kmalloc

2016-06-17 (http://jake.dothome.co.kr/kmalloc/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

kmalloc allocator

The kernel is primarily allocated and used a small contiguous physical address space. Since it is a physically contiguous space, it can also be used as a DMA buffer.

feature

It is used when the kernel requires the allocation of contiguous memory with physical and virtual addresses. The allocation of kernel memory using kmalloc has the following characteristics:

- It uses pre-mapped kernel memory, so it can be used right out of the box without any mapping.
- Use a pre-prepared slab cache for kmalloc.
  - The advantage of using a slap cache is that it provides physically contiguous memory, so it can be used for DMA.
  - The disadvantage is that it does not use a single page, but uses pages per order, so it is sensitive to fragment management.

KMALLOC type

It is a form of pre-creating a slab cache by specifying the size that is frequently used in the kernel, and three types of each size are supported as follows.

- kmalloc-<size>
  - It is a commonly used type and cannot be recalled from memory.
- kmalloc-rcl-<size>
  - It is a type that is used for the type that can be recovered by providing a shrinker.
  - Use this type when using __GFP_RECLAIMABLE flags.
  - 참고: mm, slab/slub: introduce kmalloc-reclaimable caches (v4.20-rc1, 2018) (https://github.com/torvalds/linux/commit/1291523f2c1d631fea34102fd241fb54a4e8f7a0#diff-f966790d3e3bbaa3d6b7f7d8fb77abbb)
- dma-kmalloc-<size>
  - This type is only used in systems that require DMA zones with address restrictions.
  - Use __GFP_DMA flag to use ZONE_DMA area for the allocation of kernel memory for DMA.
    - In systems that do not require a separate DMA zone, a generic type of KMALLOC is used to allocate memory for DMA.

kmalloc size

- Slab caches are prepared in advance from the L1 cache line size of the architecture to the multiple size of 2.
- In the case of the Slub cache, it is prepared in advance up to 2 pages in size. If you request more than this amount of memory, it will be served using a page allocator that uses the buddy system, not the slab cache for kmalloc.
  - Depending on the architecture, some systems offer sizes 96 or 192.

kmalloc using the Slub cache supports the following sizes: (In the case of ARM64, use the items marked in orange.)

- kmalloc-8
- kmalloc-16
- kmalloc-32
- kmalloc-64
- kmalloc-96
- kmalloc-128
- kmalloc-192
- kmalloc-256
- kmalloc-512
- kmalloc-1k
- kmalloc-2k
- kmalloc-4k
- kmalloc-8k

The following shows the different types of kmalloc. (Utility that compiles and uses kernel/tools/vm/slabinfo.c)

```
$ slabinfo kmalloc
Name                   Objects Objsize           Space Slabs/Part/Cpu  O/S O %Fr %E
f Flg
kmalloc-128               8081     128            1.0M      246/61/14   32 0  23   9
7
kmalloc-256               1232     256          315.3K        65/0/12   16 0   0  10
0
kmalloc-512                707     512          483.3K        50/28/9   16 1  47   7
4
kmalloc-1k                 590    1024          638.9K         35/6/4   16 2  15   9
4
kmalloc-2k                 176    2048          360.4K          9/0/2   16 3   0  10
0
kmalloc-4k                  53    4096          229.3K          4/1/3    8 3  14   9
4
kmalloc-8k                  32    8192          262.1K          7/0/1    4 3   0  10
0
kmalloc-rcl-128            448     128           57.3K         13/0/1   32 0   0  10
0 a
```

API

The main APIs for allocation and release are as follows:

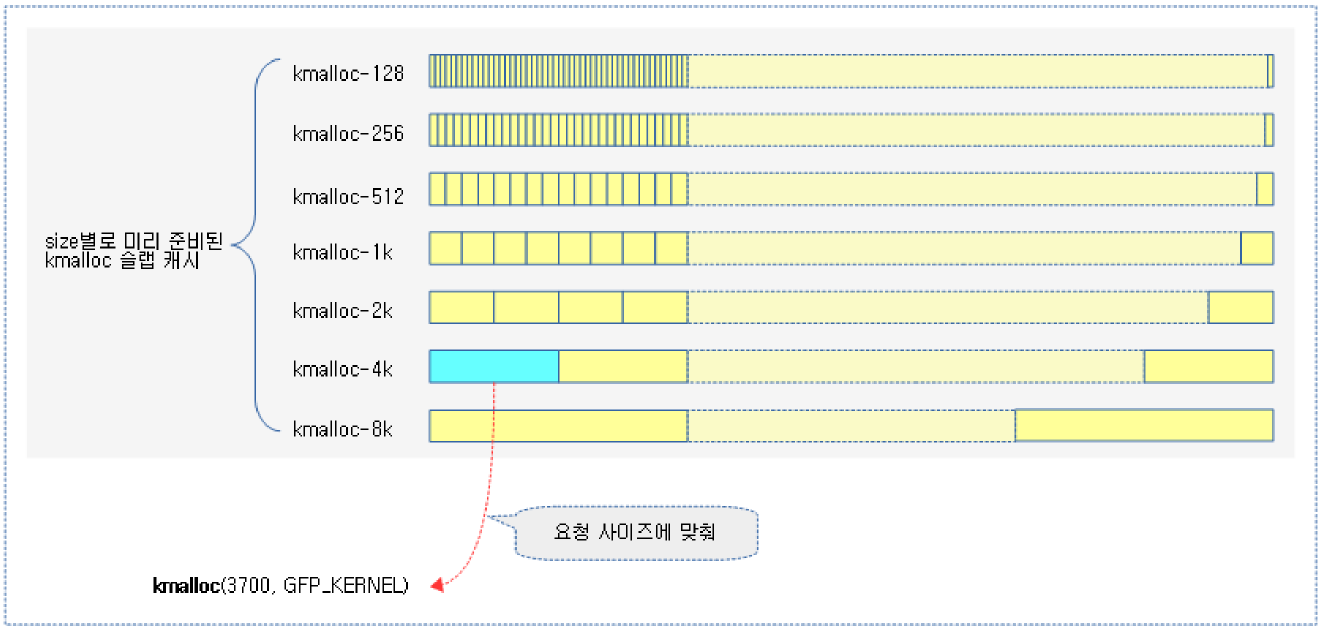
- kmalloc()
- kfree()

