Memory Model -2- (mem_map)

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

<kernel v5.15>

Memory Model -2- (mem_map)

Learn how to manage page frame numbers (PFNs) and mem_map by physics model.

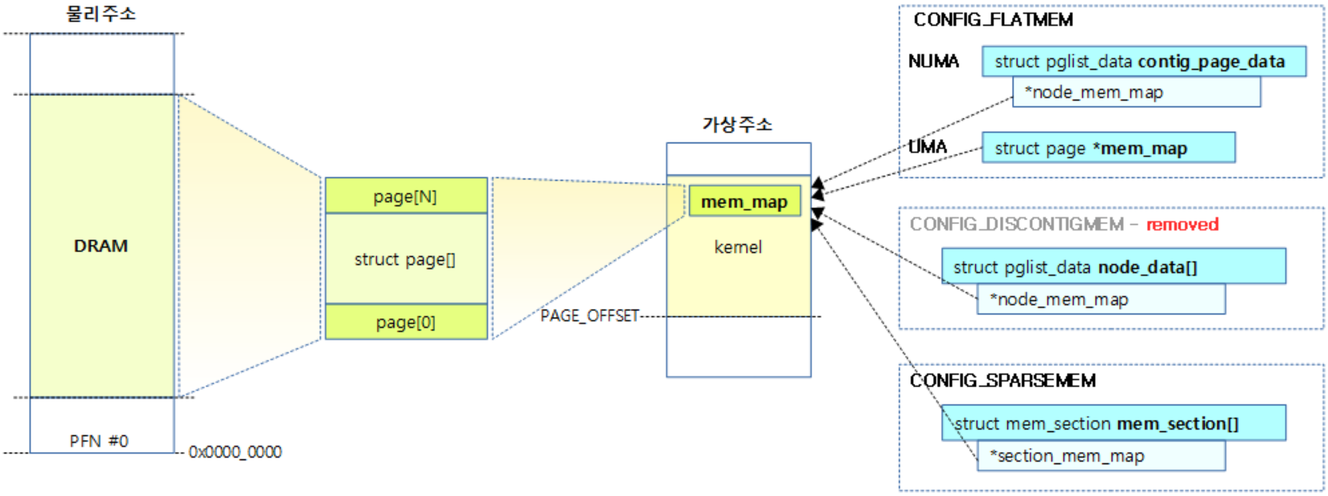
PFN(Page Frame Number)

A PFN is a page-by-page number used for a physical address, starting with zero. 0K is usually used for page units, and in the case of ARM4, there are systems that are set to 64K and 16K for the purpose of operating large-scale databases.

- e.g. 4K Page Standard: 0x0000_0000
  - The PFN value is 0.
- e.g. 4K Page Standard: 0x1234_5678
  - The PFN value is 0x12345.

mem_map

- mem_map is an array of page structures that contain information about all physical memory page frames.
  - In Linux, it started with an array of reference counters of type unsigned short, but gradually the number of members it needed increased, and it became the page structure of today.
- The approach is different for NUMA and UMA systems, and for each physical memory model.
- The path to the initialization function in mem_map varies by architecture and kernel configuration.
  - arm with flatmem
    - setup_arch() → paging_init() → bootmem_init() → zone_sizes_init() → free_area_init_node() (http://jake.dothome.co.kr/free_area_init_node)
  - arm/arm64 with sparsemem
    - setup_arch() → bootmem_init() → sparse_init()



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mem_map-1d.png)

