Memory Model -4- (APIs)

2016-03-31 (http://jake.dothome.co.kr/mem_map_page/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

Memory Model -4- (APIs)

for_each_migratetype_order()

include/linux/mmzone.h

```
#define for_each_migratetype_order(order, type) \
        for (order = 0; order < MAX_ORDER; order++) \
                for (type = 0; type < MIGRATE_TYPES; type++)
```

- Loop around the number of MAX_ORDER (11) used by the buddy memory allocator.
- Loop as much as you MIGRATE_TYPES responsible for managing migration flags for memory hotplugs.

Zones and Nodes

is_highmem_idx()

include/linux/mmzone.h

```
static inline int is_highmem_idx(enum zone_type idx)
{
#ifdef CONFIG_HIGHMEM
        return (idx == ZONE_HIGHMEM ||
                (idx == ZONE_MOVABLE && movable_zone == ZONE_HIGHMEM));
#else
        return 0;
#endif
}
```

zone_idx()

include/linux/mmzone.h

```
/*
 * zone_idx() returns 0 for the ZONE_DMA zone, 1 for the ZONE_NORMAL zone, etc.
 */
#define zone_idx(zone)          ((zone) - (zone)->zone_pgdat->node_zones)
```

- Returns the zone index number.

  - e.g. 0 and 1 are returned when using ZONE_DMA and ZONE_NORMAL.
  - e.g. 0 is returned if only ZONE_NORMAL is used.

set_page_links()

include/linux/mm.h

```
static inline void set_page_links(struct page *page, enum zone_type zone,
        unsigned long node, unsigned long pfn)
{
        set_page_zone(page, zone);
        set_page_node(page, node);
#ifdef SECTION_IN_PAGE_FLAGS
        set_page_section(page, pfn_to_section_nr(pfn));
#endif
}
```

- Set the zone, node, and section information in page->flags.

set_page_zone()

include/linux/mm.h

```
static inline void set_page_zone(struct page *page, enum zone_type zone)
{
        page->flags &= ~(ZONES_MASK << ZONES_PGSHIFT);
        page->flags |= (zone & ZONES_MASK) << ZONES_PGSHIFT;
}
```

- Set the zone information in page->flags.

set_page_node()

include/linux/mm.h

```
static inline void set_page_node(struct page *page, unsigned long node)
{
        page->flags &= ~(NODES_MASK << NODES_PGSHIFT);
        page->flags |= (node & NODES_MASK) << NODES_PGSHIFT;
}
```

- page->flags.

page_zone_id()

include/linux/mm.h

```
/*
 * The identification function is mainly used by the buddy allocator for
 * determining if two pages could be buddies. We are not really identifying
 * the zone since we could be using the section number id if we do not have
 * node id available in page flags.
 * We only guarantee that it will return the same value for two combinable
 * pages in a zone.
 */
static inline int page_zone_id(struct page *page)
{
        return (page->flags >> ZONEID_PGSHIFT) & ZONEID_MASK;
}
```

Extracts the zone id from the page and returns it.

About the Sprsemem section

set_page_section()

include/linux/mm.h

```
#ifdef SECTION_IN_PAGE_FLAGS
static inline void set_page_section(struct page *page, unsigned long section)
{
        page->flags &= ~(SECTIONS_MASK << SECTIONS_PGSHIFT);
        page->flags |= (section & SECTIONS_MASK) << SECTIONS_PGSHIFT;
}
#endif
```

- page->flags.

__pfn_to_section()

include/linux/mmzone.h

```
static inline struct mem_section *__pfn_to_section(unsigned long pfn)
{
        return __nr_to_section(pfn_to_section_nr(pfn));
}
```

Returns information about the mem_section structure that corresponds to the pfn value.