GFP flags used by kmalloc()

- GFP_KERNEL
  - kernel memory allocation and can be put to sleep.
- GFP_ATOMIC
  - It is used when it should not be sleeped, and it is used as an interrupt handler, etc.
- GFP_HIGHUSER
  - First, allocate the page to the user's highmem (high memory).
- GFP_NOWAIT
  - If you run out of memory while allocating memory, you can wake up kswapd and allow it to reclaim.
- __GFP_HIGH
  - It is used when requesting that it be processed in a high priority manner.
- __GFP_NOFAIL
  - It is used to request that a memory allocation request be processed until it succeeds, without allowing failures.
- __GFP_NORETRY
  - Do not retry on failure for memory allocation requests.
- __GFP_NOWARN
  - Don't handle any warnings when memory allocation fails.
- __GFP_RETRY_MAYFAIL

- If the memory allocation fails the first time, it can be retried, but it can fail.

The following illustration shows the kmalloc() function allocating an object using one of the pre-prepared kmalloc slab caches.



(http://jake.dothome.co.kr/wp-content/uploads/2016/06/kmalloc-2.png)

Initialize Kmalloc

size_index table

mm/slab_common.c"

```
/*
 * Conversion table for small slabs sizes / 8 to the index in the
 * kmalloc array. This is necessary for slabs < 192 since we have non po
wer
 * of two cache sizes there. The size of larger slabs can be determined
using
 * fls.
 */
```

```
static s8 size_index[24] = {
        3,      /* 8 */
        4,      /* 16 */
        5,      /* 24 */
        5,      /* 32 */
        6,      /* 40 */
        6,      /* 48 */
        6,      /* 56 */
        6,      /* 64 */
        1,      /* 72 */
        1,      /* 80 */
        1,      /* 88 */
        1,      /* 96 */
        7,      /* 104 */
        7,      /* 112 */
        7,      /* 120 */
        7,      /* 128 */
        2,      /* 136 */
        2,      /* 144 */
        2,      /* 152 */
        2,      /* 160 */
        2,      /* 168 */
        2,      /* 176 */
        2,      /* 184 */
        2       /* 192 */
};
```

This table is for selecting the kmalloc slab cache to use when the slab object size is 192 bytes or less. The index value in parentheses is the size divided by 8, and the entry value and the resulting kmalloc slab cache name are as follows:

- KMALLOC-1 using size 72 -> 96~96
  - If the cache line supported by the architecture is 64 or higher, it will abandon the creation of kmalloc-96 and replace it with kmalloc-128
- KMALLOC-2 using size 136 -> 192~192
  - If the cache line supported by the architecture is greater than 128, it will abandon the creation of kmalloc-192 and replace it with kmalloc-256.
- 3 -> kmalloc-8
- 4 -> kmalloc-16
- 5 -> kmalloc-32
- 6 -> kmalloc-64
- 7 -> kmalloc-128

kmalloc_info[] table

mm/slab_common.c

```
/*
 * kmalloc_info[] is to make slub_debug=,kmalloc-xx option work at boot time.
 * kmalloc_index() supports up to 2^26=64MB, so the final entry of the table is
 * kmalloc-67108864.
 */
```