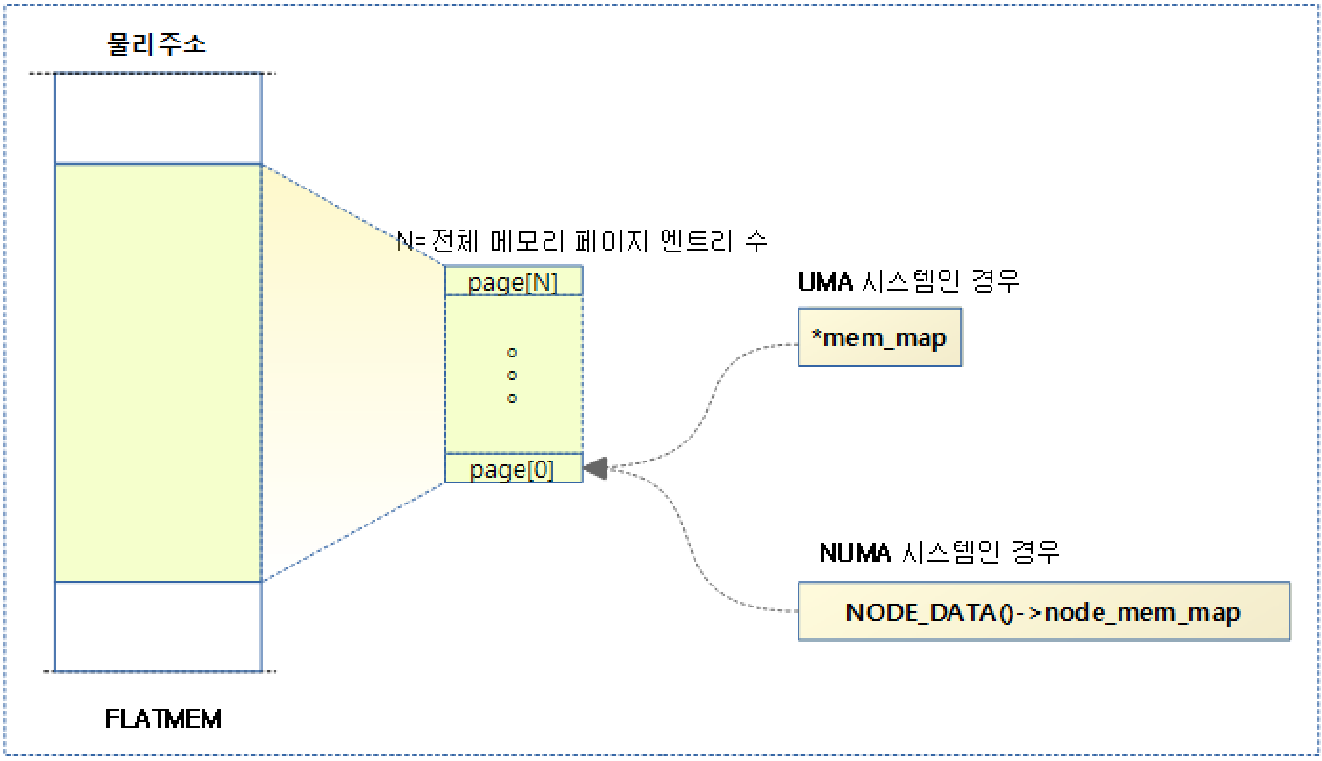
PFN vs page 변환 API

The Linux kernel uses very frequent conversions to access page structures with PFN values, and vice versa. The corresponding APIs are as follows.

- pfn_to_page()
  - Returns a pointer to the page structure with a pfn number.
- page_to_pfn()
  - Returns the pfn number as a pointer to the page structure.

Flat Memory with mem_map

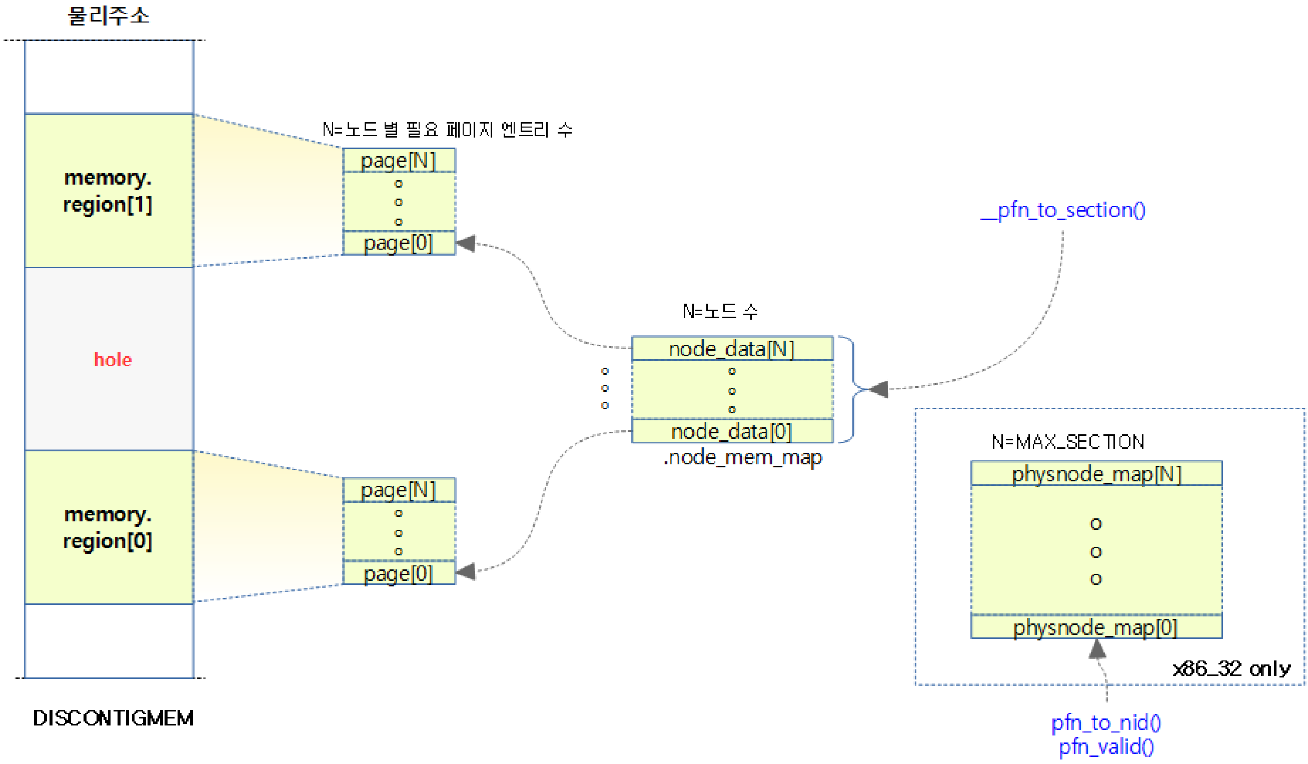
- When the NEED_MULTIPLE_NODES kernel option is not used, the *mem_map pointer variable points to an array of a single page[] structure.
- If you use the FLAT_NODE_MEM_MAP kernel option, contig_page_data.node_mem_map points to an array of page[] structures.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mm-8c.png)

