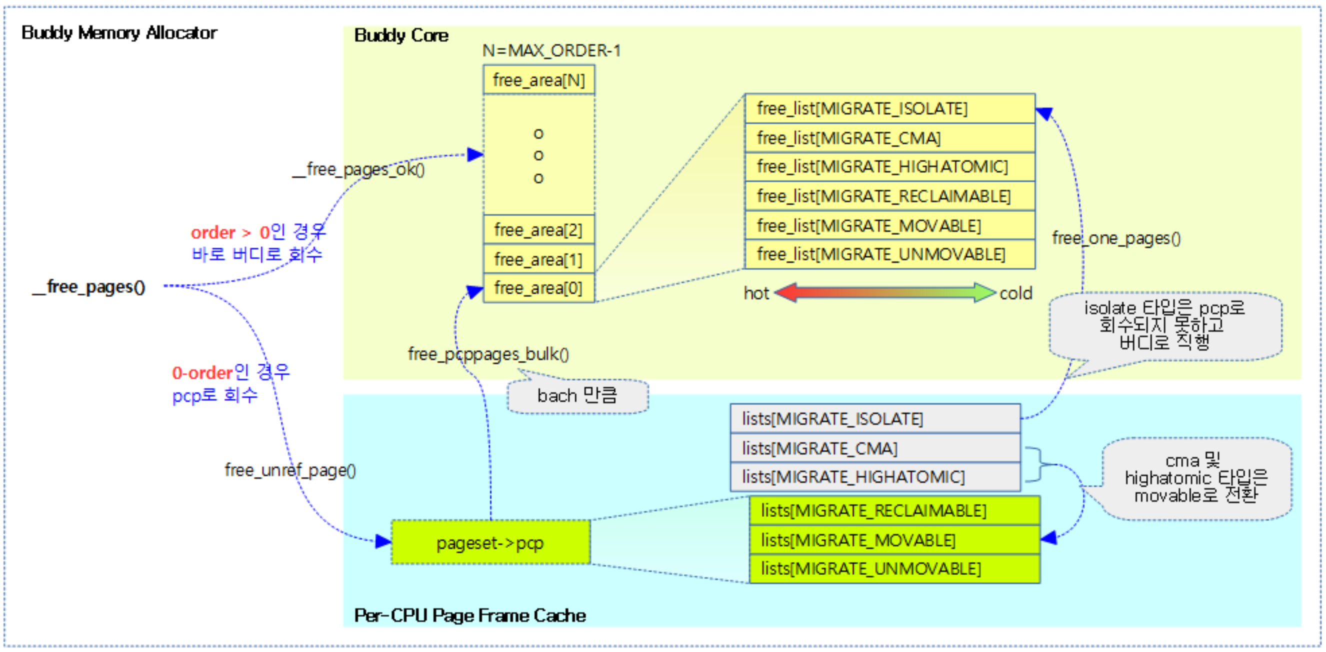
```
static inline void free_the_page(struct page *page, unsigned int order)
{
        if (order == 0)         /* Via pcp? */
                free_unref_page(page);
        else
                __free_pages_ok(page, order);
}
```

If it is a 2^order page that has been used, it will be free, but in the case of a 0-order page, it will try to free it to Per CPU Page Frame Cache.

- Migrate the 0-order page to the hot page of pcp.
- When migrating PCP, the processing depends on the migrate type.
    - Unmovable, reclaimable, and movable types are migrated as they are.
    - If it is a reserve, CMA type, it is converted to a movable type and migrated.
    - If it is an isolate type, it is not migrated to PCP, but sent back to buddy.

The figure below shows how the order bit is processed.

- When freeing a single page with order=0
    - PCP, and in the case of the isolate type, the page is retrieved to the buddy system. And the CMA and HighAtomic types are recovered by the movable type PCP.
    - If it is overflow when migrating PCP, it will be recalled by the buddy system by the number of batches.
- When order is non-zero and multi page free.
    - It does not use PCP and collects it with the buddy system.