- pfn_to_section_nr()
  - Get the section number by the pfn value.
- __nr_to_section()
  - Returns mem_section struct information with a section number.

pfn_to_section_nr()

include/linux/mmzone.h

```
#define pfn_to_section_nr(pfn) ((pfn) >> PFN_SECTION_SHIFT)
```

- Returns the sparse index of the pfn.
  - e.g. Realview-PBX
    - Since the section size is 256M units (PFN_SECTION_SHIFT=16), the section number is from 0~15

__nr_to_section()

include/linux/mmzone.h

```
static inline struct mem_section *__nr_to_section(unsigned long nr)
{
        if (!mem_section[SECTION_NR_TO_ROOT(nr)])
                return NULL;
        return &mem_section[SECTION_NR_TO_ROOT(nr)][nr & SECTION_ROOT_MASK];
}
```

- Returns mem_section struct information with a section number.

SECTION_NR_TO_ROOT()

include/linux/mmzone.h

```
#define SECTION_NR_TO_ROOT(sec) ((sec) / SECTIONS_PER_ROOT)
```

- Returns the ROOT number as the section number.

```
#ifdef CONFIG_SPARSEMEM_EXTREME
#define SECTIONS_PER_ROOT       (PAGE_SIZE / sizeof (struct mem_section))
#else
#define SECTIONS_PER_ROOT       1
#endif
```

- Number of sections per ROOT
  - PAGE_SIZE (4K) can fit mem_section structure

present_section_nr()

include/linux/mmzone.h

```
static inline int present_section_nr(unsigned long nr)
{
        return present_section(__nr_to_section(nr));
}
```

Make sure you have a mem_section that corresponds to the section number. If it is not ready, the section means hole.

- __nr_to_section()
  - Retrieves the mem_section struct information by section number.
- present_section()
  - mem_section Check if a section exists in the struct information.

present_section()

include/linux/mmzone.h

```
static inline int present_section(struct mem_section *section)
```

```
{
        return (section && (section->section_mem_map & SECTION_MARKED_PRESENT));
}
```

mem_section Check if a section exists in the struct information.

- Make sure the SECTION_MARKED_PRESENT identification bit is set.

__section_mem_map_addr()

include/linux/mmzone.h

```
static inline struct page *__section_mem_map_addr(struct mem_section *section)
{
        unsigned long map = section->section_mem_map;
        map &= SECTION_MAP_MASK;
        return (struct page *)map;
}
```

Returns the mem_map address for the Sparse memory section.

include/linux/mmzone.h

```
/*
 * We use the lower bits of the mem_map pointer to store
 * a little bit of information.  There should be at least
 * 3 bits here due to 32-bit alignment.
 */
#define SECTION_MARKED_PRESENT  (1UL<<0)
#define SECTION_HAS_MEM_MAP     (1UL<<1)
#define SECTION_MAP_LAST_BIT    (1UL<<2)
#define SECTION_MAP_MASK        (~(SECTION_MAP_LAST_BIT-1))
#define SECTION_NID_SHIFT       2
```

Enabling and disabling references on a page

get_page()

include/linux/mm.h

```
static inline void get_page(struct page *page)
{
        page = compound_head(page);
        /*
         * Getting a normal page or the head of a compound page
         * requires to already have an elevated page->_refcount.
         */
        VM_BUG_ON_PAGE(page_ref_count(page) <= 0, page);
        page_ref_inc(page);
}
```

Increments the reference counter by 1.

get_page_unless_zero()

include/linux/mm.h

```
/*
 * Try to grab a ref unless the page has a refcount of zero, return false if
 * that is the case.
 * This can be called when MMU is off so it must not access
 * any of the virtual mappings.
 */
```

```
static inline int get_page_unless_zero(struct page *page)
{
        return page_ref_add_unless(page, 1, 0);
}
```

After reading the reference counter (p->_refcount), increment only if it differs from the 0 value. If the result value is not zero, it returns true.

put_page()

include/linux/mm.h

```
static inline void put_page(struct page *page)
{
        page = compound_head(page);

        /*
         * For devmap managed pages we need to catch refcount transition from
         * 2 to 1, when refcount reach one it means the page is free and we
         * need to inform the device driver through callback. See
         * include/linux/memremap.h and HMM for details.
         */
        if (put_devmap_managed_page(page))
                return;

        if (put_page_testzero(page))
                __put_page(page);
}
```