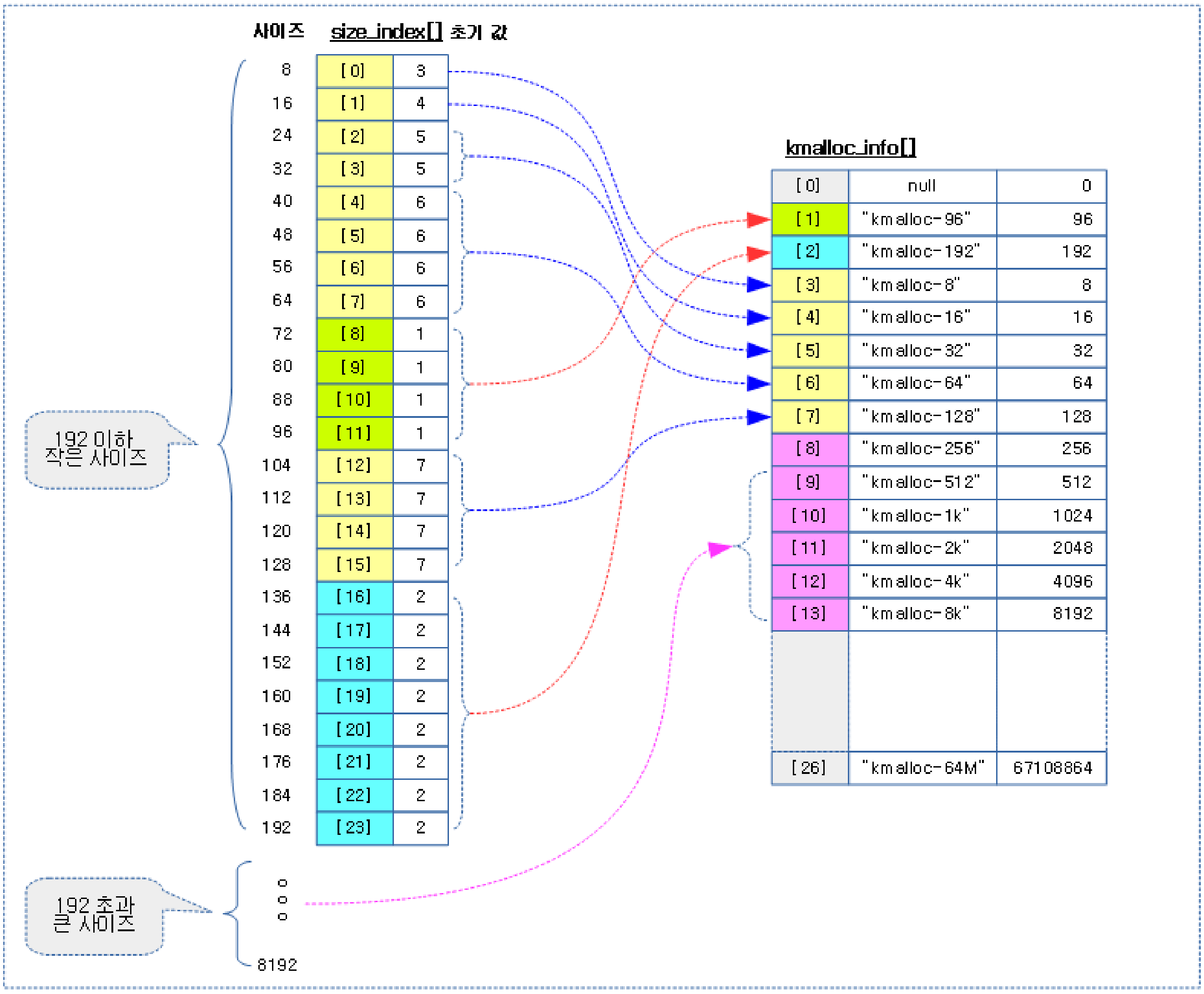
```
const struct kmalloc_info_struct kmalloc_info[] __initconst = {
        {NULL,                      0},         {"kmalloc-96",             96},
        {"kmalloc-192",           192},         {"kmalloc-8",               8},
        {"kmalloc-16",             16},         {"kmalloc-32",             32},
        {"kmalloc-64",             64},         {"kmalloc-128",           128},
        {"kmalloc-256",           256},         {"kmalloc-512",           512},
        {"kmalloc-1k",           1024},         {"kmalloc-2k",           2048},
        {"kmalloc-4k",           4096},         {"kmalloc-8k",           8192},
        {"kmalloc-16k",         16384},         {"kmalloc-32k",         32768},
        {"kmalloc-64k",         65536},         {"kmalloc-128k",       131072},
        {"kmalloc-256k",       262144},         {"kmalloc-512k",       524288},
        {"kmalloc-1M",       1048576},        {"kmalloc-2M",       2097152},
        {"kmalloc-4M",       4194304},        {"kmalloc-8M",       8388608},
        {"kmalloc-16M",     16777216},        {"kmalloc-32M",     33554432},
        {"kmalloc-64M",     67108864}
};
```

kmalloc_info table initially consists of information about the name and size of the kmalloc cache up to index 26.

- KMALLOC-1 96 will not operate if the cache line of the architecture is 64 or higher.
  - ARM64 does not operate.
- KMALLOC-2 192 will not operate if the cache line of the architecture is 128 or higher.
  - ARM64 does not operate.
- In the case of kmalloc-3 ~ kmalloc-6 8~64, kmalloc smaller than the size of the cache line will not be operated.
  - ARM64 is not available at all.
- The SLUB cache only supports twice the size of the page.
  - e.g. ARM64 and 4K pages support kmalloc-128 ~ kmalloc-8k.

The figure below shows how the appropriate kmalloc slab cache is selected via the size_index[] table according to size.

- The value of the size_index[] table is the initial state before it is changed.



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

Initialize the kmalloc cache index table

setup_kmalloc_cache_index_table()