(http://jake.dothome.co.kr/wp-content/uploads/2016/05/free_pages-2b.png)

__free_pages_ok()

mm/page_alloc.c

```
static void __free_pages_ok(struct page *page, unsigned int order)
{
        unsigned long flags;
        int migratetype;
        unsigned long pfn = page_to_pfn(page);

        if (!free_pages_prepare(page, order, true))
                return;
09
10          migratetype = get_pfnblock_migratetype(page, pfn);
11          local_irq_save(flags);
12          __count_vm_events(PGFREE, 1 << order);
13          free_one_page(page_zone(page), page, pfn, order, migratetype);
14          local_irq_restore(flags);
15  }
```

The order page that has been used is retrieved by the buddy system.

- Check the pages you want to unassign in line 7~8 of the code.
- On line 12 of code, increment the PGFREE counter by the number of pages.
- At line 13 of code, the order page is retrieved to the buddy system.

free_one_page()

mm/page_alloc.c

```
01  static void free_one_page(struct zone *zone,
02                                  struct page *page, unsigned long pfn,
03                                  unsigned int order,
04                                  int migratetype)
05  {
06          spin_lock(&zone->lock);
07          if (unlikely(has_isolate_pageblock(zone) ||
08                  is_migrate_isolate(migratetype))) {
09                  migratetype = get_pfnblock_migratetype(page, pfn);
10          }
11          __free_one_page(page, pfn, zone, order, migratetype);
12          spin_unlock(&zone->lock);
13  }
```

The order page that has been used is retrieved by the buddy system.

- In line 7~10 of the code, if there is a small probability that an isolate type exists in a zone, or if an isolate type is specified as an argument, use the migratetype of the pageblock to which the page belongs.
- In line 11 of the code, the order page is recalled to the migrate type of the buddy system.

__free_one_page()

mm/page_alloc.c -1/2-

```
01  /*
02   * Freeing function for a buddy system allocator.
03   *
04   * The concept of a buddy system is to maintain direct-mapped table
05   * (containing bit values) for memory blocks of various "orders".
06   * The bottom level table contains the map for the smallest allocatable
07   * units of memory (here, pages), and each level above it describes
08   * pairs of units from the levels below, hence, "buddies".
09   * At a high level, all that happens here is marking the table entry
10   * at the bottom level available, and propagating the changes upward
11   * as necessary, plus some accounting needed to play nicely with other
12   * parts of the VM system.
13   * At each level, we keep a list of pages, which are heads of continuous
14   * free pages of length of (1 << order) and marked with PageBuddy.
15   * Page's order is recorded in page_private(page) field.
16   * So when we are allocating or freeing one, we can derive the state of
    the
```

```
 * other.  That is, if we allocate a small block, and both were
 * free, the remainder of the region must be split into blocks.
 * If a block is freed, and its buddy is also free, then this
 * triggers coalescing into a block of larger size.
 *
 * -- nyc
 */
```

```
static inline void __free_one_page(struct page *page,
                unsigned long pfn,
                struct zone *zone, unsigned int order,
                int migratetype)
{
        unsigned long combined_pfn;
        unsigned long uninitialized_var(buddy_pfn);
        struct page *buddy;
        unsigned int max_order;

        max_order = min_t(unsigned int, MAX_ORDER, pageblock_order + 1);

        VM_BUG_ON(!zone_is_initialized(zone));
        VM_BUG_ON_PAGE(page->flags & PAGE_FLAGS_CHECK_AT_PREP, page);

        VM_BUG_ON(migratetype == -1);
        if (likely(!is_migrate_isolate(migratetype)))
                __mod_zone_freepage_state(zone, 1 << order, migratetype);

        VM_BUG_ON_PAGE(pfn & ((1 << order) - 1), page);
        VM_BUG_ON_PAGE(bad_range(zone, page), page);

continue_merging:
        while (order < max_order - 1) {
                buddy_pfn = __find_buddy_pfn(pfn, order);
                buddy = page + (buddy_pfn - pfn);

                if (!pfn_valid_within(buddy_pfn))
                        goto done_merging;
                if (!page_is_buddy(page, buddy, order))
                        goto done_merging;
                /*
                 * Our buddy is free or it is CONFIG_DEBUG_PAGEALLOC guard page,
                 * merge with it and move up one order.
                 */
                if (page_is_guard(buddy)) {
                        clear_page_guard(zone, buddy, order, migratetype);
                } else {
                        list_del(&buddy->lru);
                        zone->free_area[order].nr_free--;
                        rmv_page_order(buddy);
                }
                combined_pfn = buddy_pfn & pfn;
                page = page + (combined_pfn - pfn);
                pfn = combined_pfn;
                order++;
        }
        if (max_order < MAX_ORDER) {
                /* If we are here, it means order is >= pageblock_order.
                 * We want to prevent merge between freepages on isolate
                 * pageblock and normal pageblock. Without this, pageblock
                 * isolation could cause incorrect freepage or CMA accounting.
                 *
                 * We don't want to hit this code for the more frequent
                 * low-order merging.
56                       */
57                      if (unlikely(has_isolate_pageblock(zone))) {
58                              int buddy_mt;
59
60                              buddy_pfn = __find_buddy_pfn(pfn, order);
61                              buddy = page + (buddy_pfn - pfn);
62                              buddy_mt = get_pageblock_migratetype(buddy);
63
64                              if (migratetype != buddy_mt
65                                              && (is_migrate_isolate(migratetype) ||
66                                                      is_migrate_isolate(buddy_mt)))
67                                      goto done_merging;
68                      }
69                      max_order++;
70                      goto continue_merging;
71              }
```