Reduces the reference counter by 1. If it reaches 0, the page will be retrieved.

put_page_testzero()

include/linux/mm.h

```
/*
 * Methods to modify the page usage count.
 *
 * What counts for a page usage:
 * - cache mapping   (page->mapping)
 * - private data    (page->private)
 * - page mapped in a task's page tables, each mapping
 *   is counted separately
 *
 * Also, many kernel routines increase the page count before a critical
 * routine so they can be sure the page doesn't go away from under them.
 */

/*
```

```
 * Drop a ref, return true if the refcount fell to zero (the page has no
users)
 */
```

```
static inline int put_page_testzero(struct page *page)
{
        VM_BUG_ON_PAGE(page_ref_count(page) == 0, page);
        return page_ref_dec_and_test(page);
}
```

Decrement the reference counter on the page and check if it is 0 (used complete) to return whether it is used or not.

- 0=in use, 1=in use (when the reference _count becomes 0)

Page vs PFN Conversion

Conversion between PFN and page structure pointers uses two APIs:

include/asm-generic/memory_model.h

```
#define page_to_pfn __page_to_pfn
#define pfn_to_page __pfn_to_page
```

- page_to_pfn()
  - Page struct pointer to get the pfn value.
- pfn_to_page()
  - pfn value to get the pointer to the page structure.

As follows, the conversion method varies depending on the flat and sparse physics memory models, and in the case of the sparse physics model, it is further divided into two types depending on whether vmemmap is used or not.

CONFIG_FLATMEM

```
#define __pfn_to_page(pfn)      (mem_map + ((pfn) - ARCH_PFN_OFFSET))
#define __page_to_pfn(page)     ((unsigned long)((page) - mem_map) + \
                                 ARCH_PFN_OFFSET)
```

- __pfn_to_page()
  - ARCH_PFN_OFFSET refers to the starting PFN value of the physical DRAM.
  - mem_map[@pfn – Physics DRAM Start PFN]

CONFIG_SPARSEMEM

```
/*
 * Note: section's mem_map is encoded to reflect its start_pfn.
 * section[i].section_mem_map == mem_map's address - start_pfn;
 */
#define __page_to_pfn(pg)                                       \
({      const struct page *__pg = (pg);                         \
        int __sec = page_to_section(__pg);                      \
        (unsigned long)(__pg - __section_mem_map_addr(__nr_to_section(__sec))); \
})

#define __pfn_to_page(pfn)                              \
({      unsigned long __pfn = (pfn);                    \
        struct mem_section *__sec = __pfn_to_section(__pfn);    \
        __section_mem_map_addr(__sec) + __pfn;          \
})
```

- __pfn_to_page()
  - Replace pfn with sections, and then access mem_section[][] to return the mam_map[@pfn] address for the section.

CONFIG_SPARSEMEM & CONFIG_SPARSEMEM_VMEMMAP

```
/* memmap is virtually contiguous.  */
#define __pfn_to_page(pfn)      (vmemmap + (pfn))
#define __page_to_pfn(page)     (unsigned long)((page) - vmemmap)
```

- __pfn_to_page()
  - = mem_map[@pfn]
  - vmemmap = mem_map[0].