mm/slab_common.c

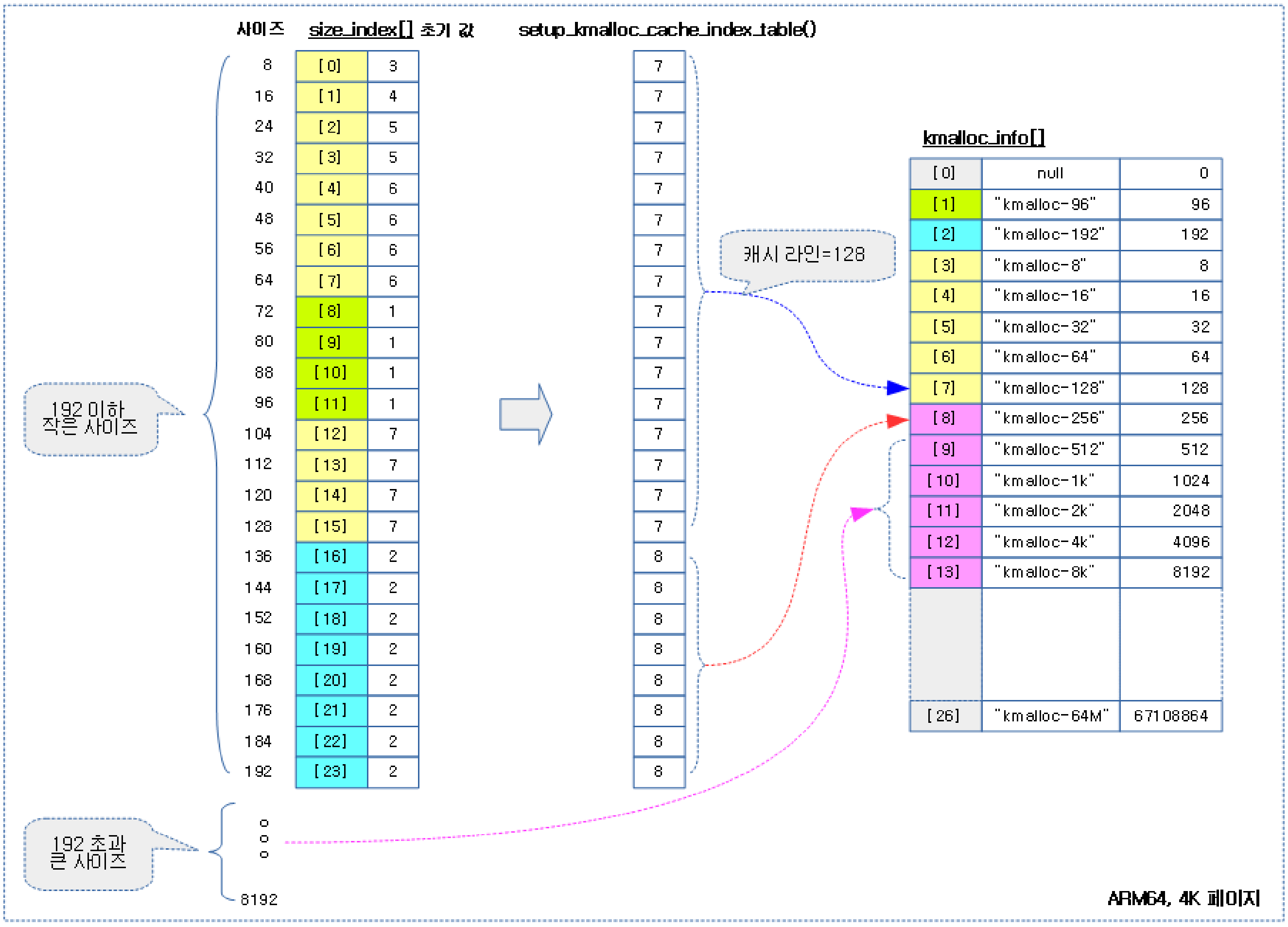
```
/*
 * Patch up the size_index table if we have strange large alignment
 * requirements for the kmalloc array. This is only the case for
 * MIPS it seems. The standard arches will not generate any code here.
 *
 * Largest permitted alignment is 256 bytes due to the way we
 * handle the index determination for the smaller caches.
 *
 * Make sure that nothing crazy happens if someone starts tinkering
 * around with ARCH_KMALLOC_MINALIGN
 */
```

```
void __init setup_kmalloc_cache_index_table(void)
{
        unsigned int i;

        BUILD_BUG_ON(KMALLOC_MIN_SIZE > 256 ||
                (KMALLOC_MIN_SIZE & (KMALLOC_MIN_SIZE - 1)));

        for (i = 8; i < KMALLOC_MIN_SIZE; i += 8) {
                unsigned int elem = size_index_elem(i);
10
11                      if (elem >= ARRAY_SIZE(size_index))
12                              break;
13                      size_index[elem] = KMALLOC_SHIFT_LOW;
14              }
15
16              if (KMALLOC_MIN_SIZE >= 64) {
17                      /*
18                       * The 96 byte size cache is not used if the alignment
19                       * is 64 byte.
20                       */
21                      for (i = 64 + 8; i <= 96; i += 8)
22                              size_index[size_index_elem(i)] = 7;
23
24              }
25
26              if (KMALLOC_MIN_SIZE >= 128) {
27                      /*
28                       * The 192 byte sized cache is not used if the alignment
29                       * is 128 byte. Redirect kmalloc to use the 256 byte cache
30                       * instead.
31                       */
32                      for (i = 128 + 8; i <= 192; i += 8)
33                              size_index[size_index_elem(i)] = 8;
34              }
35      }
```

Initialize the size_index table for the kmalloc cache

- From code lines 8~14 to KMALLOC_MIN_SIZE size, the lowest size cache is integrated and operated.
  - For ARM32 rpi2, the lowest size is KMALLOC-64.
  - In the case of ARM32 rpi3, the lowest size is kmalloc-128.
  - In the case of ARM64 rpi4, the lowest size is kmalloc-128.
- If the KMALLOC_MIN_SIZE size from code lines 16~24 is 64 or more, the size 72~96 should use kmalloc-96 instead of kmalloc-128.
  - In the case of ARM32 rpi2, KMALLOC-72 is used for sizes from 96~128
  - In the case of ARM32 rpi3, KMALLOC-72 is used for sizes from 96~128
  - For ARM64 rpi4, KMALLOC-72 is used for sizes from 96~128.
- If the KMALLOC_MIN_SIZE size is 26 or more in line 34~128, the size 140~192 will use kmalloc-192 instead of kmalloc-256.
  - In the case of ARM32 rpi2, KMALLOC-72 is supported for sizes from 96~192.
  - In the case of ARM32 rpi3, KMALLOC-72 is used for sizes from 96~256
  - For ARM64 rpi4, KMALLOC-72 is used for sizes from 96~256.