Discontiguous Memory with mem_map

- node_data[].node_mem_map refers to an array of page[] structures.
- In x86_32, a separate implementation of a table for section-to-node mapping is used.
- It only manages the memory it actually uses, and it doesn't manage the hole area at all.
- In the kernel mainline, this model has been completely removed so that it can no longer be used.
  - mm: remove CONFIG_DISCONTIGMEM (https://github.com/torvalds/linux/commit/bb1c50d3967f69f413b333713c2718d48d1ab7ea#diff-6a0166b1d8bbb576287048e4de42eb7e8a1f118e357f407d3bdf7da9fc26d94d) (2021, v5.14-rc1)



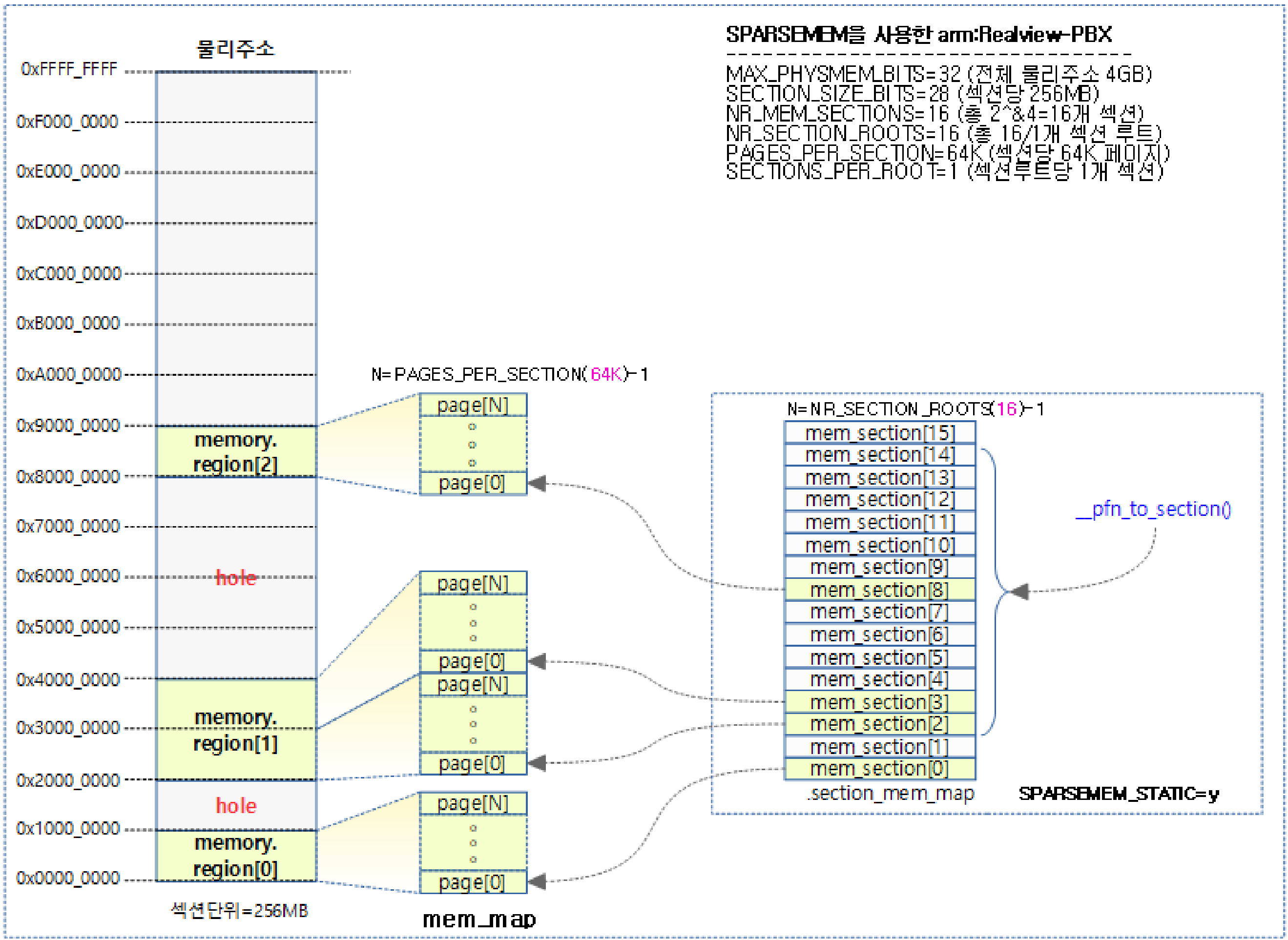
(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mm-6.png)