Page Flags

include/linux/page-flags.h

```
/*
 * Various page->flags bits:
 *
 * PG_reserved is set for special pages. The "struct page" of such a page
 * should in general not be touched (e.g. set dirty) except by its owner.
 * Pages marked as PG_reserved include:
 * - Pages part of the kernel image (including vDSO) and similar (e.g. BIOS,
 *   initrd, HW tables)
 * - Pages reserved or allocated early during boot (before the page allocator
 *   was initialized). This includes (depending on the architecture) the
 *   initial vmemmap, initial page tables, crashkernel, elfcorehdr, and much
 *   much more. Once (if ever) freed, PG_reserved is cleared and they will
 *   be given to the page allocator.
 * - Pages falling into physical memory gaps - not IORESOURCE_SYSRAM. Trying
 *   to read/write these pages might end badly. Don't touch!
 * - The zero page(s)
 * - Pages not added to the page allocator when onlining a section because
 *   they were excluded via the online_page_callback() or because they are
 *   PG_hwpoison.
 * - Pages allocated in the context of kexec/kdump (loaded kernel image,
 *   control pages, vmcoreinfo)
 * - MMIO/DMA pages. Some architectures don't allow to ioremap pages that are
 *   not marked PG_reserved (as they might be in use by somebody else who does
 *   not respect the caching strategy).
 * - Pages part of an offline section (struct pages of offline sections should
 *   not be trusted as they will be initialized when first onlined).
 * - MCA pages on ia64
 * - Pages holding CPU notes for POWER Firmware Assisted Dump
 * - Device memory (e.g. PMEM, DAX, HMM)
 * Some PG_reserved pages will be excluded from the hibernation image.
 * PG_reserved does in general not hinder anybody from dumping or swapping
 * and is no longer required for remap_pfn_range(). ioremap might require it.
 * Consequently, PG_reserved for a page mapped into user space can indicate
 * the zero page, the vDSO, MMIO pages or device memory.
 *
 * The PG_private bitflag is set on pagecache pages if they contain filesystem
 * specific data (which is normally at page->private). It can be used by
 * private allocations for its own usage.
 *
 * During initiation of disk I/O, PG_locked is set. This bit is set before I/O
 * and cleared when writeback _starts_ or when read _completes_. PG_writeback
 * is set before writeback starts and cleared when it finishes.
 *
 * PG_locked also pins a page in pagecache, and blocks truncation of the file
 * while it is held.
 *
 * page_waitqueue(page) is a wait queue of all tasks waiting for the page
 * to become unlocked.
 *
 * PG_swapbacked is set when a page uses swap as a backing storage.  This are
 * usually PageAnon or shmem pages but please note that even anonymous pages
 * might lose their PG_swapbacked flag when they simply can be dropped (e.g. as
 * a result of MADV_FREE).
 *
 * PG_uptodate tells whether the page's contents is valid.  When a read
 * completes, the page becomes uptodate, unless a disk I/O error happened.
 *
 * PG_referenced, PG_reclaim are used for page reclaim for anonymous and
 * file-backed pagecache (see mm/vmscan.c).
 *
 * PG_error is set to indicate that an I/O error occurred on this page.
 *
 * PG_arch_1 is an architecture specific page state bit.  The generic code
 * guarantees that this bit is cleared for a page when it first is entered into
 * the page cache.
 *
 * PG_hwpoison indicates that a page got corrupted in hardware and contains
 * data with incorrect ECC bits that triggered a machine check. Accessing is
 * not safe since it may cause another machine check. Don't touch!