The figure below shows the selection of the appropriate kmalloc slab cache via a size_index[] table tuned by the setup_kmalloc_cache_index_table() function



사이즈	size_index[]	초기 값	setup_kmalloc_cache_index_table()
8	[0]	3	7
16	[1]	4	7
24	[2]	5	7
32	[3]	5	7
40	[4]	6	7
48	[5]	6	7
56	[6]	6	7
64	[7]	6	7
72	[8]	1	7
80	[9]	1	7
88	[10]	1	7
96	[11]	1	7
104	[12]	7	7
112	[13]	7	7
120	[14]	7	7
128	[15]	7	7
136	[16]	2	8
144	[17]	2	8
152	[18]	2	8
160	[19]	2	8
168	[20]	2	8
176	[21]	2	8
184	[22]	2	8
192	[23]	2	8

kmalloc_info[]

[0]	null	0
[1]	"kmalloc-96"	96
[2]	"kmalloc-192"	192
[3]	"kmalloc-8"	8
[4]	"kmalloc-16"	16
[5]	"kmalloc-32"	32
[6]	"kmalloc-64"	64
[7]	"kmalloc-128"	128
[8]	"kmalloc-256"	256
[9]	"kmalloc-512"	512
[10]	"kmalloc-1k"	1024
[11]	"kmalloc-2k"	2048
[12]	"kmalloc-4k"	4096
[13]	"kmalloc-8k"	8192
[26]	"kmalloc-64M"	67108864

(http://jake.dothome.co.kr/wp-content/uploads/2016/06/size_index-2a.png)

Initialize the kmalloc cache

create_kmalloc_caches() – (slab, slub)

mm/slab_common.c

```
/*
 * Create the kmalloc array. Some of the regular kmalloc arrays
 * may already have been created because they were needed to
 * enable allocations for slab creation.
 */
```

```
void __init create_kmalloc_caches(slab_flags_t flags)
{
        int i, type;

        for (type = KMALLOC_NORMAL; type <= KMALLOC_RECLAIM; type++) {
                for (i = KMALLOC_SHIFT_LOW; i <= KMALLOC_SHIFT_HIGH; i++) {
                        if (!kmalloc_caches[type][i])
                                new_kmalloc_cache(i, type, flags);

                        /*
                         * Caches that are not of the two-to-the-power-of size.
                         * These have to be created immediately after the
                         * earlier power of two caches
                         */
                        if (KMALLOC_MIN_SIZE <= 32 && i == 6 &&
                                        !kmalloc_caches[type][1])
17                                 new_kmalloc_cache(1, type, flags);
18                         if (KMALLOC_MIN_SIZE <= 64 && i == 7 &&
19                                         !kmalloc_caches[type][2])
20                                 new_kmalloc_cache(2, type, flags);
21                 }
22         }
23 
24         /* Kmalloc array is now usable */
25         slab_state = UP;
26 
27 #ifdef CONFIG_ZONE_DMA
28         for (i = 0; i <= KMALLOC_SHIFT_HIGH; i++) {
29                 struct kmem_cache *s = kmalloc_caches[KMALLOC_NORMAL][i];
30 
31                 if (s) {
32                         unsigned int size = kmalloc_size(i);
33                         const char *n = kmalloc_cache_name("dma-kmalloc", size);
34 
35                         BUG_ON(!n);
36                         kmalloc_caches[KMALLOC_DMA][i] = create_kmalloc_cache(
37                                 n, size, SLAB_CACHE_DMA | flags, 0, 0);
38                 }
39         }
40 #endif
41 }
```

Generate kmalloc slab caches in advance by type and size.

- In line 5~22 of code, it generates a kmalloc cache with two types, normal type and reclaim type, traversing the size supported by the slab.
- In line 25 of code, the kmalloc array cache can now be used.
- In code lines 28~39, it traverses against the size supported by the slab and generates a kmalloc cache of type DMA.

new_kmalloc_cache()

mm/slab_common.c

```
static void __init
new_kmalloc_cache(int idx, int type, slab_flags_t flags)
{
        const char *name;

        if (type == KMALLOC_RECLAIM) {
                flags |= SLAB_RECLAIM_ACCOUNT;
                name = kmalloc_cache_name("kmalloc-rcl",
                                                kmalloc_info[idx].size);
                BUG_ON(!name);
        } else {
                name = kmalloc_info[idx].name;
        }

        kmalloc_caches[type][idx] = create_kmalloc_cache(name,
                                        kmalloc_info[idx].size, flags, 0,
                                        kmalloc_info[idx].size);
}
```

kmalloc_infi Using the [@idx] information, create a kmalloc slab cache for @type.

- 예) idx=8, type=KMALLOC_RECLAIM
    - kmalloc-rcl-256
- 예) idx=10, type=KMALLOC_NORMAL
    - kmalloc-1k

size_index_elem()

mm/slab_common.c

```
static inline int size_index_elem(size_t bytes)
{
        return (bytes - 1) / 8;
}
```

Returns the value of the size_index[] array element index, which is used in 8-byte increments for argument bytes.

- e.g. 0~7=0, 8~15=1, 16~23=2, …

kmalloc_size()

include/linux/slab.h

```
/*
 * Determine size used for the nth kmalloc cache.
 * return size or 0 if a kmalloc cache for that
 * size does not exist
 */
```

```
static __always_inline unsigned int kmalloc_size(unsigned int n)
{
#ifndef CONFIG_SLOB
        if (n > 2)
                return 1U << n;

        if (n == 1 && KMALLOC_MIN_SIZE <= 32)
                return 96;

        if (n == 2 && KMALLOC_MIN_SIZE <= 64)
                return 192;
#endif
        return 0;
}
```

Returns the size corresponding to the value of the argument n.