Sparse Memory with mem_map

- It manages the mem_map through multiple sections, and uses mem_section structs to manage a single section.
- The size of a single section should be tens of MB ~ several GB in size.
  - Note: Not to be confused with the section terminology used in the page table.
  - In the case of the arm64 system, the initial 1G was used, but it was changed to a slightly smaller unit, the 128M.
    - Note: arm64/sparsemem: reduce SECTION_SIZE_BITS (https://github.com/torvalds/linux/commit/f0b13ee23241846f6f6cd0d119a8ac8059416ec4#diff-9590f22a80ad3a29d9c97bab548e481c48ac04e5ca5441c9360c69458829634c) (2021, v5.12-rc1)
- A mem_section struct points to and manages a PAGES_PER_SECTION number of page[] arrays.
- There are two implementations of mem_section structs that manage sections to manage memory allocations.
  - SPARSEMEM_STATIC
    - On systems with small physical memory, such as 32-bit systems, mem_secton structures are generated at compile time en masse.
    - mem_section[][1] is a double array, but does not use a second index.
  - SPARSEMEM_EXTREME
    - On systems with large physical memory sizes, such as 64-bit systems, mem_secton structs are created and used at runtime when needed.
      - It is designed to reduce memory waste in case the hole size is very large.

   - **mem_section Use a double pointer, and use it in the form of a mem_section[SECTIONS_PER_ROOT][SECTIONS_PER_ROOT], and is used as follows:
     - For the first array index, it means the section root
       - NR_SECTION_ROOTS (number of section roots) = NR_MEM_SECTIONS (number of sections to use for total memory) / SECTIONS_PER_ROOT (number of mem_section structs that can fit in a 1 page frame)
     - The second array index refers to the mem_section index within the root section.
       - SECTIONS_PER_ROOT (the number of mem_section structs that can fit in a 1 page frame)

The following illustration shows a 32-bit arm – Realview-PBX board without using SPARSEMEM_EXTREME.