The order page that has been used is retrieved by the migrate type of the buddy system.

- On line 11 of the code, use the smaller of the page block order or max order as the maximum order value to combine. It is used for loops, so it is an added value of 1.
- In line 17~18 of the code, when managing the number of free pages in the buddy system, the isolate type is not added to the number of free pages. Therefore, only when the isolate type is excluded, the number of free pages is reduced by the number of order pages to be recovered.
- In code lines 23~47, continue_merging: Label. It increments the order up to just before the max order, and if it finds a buddy page that is not a guard page to combine the page, it removes the entry for that order. If there are no more buddy pages to combine, exit the loop and go to the done_merging label.
- From lines 48~71 to the page block, it didn't matter if I combined different migrate types. However, since the isolate type is not added to the free page counter, in zones where isolation is in progress, buddy pages that will be combined for orders over a block of pages will not be combined if they do not use the same migrate type.
  - Consultation:
    - mm/page_alloc: fix incorrect isolation behavior by rechecking migrate type (https://github.com/torvalds/linux/commit/ad53f92eb416d81e469fa8ea57153e59455e7175)
    - mm/page_alloc: prevent merging between isolated and other pageblocks (https://github.com/torvalds/linux/commit/d9dddbf556674bf125ecd925b24e43a5cf2a568a)