- 예) rpi2: 0->0, 1->0, 2->192, 3->8, 4->16, 5->32, 6->64, 7->128, …

create_kmalloc_cache()

mm/slab_common.c

```
struct kmem_cache *__init create_kmalloc_cache(const char *name,
                unsigned int size, slab_flags_t flags,
                unsigned int useroffset, unsigned int usersize)
{
        struct kmem_cache *s = kmem_cache_zalloc(kmem_cache, GFP_NOWAI
T);
06
07          if (!s)
08                  panic("Out of memory when creating slab %s\n", name);
09
10          create_boot_cache(s, name, size, flags, useroffset, usersize);
11          list_add(&s->list, &slab_caches);
12          memcg_link_cache(s);
13          s->refcount = 1;
14          return s;
15  }
```

Create a slab cache with the @name, @size, and @flags given as arguments.

- In line 5~8 of the code, the kmem_cache gets a slab object from the cache.
- In line 10 of the code, the per cpu cache used by the kmem_cache cache is allocated as the per-cpu data type, and the structure kmem_cache_node used by the per node is allocated a slub object from the kmem_cache_node cache and registered in a partial list
- In line 11~14 of the code, add the created cache to the global slab_caches list and add it to memcg, then substitute 1 for the reference counter and return the created slab cache.

mm/slab_common.c

```
struct kmem_cache *
kmalloc_caches[NR_KMALLOC_TYPES][KMALLOC_SHIFT_HIGH + 1] __ro_after_init;
EXPORT_SYMBOL(kmalloc_caches);
```

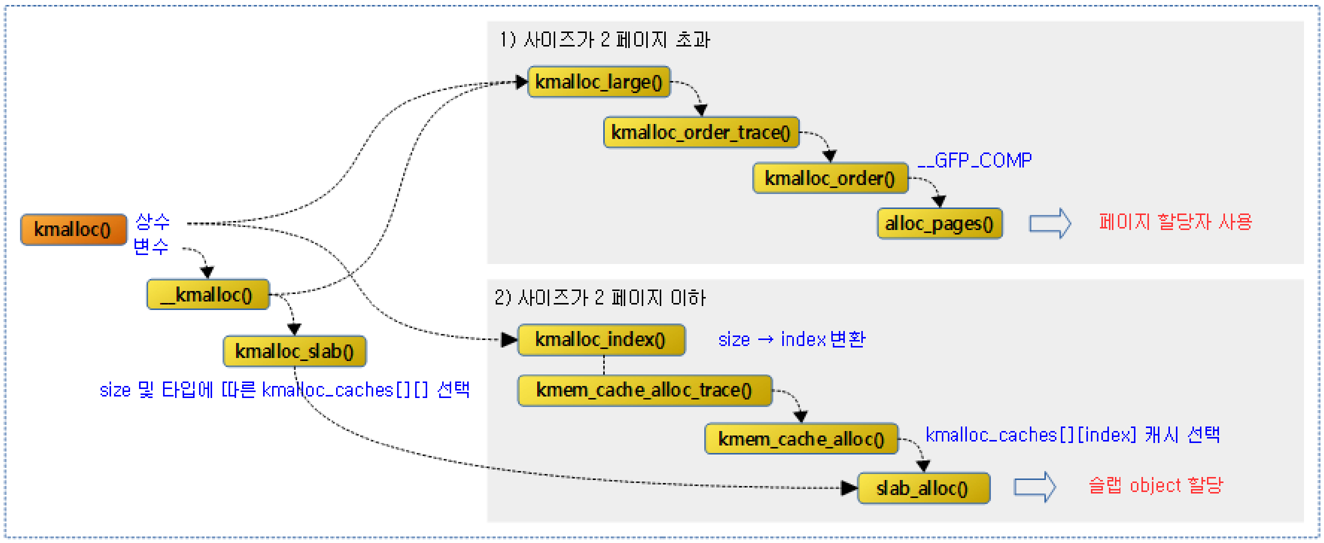
kmalloc slab cache list.

Kmalloc Allocator

If the kernel requires the allocation of memory with contiguous physical and virtual addresses, it should be allocated using the following memory allocators depending on the request size.

- SLUB Allocator
  - For requests of 2 pages or less, a slab object is allocated using the pre-created KMALLOC slab cache in multiples of 2.
- Page allocator
  - For requests exceeding 2 pages, use a page allocator that uses a buddy system to allocate pages in order units.

The following figure shows the functions that the kmalloc() function is called according to its size. If the allocation request size is large, it is requested from the page allocator, and if it is small, it is requested to the pre-prepared kmalloc slub cache.



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

kmalloc()