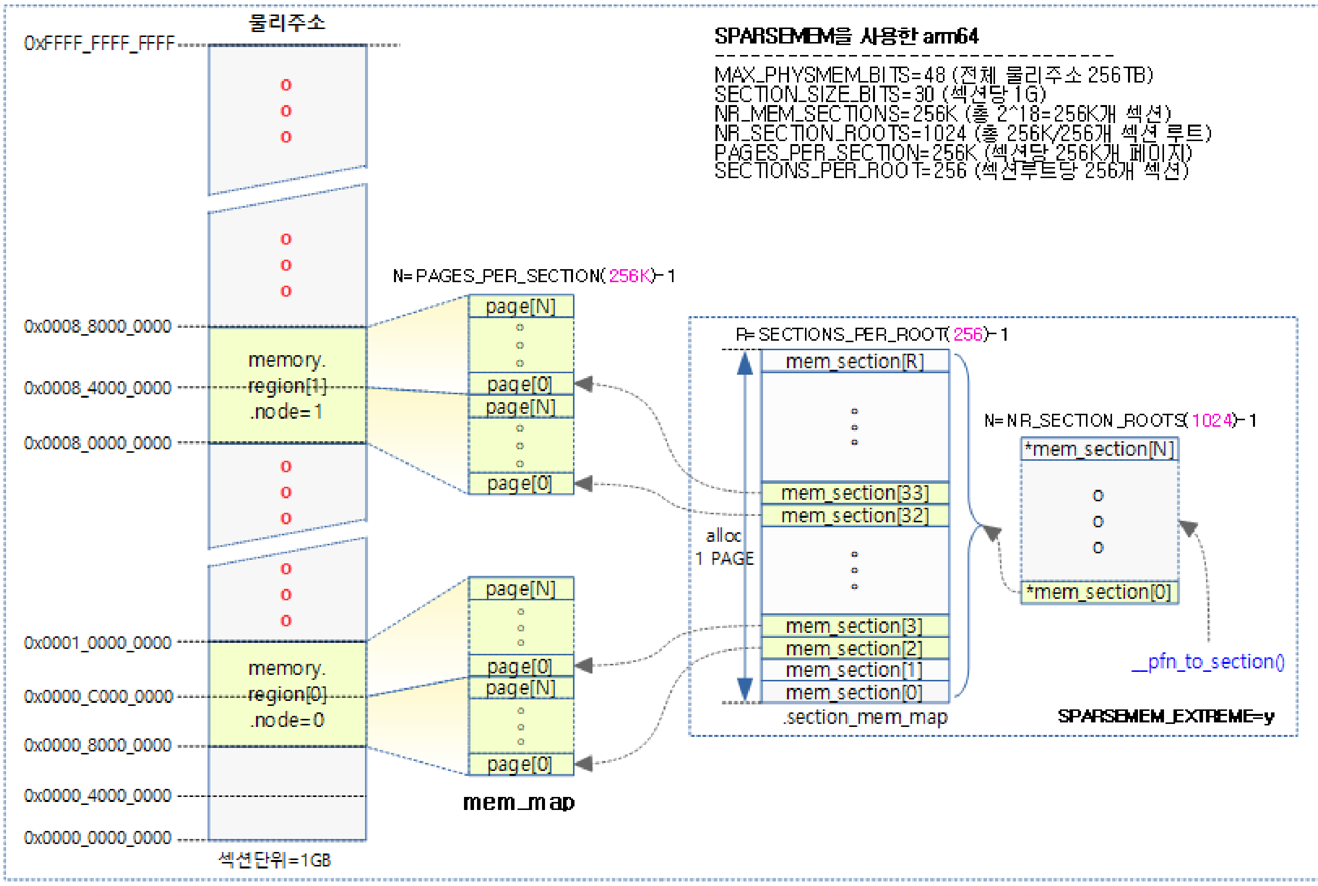


(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mm-7c.png)

The following figure shows the use of 64G DRAM in the arm4 architecture, using SPARSEMEM_EXTREME.

- Below, we have shown an example of using 1G for the section size, but since it has recently been changed to 128M, this should be taken into account depending on the kernel version.
  - Note: arm64/sparsemem: reduce SECTION_SIZE_BITS (https://github.com/torvalds/linux/commit/f0b13ee23241846f6f6cd0d119a8ac8059416ec4

#diff-9590f22a80ad3a29d9c97bab548e481c48ac04e5ca5441c9360c69458829634c) (2021, v5.12-rc1)

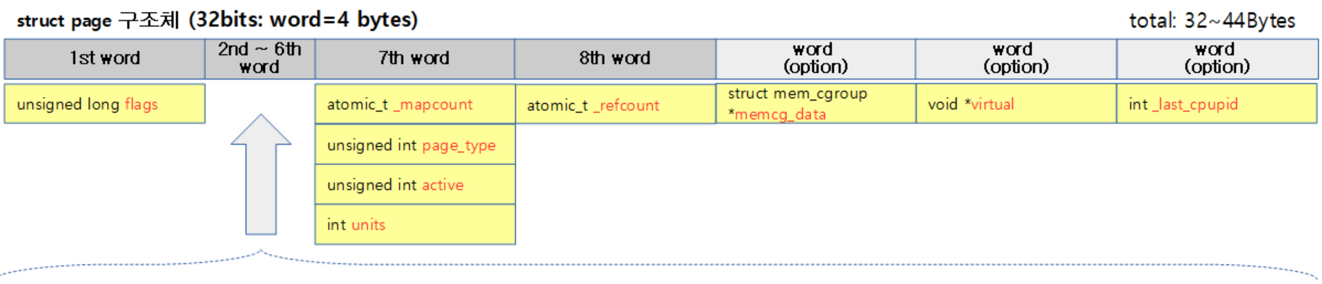


(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mm-9b.png)