 */
```

```
/*
 * Don't use the pageflags directly.  Use the PageFoo macros.
 *
 * The page flags field is split into two parts, the main flags area
 * which extends from the low bits upwards, and the fields area which
 * extends from the high bits downwards.
 *
 *  | FIELD | ... | FLAGS |
 *  N-1           ^       0
 *               (NR_PAGEFLAGS)
 *
 * The fields area is reserved for fields mapping zone, node (for NUMA) and
 * SPARSEMEM section (for variants of SPARSEMEM that require section ids like
 * SPARSEMEM_EXTREME with !SPARSEMEM_VMEMMAP).
```

```
enum pageflags {
        PG_locked,              /* Page is locked. Don't touch. */
        PG_referenced,
        PG_uptodate,
        PG_dirty,
        PG_lru,
        PG_active,
        PG_workingset,
        PG_waiters,             /* Page has waiters, check its waitqueue. Must be bit #7 and in the same byte as "PG_locked" */
        PG_error,
        PG_slab,
        PG_owner_priv_1,        /* Owner use. If pagecache, fs may use*/
        PG_arch_1,
        PG_reserved,
        PG_private,             /* If pagecache, has fs-private data */
        PG_private_2,           /* If pagecache, has fs aux data */
        PG_writeback,           /* Page is under writeback */
        PG_head,                /* A head page */
        PG_mappedtodisk,        /* Has blocks allocated on-disk */
        PG_reclaim,             /* To be reclaimed asap */
        PG_swapbacked,          /* Page is backed by RAM/swap */
        PG_unevictable,         /* Page is "unevictable"  */
#ifdef CONFIG_MMU
        PG_mlocked,             /* Page is vma mlocked */
#endif
#ifdef CONFIG_ARCH_USES_PG_UNCACHED
        PG_uncached,            /* Page has been mapped as uncached */
#endif
#ifdef CONFIG_MEMORY_FAILURE
        PG_hwpoison,            /* hardware poisoned page. Don't touch */
#endif
#if defined(CONFIG_PAGE_IDLE_FLAG) && defined(CONFIG_64BIT)
        PG_young,
        PG_idle,
#endif
#ifdef CONFIG_64BIT
        PG_arch_2,
#endif
#ifdef CONFIG_KASAN_HW_TAGS
        PG_skip_kasan_poison,
#endif
        __NR_PAGEFLAGS,

        /* Filesystems */
        PG_checked = PG_owner_priv_1,

        /* SwapBacked */
        PG_swapcache = PG_owner_priv_1, /* Swap page: swp_entry_t in private */

        /* Two page bits are conscripted by FS-Cache to maintain local caching
         * state.  These bits are set on pages belonging to the netfs's inodes
         * when those inodes are being locally cached.
         */
        PG_fscache = PG_private_2,      /* page backed by cache */

        /* XEN */
        /* Pinned in Xen as a read-only pagetable page. */
        PG_pinned = PG_owner_priv_1,
        /* Pinned as part of domain save (see xen_mm_pin_all()). */
        PG_savepinned = PG_dirty,
        /* Has a grant mapping of another (foreign) domain's page. */
        PG_foreign = PG_owner_priv_1,
        /* Remapped by swiotlb-xen. */
        PG_xen_remapped = PG_owner_priv_1,

        /* SLOB */
        PG_slob_free = PG_private,

        /* Compound pages. Stored in first tail page's flags */
        PG_double_map = PG_workingset,

#ifdef CONFIG_MEMORY_FAILURE
        /*
         * Compound pages. Stored in first tail page's flags.
         * Indicates that at least one subpage is hwpoisoned in the
         * THP.
         */
        PG_has_hwpoisoned = PG_mappedtodisk,
#endif

        /* non-lru isolated movable page */
        PG_isolated = PG_reclaim,

        /* Only valid for buddy pages. Used to track pages that are reported */
        PG_reported = PG_uptodate,
};
```