include/mm/slab.h

```
/**
 * kmalloc - allocate memory
 * @size: how many bytes of memory are required.
 * @flags: the type of memory to allocate.
 *
 * kmalloc is the normal method of allocating memory
 * for objects smaller than page size in the kernel.
 *
 * The @flags argument may be one of the GFP flags defined at
 * include/linux/gfp.h and described at
 * :ref:`Documentation/core-api/mm-api.rst <mm-api-gfp-flags>`
 *
 * The recommended usage of the @flags is described at
 * :ref:`Documentation/core-api/memory-allocation.rst <memory-allocation>`
 *
 * Below is a brief outline of the most useful GFP flags
 *
 * %GFP_KERNEL
 *      Allocate normal kernel ram. May sleep.
 *
 * %GFP_NOWAIT
 *      Allocation will not sleep.
 *
 * %GFP_ATOMIC
 *      Allocation will not sleep.  May use emergency pools.
 *
 * %GFP_HIGHUSER
 *      Allocate memory from high memory on behalf of user.
 *
 * Also it is possible to set different flags by OR'ing
 * in one or more of the following additional @flags:
 *
 * %__GFP_HIGH
 *      This allocation has high priority and may use emergency pools.
 *
 * %__GFP_NOFAIL
 *      Indicate that this allocation is in no way allowed to fail
 *      (think twice before using).
 *
 * %__GFP_NORETRY
 *      If memory is not immediately available,
 *      then give up at once.
 *
 * %__GFP_NOWARN
 *      If allocation fails, don't issue any warnings.
 *
 * %__GFP_RETRY_MAYFAIL
 *      Try really hard to succeed the allocation but fail
 *      eventually.
 */
```

```
01 static __always_inline void *kmalloc(size_t size, gfp_t flags)
02 {
03         if (__builtin_constant_p(size)) {
04 #ifndef CONFIG_SLOB
05                 unsigned int index;
06 #endif
07                 if (size > KMALLOC_MAX_CACHE_SIZE)
08                         return kmalloc_large(size, flags);
09 #ifndef CONFIG_SLOB
10                 index = kmalloc_index(size);
11 
12                 if (!index)
13                         return ZERO_SIZE_PTR;
14 
15                 return kmem_cache_alloc_trace(
16                                 kmalloc_caches[kmalloc_type(flags)][index],
17                                 flags, size);
18 #endif
19         }
20         return __kmalloc(size, flags);
21 }
```

For memory allocation requests from the kernel, if the allocation request size exceeds 2 pages (based on slub), the request is made to the page allocator using the buddy system, otherwise the request is made using the kmalloc slab cache.

- In line 3~8 of the code, if the request size is constant and exceeds KMALLOC_MAX_CACHE_SIZE (2 pages per slub), the request is made to the page allocator using the direct buddy system.
  - KMALLOC_MAX_CACHE_SIZE
    - If you use a slub 2 page size
- In line 10~17 of code, if the request size is constant, the slab object is allocated by selecting the kmalloc slab cache according to the size and type.
  - The kmalloc_index() function calculates the index value by size.
- If you used a variable for size on line 20 of code, call the __kmalloc() function.

kmalloc_large()

include/linux/slab.h

```
static __always_inline void *kmalloc_large(size_t size, gfp_t flags)
{
        unsigned int order = get_order(size);
        return kmalloc_order_trace(size, flags, order);
}
```

After determining the order required for size, the page is assigned by the page allocator who uses the buddy system.

kmalloc_order_trace()

mm/slab_common.c

```
#ifdef CONFIG_TRACING
void *kmalloc_order_trace(size_t size, gfp_t flags, unsigned int order)
{
        void *ret = kmalloc_order(size, flags, order);
        trace_kmalloc(_RET_IP_, ret, size, PAGE_SIZE << order, flags);
        return ret;
}
EXPORT_SYMBOL(kmalloc_order_trace);
#endif
```

You will be assigned as many pages as you order from the page allocator who uses the buddy system.

kmalloc_order()

mm/slab_common.c

```
/*
 * To avoid unnecessary overhead, we pass through large allocation reque
sts
 * directly to the page allocator. We use __GFP_COMP, because we will ne
ed to
 * know the allocation order to free the pages properly in kfree.
 */
```

```
void *kmalloc_order(size_t size, gfp_t flags, unsigned int order)
{
        void *ret;
        struct page *page;

        flags |= __GFP_COMP;
        page = alloc_pages(flags, order);
        ret = page ? page_address(page) : NULL;
        ret = kasan_kmalloc_large(ret, size, flags);
        /* As ret might get tagged, call kmemleak hook after KASAN. */
        kmemleak_alloc(ret, size, 1, flags);
        return ret;
}
EXPORT_SYMBOL(kmalloc_order);
```

Since this is an allocation request for a large page, we add a __GFP_COMP flag to allocate compound pages, and we get as many pages as we order from the page allocator who uses the buddy system.

kmalloc_index()

include/linux/slab.h

```
#ifndef CONFIG_SLOB
/*
 * Figure out which kmalloc slab an allocation of a certain size
 * belongs to.
 * 0 = zero alloc
 * 1 =  65 .. 96 bytes
 * 2 = 120 .. 192 bytes
 * n = 2^(n-1) .. 2^n -1
 */
```

```
static __always_inline int kmalloc_index(size_t size)
{
        if (!size)
                return 0;

        if (size <= KMALLOC_MIN_SIZE)
                return KMALLOC_SHIFT_LOW;

        if (KMALLOC_MIN_SIZE <= 32 && size > 64 && size <= 96)
                return 1;
        if (KMALLOC_MIN_SIZE <= 64 && size > 128 && size <= 192)
                return 2;
        if (size <=          8) return 3;
        if (size <=         16) return 4;
        if (size <=         32) return 5;
        if (size <=         64) return 6;
        if (size <=        128) return 7;
        if (size <=        256) return 8;
        if (size <=        512) return 9;
        if (size <=       1024) return 10;
        if (size <=   2 * 1024) return 11;
        if (size <=   4 * 1024) return 12;
        if (size <=   8 * 1024) return 13;
        if (size <=  16 * 1024) return 14;
        if (size <=  32 * 1024) return 15;
        if (size <=  64 * 1024) return 16;
        if (size <= 128 * 1024) return 17;
        if (size <= 256 * 1024) return 18;
        if (size <= 512 * 1024) return 19;
        if (size <= 1024 * 1024) return 20;
        if (size <=  2 * 1024 * 1024) return 21;
        if (size <=  4 * 1024 * 1024) return 22;
        if (size <=  8 * 1024 * 1024) return 23;
        if (size <=  16 * 1024 * 1024) return 24;
        if (size <=  32 * 1024 * 1024) return 25;
        if (size <=  64 * 1024 * 1024) return 26;
        BUG();

        /* Will never be reached. Needed because the compiler may compla
in */
        return -1;
}
#endif /* !CONFIG_SLOB */<span style="line-height: 1.5;"> </span>
```

Returns the index according to the request size of 0~64M or less as a number from 0~23. If it exceeds 64M, it returns -1 as an error.