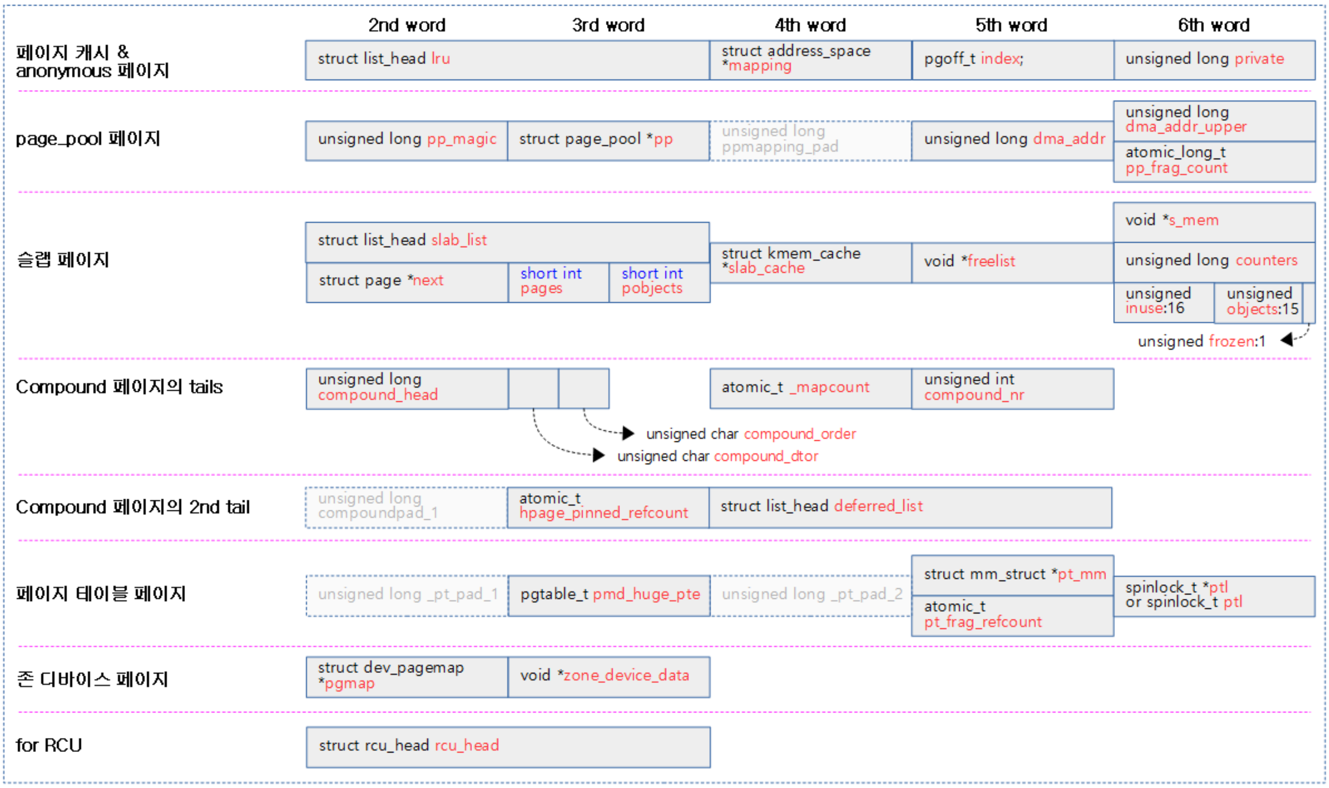
page descriptor

One page descriptor is assigned to every physical memory page. It is generated for every memory page, so it is size-sensitive. Therefore, in order to reduce the size as much as possible, it is designed to group the members managed in the page descriptor into a union type.

The following shows the page descriptor that works on a 32-bit system. (Use 32 bytes as the minimum configuration excluding all options.)

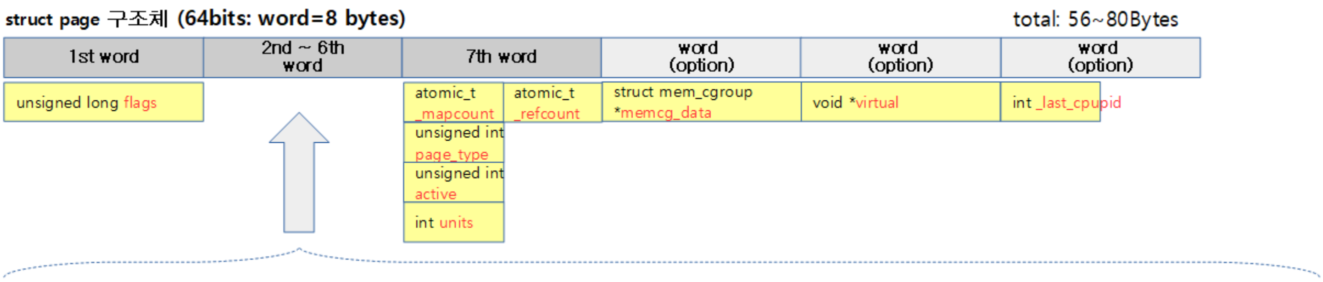


(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mem_map-3b.png)