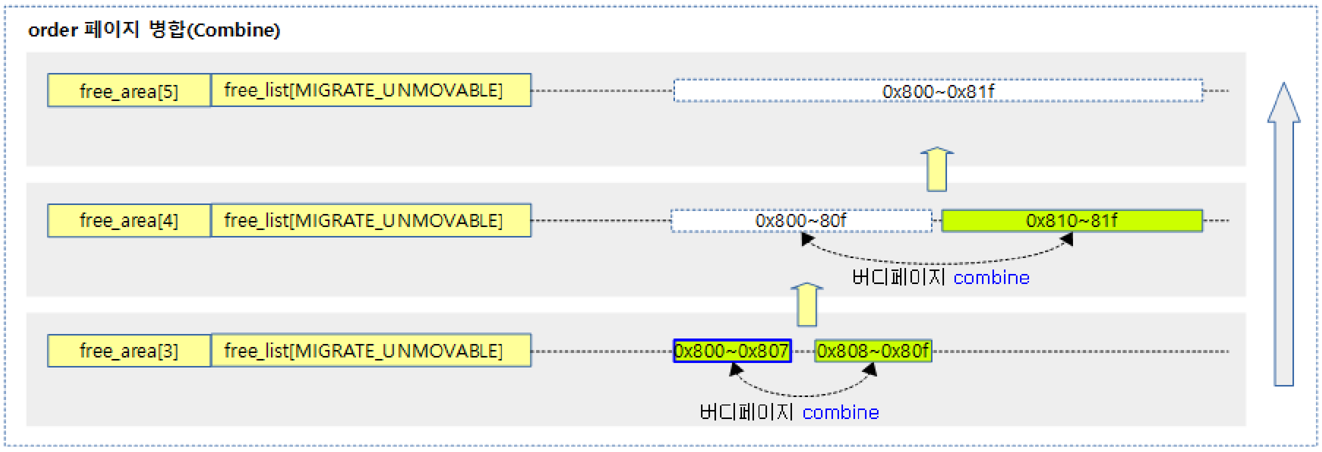
mm/page_alloc.c -2/2-

```
01 done_merging:
02         set_page_order(page, order);
03
04         /*
05          * If this is not the largest possible page, check if the buddy
06          * of the next-highest order is free. If it is, it's possible
07          * that pages are being freed that will coalesce soon. In case,
08          * that is happening, add the free page to the tail of the list
09          * so it's less likely to be used soon and more likely to be merged
10          * as a higher order page
```

```
11           */
12          if ((order < MAX_ORDER-2) && pfn_valid_within(buddy_pfn)) {
13                  struct page *higher_page, *higher_buddy;
14                  combined_pfn = buddy_pfn & pfn;
15                  higher_page = page + (combined_pfn - pfn);
16                  buddy_pfn = __find_buddy_pfn(combined_pfn, order + 1);
17                  higher_buddy = higher_page + (buddy_pfn - combined_pfn);
18                  if (pfn_valid_within(buddy_pfn) &&
19                      page_is_buddy(higher_page, higher_buddy, order + 1))
    {
20                          list_add_tail(&page->lru,
21                                  &zone->free_area[order].free_list[migrat
    etype]);
22                          goto out;
23                  }
24          }
25
26          list_add(&page->lru, &zone->free_area[order].free_list[migratety
    pe]);
27  out:
28          zone->free_area[order].nr_free++;
29  }
```

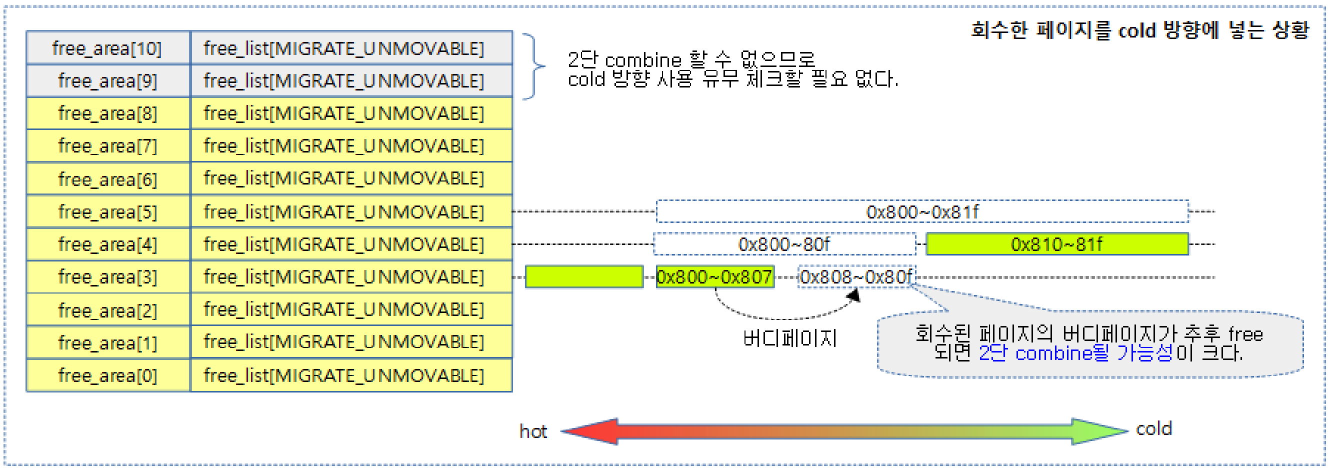
- In code lines 1~2, done_merging: Labels. I entered from a buddy page where there was no longer a buddy page to combine. Record the order on the free page.
- When there is a possibility of two-stage combine in code lines 12~24, add it in the cold direction. Check if there is a buddy page on the order page above step 2 of the order page to be retrieved. When the matching buddy page of the retrieved order page is later freed, it is more likely to be combined once more in the higher order. Therefore, in order to preserve as much as possible so that it is not immediately allocated, the page is added in the tail (cold) direction.
- In line 26~28 of the code, add the page in the head(hot) direction, and increase the number of entries in the order by 1.