Additional information logged in page->flags

linux/page-flags-layout.h

```
/*
 * page->flags layout:
 *
 * There are five possibilities for how page->flags get laid out.  The first
 * pair is for the normal case without sparsemem. The second pair is for
 * sparsemem when there is plenty of space for node and section information.
 * The last is when there is insufficient space in page->flags and a separate
 * lookup is necessary.
 *
 * No sparsemem or sparsemem vmemmap: |       NODE     | ZONE |             ... | FLAGS |
 *      " plus space for last_cpupid: |       NODE     | ZONE | LAST_CPUPID ... | FLAGS |
 * classic sparse with space for node:| SECTION | NODE | ZONE |             ... | FLAGS |
 *      " plus space for last_cpupid: | SECTION | NODE | ZONE | LAST_CPUPID ... | FLAGS |
 * classic sparse no space for node:  | SECTION |     ZONE    | ... | FLAGS |
 */
```

Depending on the kernel configuration, page->flags records SECTION, NODE, ZONE, and LAST_CPUPID information in addition to flags.

Reserved flag (example)

PageReserved(), SetPageReserved(), ClearPageReserved(), __ClearPageReserved()

include/linux/page-flags.h

```
PAGEFLAG(Reserved, reserved) __CLEARPAGEFLAG(Reserved, reserved)
```

- PageReserved(), SetPageReserved(), ClearPageReserved(), and __ClearPageReserved() static inline functions are created.

```
#define PAGEFLAG(uname, lname) TESTPAGEFLAG(uname, lname) \
        SETPAGEFLAG(uname, lname) CLEARPAGEFLAG(uname, lname)
```

- The following macros are used to create the PageXXX(), SetPageXXX(), and ClearPageXXX() static inline functions.

```
/*
 * Macros to create function definitions for page flags
 */
#define TESTPAGEFLAG(uname, lname) \
static inline int Page##uname(const struct page *page) \
                        { return test_bit(PG_##lname, &page->flags); }

#define SETPAGEFLAG(uname, lname) \
static inline void SetPage##uname(struct page *page) \
                        { set_bit(PG_##lname, &page->flags); }

#define CLEARPAGEFLAG(uname, lname) \
static inline void ClearPage##uname(struct page *page) \
                        { clear_bit(PG_##lname, &page->flags); }
```

```
#define __CLEARPAGEFLAG(uname, lname) \
static inline void __ClearPage##uname(struct page *page) \
                        { __clear_bit(PG_##lname, &page->flags); }
```

- test_bit()
  - &page->flags to find out whether the PG_xxxxx number bit has been set.
- set_bit()
  - &page->flags sets the PG_xxxxx number bit atomically.
- clear_bit()
  - &PG_xxxxx number bit of &page->flags is atomically cleared.
- __clear_bit()
  - Clear the PG_xxxxx number bit of &page->flags. (non-atomic)

Regrouping of some flags (P->page_type)

The four PG_buddy, PG_ballon, PG_kmemcg, and PG_table flags below have been replaced with P->_mapcount, which is then declared as a union and used as a shared P->page_type.

- Flags that were initially managed by P->Flags were split into P->_mapcount.
- The new kernel then uses p->page_type shared as a union instead of p->_mapcount. However, since the initial value of p->_mapcount is -1 (0xffff_ffff), the bits are set and turned off in reverse.
  - 예) Set Buddy
    - Old kernel: p->_mapcount = PAGE_BUDDY_MAPCOUNT_VALUE(-128)
    - new 커널: p->page_type &= ~0x80
  - 예) Clear Buddy
    - p->_mapcount = PAGE_BUDDY_MAPCOUNT_VALUE(-1)
    - new 커널: p->page_type |= 0x80
  - 참고: mm: split page_type out from _mapcount (https://github.com/torvalds/linux/commit/6e292b9be7f4358985ce33ae1f59ab30a8c09e08)

include/linux/page-flags.h

```
/*
 * PageBuddy() indicates that the page is free and in the buddy system
 * (see mm/page_alloc.c).
 */
PAGE_TYPE_OPS(Buddy, buddy)

/*
 * PageBalloon() is true for pages that are on the balloon page list
 * (see mm/balloon_compaction.c).
 */
PAGE_TYPE_OPS(Balloon, balloon)

/*
 * If kmemcg is enabled, the buddy allocator will set PageKmemcg() on
 * pages allocated with __GFP_ACCOUNT. It gets cleared on page free.
 */
PAGE_TYPE_OPS(Kmemcg, kmemcg)

/*
 * Marks pages in use as page tables.
 */
```

```
PAGE_TYPE_OPS(Table, table)
```

```
/*
 * For pages that are never mapped to userspace (and aren't PageSlab),
 * page_type may be used.  Because it is initialised to -1, we invert the
 * sense of the bit, so __SetPageFoo *clears* the bit used for PageFoo, and
 * __ClearPageFoo *sets* the bit used for PageFoo.  We reserve a few high and
 * low bits so that an underflow or overflow of page_mapcount() won't be
 * mistaken for a page type value.
 */
```

```
#define PAGE_TYPE_BASE  0xf0000000
/* Reserve                0x0000007f to catch underflows of page_mapcount */
#define PAGE_MAPCOUNT_RESERVE   -128
#define PG_buddy        0x00000080
#define PG_balloon      0x00000100
#define PG_kmemcg       0x00000200
#define PG_table        0x00000400

#define PageType(page, flag)                                            \
        ((page->page_type & (PAGE_TYPE_BASE | flag)) == PAGE_TYPE_BASE)

static inline int page_has_type(struct page *page)
{
        return (int)page->page_type < PAGE_MAPCOUNT_RESERVE;
}

#define PAGE_TYPE_OPS(uname, lname)                                     \
static __always_inline int Page##uname(struct page *page)               \
{                                                                       \
        return PageType(page, PG_##lname);                              \
}                                                                       \
static __always_inline void __SetPage##uname(struct page *page)         \
{                                                                       \
        VM_BUG_ON_PAGE(!PageType(page, 0), page);                       \
        page->page_type &= ~PG_##lname;                                 \
}                                                                       \
static __always_inline void __ClearPage##uname(struct page *page)       \
{                                                                       \
        VM_BUG_ON_PAGE(!Page##uname(page), page);                       \
        page->page_type |= PG_##lname;                                  \
}
```