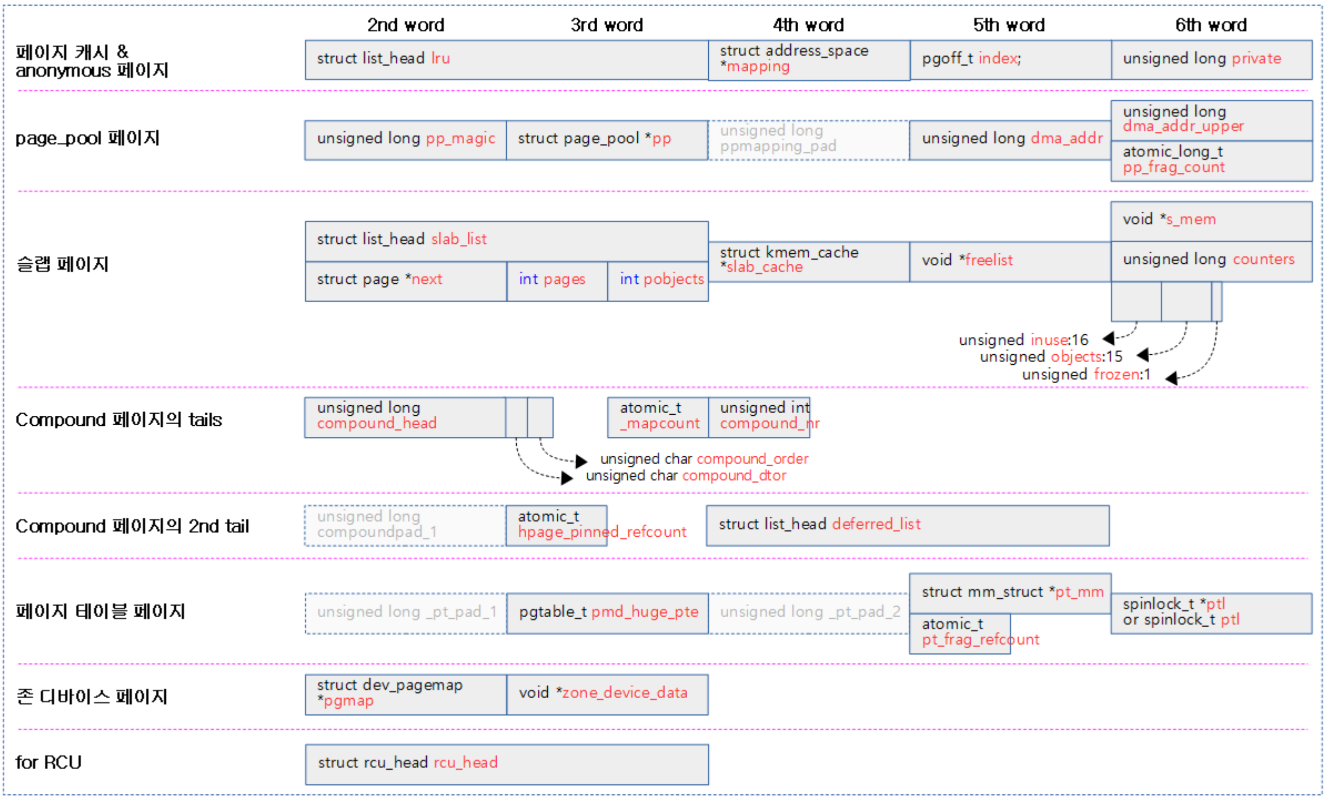


(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mem_map-4b.png)

The following shows the page descriptor that works on a 64-bit system. (The default option is 64 bytes.)



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mem_map-5a.png)



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/mem_map-6c.png)

struct page

include/linux/mm_types.h -1/4-

```
/*
 * Each physical page in the system has a struct page associated with
 * it to keep track of whatever it is we are using the page for at the
 * moment. Note that we have no way to track which tasks are using
 * a page, though if it is a pagecache page, rmap structures can tell us
 * who is mapping it.
 *
 * If you allocate the page using alloc_pages(), you can use some of the
 * space in struct page for your own purposes.  The five words in the main
 * union are available, except for bit 0 of the first word which must be
 * kept clear.  Many users use this word to store a pointer to an object
 * which is guaranteed to be aligned.  If you use the same storage as
 * page->mapping, you must restore it to NULL before freeing the page.
 *
 * If your page will not be mapped to userspace, you can also use the four
 * bytes in the mapcount union, but you must call page_mapcount_reset()
 * before freeing it.
 *
 * If you want to use the refcount field, it must be used in such a way
 * that other CPUs temporarily incrementing and then decrementing the
 * refcount does not cause problems.  On receiving the page from
 * alloc_pages(), the refcount will be positive.
 *
 * If you allocate pages of order > 0, you can use some of the fields
 * in each subpage, but you may need to restore some of their values
 * afterwards.
 *
 * SLUB uses cmpxchg_double() to atomically update its freelist and
 * counters.  That requires that freelist & counters be adjacent and
 * double-word aligned.  We align all struct pages to double-word
 * boundaries, and ensure that 'freelist' is aligned within the
```

```
 * struct.
 */
```

```
struct page {
	unsigned long flags;		/* Atomic flags, some possibly
					 * updated asynchronously */
	/*
	 * Five words (20/40 bytes) are available in this union.
	 * WARNING: bit 0 of the first word is used for PageTail(). That
	 * means the other users of this union MUST NOT use the bit to
	 * avoid collision and false-positive PageTail().
	 */
	union {
		struct {	/* Page cache and anonymous pages */
			/**
			 * @lru: Pageout list, eg. active_list protected by
			 * lruvec->lru_lock.  Sometimes used as a generic list
			 * by the page owner.
			 */
			struct list_head lru;
			/* See page-flags.h for PAGE_MAPPING_FLAGS */
			struct address_space *mapping;
			pgoff_t index;		/* Our offset within mapping. */
			/**
			 * @private: Mapping-private opaque data.
			 * Usually used for buffer_heads if PagePrivate.
			 * Used for swp_entry_t if PageSwapCache.
			 * Indicates order in the buddy system if PageBuddy.
			 */
			unsigned long private;
		};
		struct {	/* page_pool used by netstack */
			/**
			 * @pp_magic: magic value to avoid recycling non
			 * page_pool allocated pages.
			 */
			unsigned long pp_magic;
			struct page_pool *pp;
			unsigned long _pp_mapping_pad;
			unsigned long dma_addr;
			union {
				/**
				 * dma_addr_upper: might require a 64-bit
				 * value on 32-bit architectures.
				 */
				unsigned long dma_addr_upper;
				/**
				 * For frag page support, not supported in
				 * 32-bit architectures with 64-bit DMA.
				 */
				atomic_long_t pp_frag_count;
			};
		};
```