The following figure shows how when an order 3 page is retrieved, a matching buddy page is found and merged into the parent order. It was registered in order 5 through two combines.



(http://jake.dothome.co.kr/wp-content/uploads/2016/05/free_one_page-4.png)

The figure below shows that when a page to be free is added to the corresponding order slot, if there is a buddy page from the parent slot, it is more likely to be double-combined. Therefore, in order to suppress the fragmentation of the free page as much as possible, it shows that the free page that has just been retrieved is added in the cold direction.


회수한 페이지를 cold 방향에 넣는 상황
free_area[10] free_list[MIGRATE_UNMOVABLE]
free_area[9] free_list[MIGRATE_UNMOVABLE]
free_area[8] free_list[MIGRATE_UNMOVABLE]
free_area[7] free_list[MIGRATE_UNMOVABLE]
free_area[6] free_list[MIGRATE_UNMOVABLE]
free_area[5] free_list[MIGRATE_UNMOVABLE]
free_area[4] free_list[MIGRATE_UNMOVABLE]
free_area[3] free_list[MIGRATE_UNMOVABLE]
free_area[2] free_list[MIGRATE_UNMOVABLE]
free_area[1] free_list[MIGRATE_UNMOVABLE]
free_area[0] free_list[MIGRATE_UNMOVABLE]
2단 combine 할 수 없으므로 cold 방향 사용 유무 체크할 필요 없다.
0x800~0x81f
0x800~80f
0x810~81f
0x800~0x807
0x808~0x80f
버디페이지
회수된 페이지의 버디페이지가 추후 free 되면 2단 combine될 가능성이 크다.
hot
cold

(http://jake.dothome.co.kr/wp-content/uploads/2016/05/free_one_page-3b.png)

has_isolate_pageblock()

include/linux/page-isolation.h

```
#ifdef CONFIG_MEMORY_ISOLATION
static inline bool has_isolate_pageblock(struct zone *zone)
{
        return zone->nr_isolate_pageblock;
}
#else
static inline bool has_isolate_pageblock(struct zone *zone)
{
        return false;
}
#endif
```

Returns whether an isolated page exists in the specified zone.

Check the order page which you have collected

free_pages_prepare()