The above macros create inline functions such as PageBuddy(), __SetPageBuddy(), and __ClearPageBuddy().

About Page Blocks

set_pageblock_flags_group()

linux/pageblock-flags.h

```
#define set_pageblock_flags_group(page, flags, start_bitidx, end_bitidx) \
        set_pfnblock_flags_mask(page, flags, page_to_pfn(page),          \
                        end_bitidx,                                       \
                        (1 << (end_bitidx - start_bitidx + 1)) - 1)
```

set_pfnblock_flags_mask()

mm/page_alloc.c

```
/**
 * set_pfnblock_flags_mask - Set the requested group of flags for a pageblock_nr_pages block of pages
 * @page: The page within the block of interest
 * @flags: The flags to set
 * @pfn: The target page frame number
 * @end_bitidx: The last bit of interest
 * @mask: mask of bits that the caller is interested in
 */
void set_pfnblock_flags_mask(struct page *page, unsigned long flags,
                                        unsigned long pfn,
                                        unsigned long end_bitidx,
                                        unsigned long mask)
{
        struct zone *zone;
        unsigned long *bitmap;
        unsigned long bitidx, word_bitidx;
        unsigned long old_word, word;

        BUILD_BUG_ON(NR_PAGEBLOCK_BITS != 4);

        zone = page_zone(page);
        bitmap = get_pageblock_bitmap(zone, pfn);
        bitidx = pfn_to_bitidx(zone, pfn);
        word_bitidx = bitidx / BITS_PER_LONG;
        bitidx &= (BITS_PER_LONG-1);

        VM_BUG_ON_PAGE(!zone_spans_pfn(zone, pfn), page);

        bitidx += end_bitidx;
        mask <<= (BITS_PER_LONG - bitidx - 1);
        flags <<= (BITS_PER_LONG - bitidx - 1);

        word = ACCESS_ONCE(bitmap[word_bitidx]);
        for (;;) {
                old_word = cmpxchg(&bitmap[word_bitidx], word, (word & ~mask) | flags);
                if (word == old_word)
                        break;
                word = old_word;
        }
}
```

get_pfnblock_flags_mask()

mm/page_alloc.c

```
/**
 * get_pfnblock_flags_mask - Return the requested group of flags for the pageblock_nr_pages block of pages
 * @page: The page within the block of interest
 * @pfn: The target page frame number
 * @end_bitidx: The last bit of interest to retrieve
 * @mask: mask of bits that the caller is interested in
 *
 * Return: pageblock_bits flags
 */
unsigned long get_pfnblock_flags_mask(struct page *page, unsigned long pfn,
                                        unsigned long end_bitidx,
                                        unsigned long mask)
{
        struct zone *zone;
        unsigned long *bitmap;
        unsigned long bitidx, word_bitidx;
        unsigned long word;

        zone = page_zone(page);
        bitmap = get_pageblock_bitmap(zone, pfn);
        bitidx = pfn_to_bitidx(zone, pfn);
        word_bitidx = bitidx / BITS_PER_LONG;
        bitidx &= (BITS_PER_LONG-1);

        word = bitmap[word_bitidx];
        bitidx += end_bitidx;
        return (word >> (BITS_PER_LONG - bitidx - 1)) & mask;
}
```