- Index value based on request size
  - 0 = No allocation (zero alloc)
  - 1 = 65 .. 96 bytes
  - 2 = 120 .. 192 bytes
  - n = 2^(n-1) .. 2^n -1
    - 3 = 1 .. 8
    - 4 = 9 .. 16
    - 5 = 17 .. 32
    - 6 = 33 .. 64
    - …
    - 26 = 32M-1 .. 64M
  - However, if the size is 1~KMALLOC_MIN_SIZE, it will return KMALLOC_SHIFT_LOW.
    - rpi2 e.g. KMALLOC_MIN_SIZE=64, KMALLOC_SHIFT_LOW=6

- arm64e) KMALLOC_MIN_SIZE=128, KMALLOC_SHIFT_LOW=7

kmem_cache_alloc_node_trace()

mm/slub.c

```
#ifdef CONFIG_TRACING
void *kmem_cache_alloc_node_trace(struct kmem_cache *s,
                                    gfp_t gfpflags,
                                    int node, size_t size)
{
        void *ret = slab_alloc_node(s, gfpflags, node, _RET_IP_);

        trace_kmalloc_node(_RET_IP_, ret,
                           size, s->size, gfpflags, node);

        ret = kasan_kmalloc(s, ret, size, gfpflags);
        return ret;
}
EXPORT_SYMBOL(kmem_cache_alloc_node_trace);
#endif
```

It is allocated a slab object from the @node of the requested slab cache.

__kmalloc()

mm/slub.c

```
01 void *__kmalloc(size_t size, gfp_t flags)
02 {
03         struct kmem_cache *s;
04         void *ret;
05 
06         if (unlikely(size > KMALLOC_MAX_CACHE_SIZE))
07                 return kmalloc_large(size, flags);
08 
09         s = kmalloc_slab(size, flags);
10 
11         if (unlikely(ZERO_OR_NULL_PTR(s)))
12                 return s;
13 
14         ret = slab_alloc(s, flags, _RET_IP_);
15 
16         trace_kmalloc(_RET_IP_, ret, size, s->size, flags);
17 
18         kasan_kmalloc(s, ret, size);
19 
20         return ret;
21 }
22 EXPORT_SYMBOL(__kmalloc);
```

If @size value is a variable, it is called and processed as follows:

- In line 6~7 of the code, if the size exceeds KMALLOC_MAX_CACHE_SIZE (2 pages in the case of a slub), the page is allocated from the page allocator using the buddy system through the kmalloc_large() function.
- In line 9~12 of code, get the appropriate kmalloc slap cache for size and type.
- It is allocated a slab object from the kmalloc slab cache obtained in line 14 of code.

kmalloc_slab()

mm/slab_common.c

```
/*
 * Find the kmem_cache structure that serves a given size of
 * allocation
 */
```

```
01 struct kmem_cache *kmalloc_slab(size_t size, gfp_t flags)
02 {
03         unsigned int index;
04 
05         if (size <= 192) {
06                 if (!size)
07                         return ZERO_SIZE_PTR;
08 
09                 index = size_index[size_index_elem(size)];
10         } else {
11                 if (WARN_ON_ONCE(size > KMALLOC_MAX_CACHE_SIZE))
12                         return NULL;
13                 index = fls(size - 1);
14         }
15 
16         return kmalloc_caches[kmalloc_type(flags)][index];
17 }
```

@size and the type obtained by @flags retrieves the corresponding kmalloc slab cache.

- In line 5~9 of the code, if the @size is 1 ~ 192, use the already created size_index[] table to calculate the index by size value.
- In lines 10~14 of the code, if the @size is 193 ~ KMALLOC_MAX_SIZE (2 pages in the case of a slub), it returns the required number of bits.
  - 193 ~ 256 → 8
  - 257 ~ 512 → 9
  - 513 ~ 1024 → 10
  - 1025 ~ 2048 → 11
  - 2049 ~ 4096 → 12
  - 4097 ~ 8192 → 13
- In line 16 of the code, we extract the type from the @flags and return the kmallc cache obtained with the calculated index.

size_index_elem()

mm/slab_common.c

```
static inline int size_index_elem(size_t bytes)
{
        return (bytes - 1) / 8;
}
```

size_index[] tables use a range of 8 bytes per index. Thus, it returns the quotient of the request bytes-1 divided by 8.

- 1 ~ 8 → 0
- 9 ~ 16 → 1
- 17 ~ 24 → 2

- 25 ~ 32 → 3
- 33 ~ 40 → 4
- 41 ~ 48 → 5
- ...

consultation

- Slab Memory Allocator -1- (Structure) (http://jake.dothome.co.kr/slub/) | Qc
- Kmalloc vs Vmalloc (http://jake.dothome.co.kr/kmalloc-vs-vmalloc) | 문c
- Kmalloc (http://jake.dothome.co.kr/kmalloc) | Sentence C – Current post
- Vmalloc (http://jake.dothome.co.kr/vmalloc) | Qc
- Vmap((http://jake.dothome.co.kr/vmap)) | Qc

LEAVE A COMMENT

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

Comments

name *

email *

Website

WRITE A COMMENT

slab memory allocator -2- (initialize cache) (http://jake.dothome.co.kr/kmem_cache_init/)

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