mm/page_alloc.c

```
static __always_inline bool free_pages_prepare(struct page *page,
                                        unsigned int order, bool check_f
ree)
{
        int bad = 0;

        VM_BUG_ON_PAGE(PageTail(page), page);

        trace_mm_page_free(page, order);

        /*
         * Check tail pages before head page information is cleared to
         * avoid checking PageCompound for order-0 pages.
         */
        if (unlikely(order)) {
                bool compound = PageCompound(page);
                int i;
17
18                 VM_BUG_ON_PAGE(compound && compound_order(page) != order, page);
19
20                 if (compound)
21                         ClearPageDoubleMap(page);
22                 for (i = 1; i < (1 << order); i++) {
23                         if (compound)
24                                 bad += free_tail_pages_check(page, page + i);
25                         if (unlikely(free_pages_check(page + i))) {
26                                 bad++;
27                                 continue;
28                         }
29                         (page + i)->flags &= ~PAGE_FLAGS_CHECK_AT_PREP;
30                 }
31         }
32         if (PageMappingFlags(page))
33                 page->mapping = NULL;
34         if (memcg_kmem_enabled() && PageKmemcg(page))
35                 memcg_kmem_uncharge(page, order);
36         if (check_free)
37                 bad += free_pages_check(page);
38         if (bad)
39                 return false;
40
41         page_cpupid_reset_last(page);
42         page->flags &= ~PAGE_FLAGS_CHECK_AT_PREP;
43         reset_page_owner(page, order);
44
45         if (!PageHighMem(page)) {
46                 debug_check_no_locks_freed(page_address(page),
47                                            PAGE_SIZE << order);
48                 debug_check_no_obj_freed(page_address(page),
49                                            PAGE_SIZE << order);
50         }
51         arch_free_page(page, order);
52         kernel_poison_pages(page, 1 << order, 0);
53         kernel_map_pages(page, 1 << order, 0);
54         kasan_free_nondeferred_pages(page, order);
55
56         return true;
57 }
```

Before retrieving the order page to the buddy system, check the flags on each page to see if there are any bad requirements. true=no abnormality, false=bad page

- In line 14~21 of the code, if it is a compound page, clear the PG_double_map of the second page.
- From the second page of the code line 22~30 to the end of the order page, check whether there is a bad page. And remove the PAGE_FLAGS_CHECK_AT_PREP flag.
- In lines 32~33 of code, initialize the page mapping to null.
- In line 34~35 of the code, if it is a kmemcg page, uncharge it as much as the order page.
- If the @check_free is set in line 36~37 of the code
- On lines 38~39 of the code, it returns false if the result is a bad page.
- In lines 41~43 of the code, reset the cpupid information on the page, remove the PAGE_FLAGS_CHECK_AT_PREP flag, and reset the page owner.
- On line 45~50 of the code, if it is a highmem page, perform a debug check.
- In line 51 of the code, if the architecture supports checking for free pages, do it.
- If you use the poison debug function in line 52 of code, poison the free page.

- In line 53 of the code, when using the pagealloc debug function, it checks whether the memory is valid or not.
- This is the handling of the case when using the kasan debug function in line 54 of code.
- On line 56 of the code, return true because there is no abnormality on the page.

Check out the Buddy page

__find_buddy_pfn()

mm/internal.h

```
/*
 * Locate the struct page for both the matching buddy in our
 * pair (buddy1) and the combined O(n+1) page they form (page).
 *
 * 1) Any buddy B1 will have an order O twin B2 which satisfies
 * the following equation:
 *     B2 = B1 ^ (1 << O)
 * For example, if the starting buddy (buddy2) is #8 its order
 * 1 buddy is #10:
 *     B2 = 8 ^ (1 << 1) = 8 ^ 2 = 10
 *
 * 2) Any buddy B will have an order O+1 parent P which
 * satisfies the following equation:
 *     P = B & ~(1 << O)
 *
 * Assumption: *_mem_map is contiguous at least up to MAX_ORDER
 */
```

```
static inline unsigned long
__find_buddy_pfn(unsigned long page_pfn, unsigned int order)
{
        return page_pfn ^ (1 << order);
}
```

Returns the pfn of the buddy page that matches the pfn of the requested order.

- 예) pfn=0x1000, order=3
  - =0x1008
- e.g. page_idx=0x1008, order=3
  - =0x1000

page_is_buddy()