get_pageblock_bitmap()

mm/page_alloc.c

```
/* Return a pointer to the bitmap storing bits affecting a block of pages */
static inline unsigned long *get_pageblock_bitmap(struct zone *zone,
                                                        unsigned long pfn)
{
#ifdef CONFIG_SPARSEMEM
        return __pfn_to_section(pfn)->pageblock_flags;
#else
        return zone->pageblock_flags;
#endif /* CONFIG_SPARSEMEM */
}
```

Returns a page block bitmap containing the @pfn. (usemap)

- The usemap stores the mobility flags, which are represented in 4 bits.

pfn_to_bitidx()

mm/page_alloc.c

```
static inline int pfn_to_bitidx(struct zone *zone, unsigned long pfn)
{
#ifdef CONFIG_SPARSEMEM
        pfn &= (PAGES_PER_SECTION-1);
        return (pfn >> pageblock_order) * NR_PAGEBLOCK_BITS;
#else
        pfn = pfn - round_down(zone->zone_start_pfn, pageblock_nr_pages);
        return (pfn >> pageblock_order) * NR_PAGEBLOCK_BITS;
#endif /* CONFIG_SPARSEMEM */
}
```

Returns the bit index from the pageblock for the pfn.

SECTION_BLOCKFLAGS_BITS

include/linux/mmzone.h

```
#define SECTION_BLOCKFLAGS_BITS \
        ((1UL << (PFN_SECTION_SHIFT - pageblock_order)) * NR_PAGEBLOCK_BITS)
```

Number of pageblock bits per section

- NR_PAGEBLOCK_BITS
    - Number of bits required for pageblock=4
- PFN_SECTION_SHIFT
    - Number of bits required for section length representation – minus number of bits required for page length representation
        - arm64: section length=27 (128M representation) bits – 12 (4KB representation) bits = 15
- e.g. if arm64 has section size=128M, pageblock_order=9
    - SECTION_BLOCKFLAGS_BITS = 2^(15-9) * 4 bits = 256 bits

consultation

- Memory Model -1- (Basic) (http://jake.dothome.co.kr/mm-1) | 문c
- Memory Model -2- (mem_map) (http://jake.dothome.co.kr/mem_map) | 문c
- Memory Model -3- (Sparse Memory) (http://jake.dothome.co.kr/sparsemem/) | 문c
- Memory Model -4- (APIs) (http://jake.dothome.co.kr/mem_map_page) | Sentence C – Current post
- ZONE Type (http://jake.dothome.co.kr/zone-types) | Qc
- bootmem_init (http://jake.dothome.co.kr/bootmem_init-64) | Qc
- zone_sizes_init() (http://jake.dothome.co.kr/free_area_init_node/) | Qc
- NUMA -1- (ARM64 initialization) (http://jake.dothome.co.kr/numa-1) | Qc
- build_all_zonelists() (http://jake.dothome.co.kr/build_all_zonelists) | Qc

- An introduction to compound pages (https://lwn.net/Articles/619514/) | LWN.net
- agemap, from the userspace perspective (http://www.kernel.org/doc/Documentation/vm/pagemap.txt) | kernel.org
- Documentation/vm/hugetlbpage.txt (http://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt) | kernel.org

LEAVE A COMMENT

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

Comments

name *

email *

Website

WRITE A COMMENT

‹ Memory Model -3- (Sparse Memory) (http://jake.dothome.co.kr/sparsemem/)

__flush_dcache_page() › (http://jake.dothome.co.kr/__flush_dcache_page/)