mm/page_alloc.c

```
/*
 * This function checks whether a page is free && is the buddy
 * we can coalesce a page and its buddy if
 * (a) the buddy is not in a hole (check before calling!) &&
 * (b) the buddy is in the buddy system &&
 * (c) a page and its buddy have the same order &&
 * (d) a page and its buddy are in the same zone.
 *
 * For recording whether a page is in the buddy system, we set PageBudd
y.
 * Setting, clearing, and testing PageBuddy is serialized by zone->lock.
 *
 * For recording page's order, we use page_private(page).
 */
```

```
01 static inline int page_is_buddy(struct page *page, struct page *buddy,
02                                                          unsigned int order)
03 {
04         if (page_is_guard(buddy) && page_order(buddy) == order) {
05                 if (page_zone_id(page) != page_zone_id(buddy))
06                         return 0;
07 
08                 VM_BUG_ON_PAGE(page_count(buddy) != 0, buddy);
09 
10                 return 1;
11         }
12 
13         if (PageBuddy(buddy) && page_order(buddy) == order) {
14                 /*
15                  * zone check is done late to avoid uselessly
16                  * calculating zone/node ids for pages that could
17                  * never merge.
18                  */
19                 if (page_zone_id(page) != page_zone_id(buddy))
20                         return 0;
21 
22                 VM_BUG_ON_PAGE(page_count(buddy) != 0, buddy);
23 
24                 return 1;
25         }
26         return 0;
27 }
```

Returns 1 if the @page and @buddy pages in the same zone are paired, otherwise 0.

- In code lines 4~11, if page @buddy is being used as the guard page, and order is the same, return 1. However, if the @page and the @buddy page are not in the same zone, it returns 0.
- In lines 13~25 of the code, if the @buddy page is set to PG_buddy flag, and order is the same, it returns 1. However, if the @page and the @buddy page are not in the same zone, it returns 0.
- In line 26 of the code, @page and @buddy are unpaired, so return 0.

Delete an order value

rmv_page_order()

mm/page_alloc.c

```
static inline void rmv_page_order(struct page *page)
{
        __ClearPageBuddy(page);
        set_page_private(page, 0);
}
```

Clear the PG_buddy flag on the page and assign 0 to the private of the page that represents the order bit.

include/linux/mm.h

```
#define set_page_private(page, v)       ((page)->private = (v))
```

consultation

- Zoned Allocator -1- (Physics Page Assignment - Fastpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-fastpath) | Qc
- Zoned Allocator -2- (Physics Page Assignment - Slowpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-slowpath) | Qc
- Zoned Allocator -3- (Buddy Page Allocation) (http://jake.dothome.co.kr/buddy-alloc) | Qc
- Zoned Allocator -4- (Buddy Page Terminated) (http://jake.dothome.co.kr/buddy-free/) | Sentence C – Current post
- Zoned Allocator -5- (Per-CPU Page Frame Cache) (http://jake.dothome.co.kr/per-cpu-page-frame-cache) | 문c
- Zoned Allocator -6- (Watermark) (http://jake.dothome.co.kr/zonned-allocator-watermark) | 문c
- Zoned Allocator -7- (Direct Compact) (http://jake.dothome.co.kr/zonned-allocator-compaction) | 문c
- Zoned Allocator -8- (Direct Compact-Isolation) (http://jake.dothome.co.kr/zonned-allocator-isolation) | 문c
- Zoned Allocator -9- (Direct Compact-Migration) (http://jake.dothome.co.kr/zonned-allocator-migration) | 문c
- Zoned Allocator -10- (LRU & pagevec) (http://jake.dothome.co.kr/lru-lists-pagevecs) | 문c
- Zoned Allocator -11- (Direct Reclaim) (http://jake.dothome.co.kr/zonned-allocator-reclaim) | 문c
- Zoned Allocator -12- (Direct Reclaim-Shrink-1) (http://jake.dothome.co.kr/zonned-allocator-shrink-1) | 문c
- Zoned Allocator -13- (Direct Reclaim-Shrink-2) (http://jake.dothome.co.kr/zonned-allocator-shrink-2) | 문c
- Zoned Allocator -14- (Kswapd) (http://jake.dothome.co.kr/zonned-allocator-kswapd) | 문c

- page_alloc_init() (http://jake.dothome.co.kr/page_alloc_init) | Qc

LEAVE A COMMENT

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

Comments

name *

email *

Website

WRITE A COMMENT

mem_init() (http://jake.dothome.co.kr/mem_init/)

Zonned Allocator -3- (Buddy Page Allocation) (http://jake.dothome.co.kr/buddy-alloc/)