First Word

- flags
  - Page Flags

1) Page cache or anonymous page

- _lru
  - Used to connect to an LRU list.
- *mapping
  - It contains pointers related to user mapping, and the bottom 2 bits are used as flags to distinguish their purpose.
    - When used as a page cache, it points to a address_space structure.
      - Non-LRU movable pages are managed by adding PAGE_MAPPING_MOVABLE flags to address_space structure pointers.
        - e.g. ZRAM, balloon driver
    - In the case of an anonymous page for users, it depends on whether the CONFIG_KSM kernel option is enabled.
      - When the KSM kernel option is not used, only the PAGE_MAPPING_ANON(1) flag is added and points to the anon mapping area, the anon_vma structure pointer.
      - If you are using the KSM kernel options, add the PAGE_MAPPING_ANON(1) and PAGE_MAPPING_MOVABLE(2) flags, and point to a pointer to a private struct for KSM.
- index
  - The offset value in the mapping area is contained.
- private
  - It contains private data used for mapping.
    - It is used for buffer_heads on private pages.
    - Swab is used for swp_entry_t of page cache.
    - The order of the buddy page is contained.

2) page_pool pages used by Netstack

- pp_magic
  - The purpose is to identify the use of network page_pool.
- *pp
  - page_pool a pointer to a struct.
- _pp_mapping_pad
  - padding is intended and is not actually used.
- dma_addr
  - DMA Address
- dma_addr_upper
  - When using a 32-bit DMA on a 64-bit system, the lower 32 bits of the DMA address are stored in the upper dma_addr, and the upper 32 bits are stored in this member.
- pp_frag_count
  - This is a frag counter managed by atomic.
  - It is not supported on 64-bit systems that use 32-bit DMA.

include/linux/mm_types.h -2/3-

```
		struct {	/* slab, slob and slub */
			union {
				struct list_head slab_list;	/* uses lru */
				struct {	/* Partial pages */
					struct page *next;
#ifdef CONFIG_64BIT
					int pages;	/* Nr of pages left */
					int pobjects;	/* Approximate count */
#else
					short int pages;
					short int pobjects;
#endif
				};
			};
			struct kmem_cache *slab_cache; /* not slob */
			/* Double-word boundary */
			void *freelist;		/* first free object */
			union {
				void *s_mem;	/* slab: first object */
				unsigned long counters;		/* SLUB */
				struct {			/* SLUB */
					unsigned inuse:16;
					unsigned objects:15;
					unsigned frozen:1;
				};
			};
		};
		struct {	/* Tail pages of compound page */
			unsigned long compound_head;	/* Bit zero is set */

			/* First tail page only */
			unsigned char compound_dtor;
			unsigned char compound_order;
			atomic_t compound_mapcount;
			unsigned int compound_nr; /* 1 << compound_order */
		};
		struct {	/* Second tail page of compound page */
			unsigned long _compound_pad_1;	/* compound_head */
			atomic_t hpage_pinned_refcount;
			/* For both global and memcg */
			struct list_head deferred_list;
		};
```

3) Slab, slob, slub pages

- slab_list
  - Used to connect to an LRU list.
- *next
  - Partial pages.
- pages
  - Contains a partial number of pages.
    - It contains the number of slub pages that are connected to the next, including itself.
- pobjects
  - Approximate number of objects

    - It's not exact, but it contains the total number of free objects for the next slub page, including my slub page.
    - This counter is often not calculated correctly due to the objects being free
- *slab_cache
  - Point to the associated slab cache.
- *freelist
  - This is a list of free objects waiting for you.
- *s_mem
  - slab:'s first object.
- counters
  - Use the following 32 bytes to access them at once.
  - inuse:16
    - The number of objects in use.
  - objects:15
    - Number of objects managed by slabs
  - frozen:1
    - It refers to whether it is a slap page that is managed by Per-CPU.

4) Compound tail pages

Note that the head page (page[0]) of the Compound page has a PG_head flag.

- compound_head
  - All tail pages that are not the header of the compound page contain the compound header page descriptor pointer, and set bit0 to 1.
- compound_dtor
  - On the first tail page (page[1]), put the compound page destructor identifier id.
  - Contains one of the following compound page destructor IDs:
    - NULL_COMPOUND_DTOR
    - COMPOUND_PAGE_DTOR
    - HUGETLB_PAGE_DTOR
    - TRANSHUGE_PAGE_DTOR
- compound_order
  - The first tail page (page[1]) contains the order of the compound page.
- compound_mapcount
  - The first tail page (page[1]) contains the number of mapping counts.

- _compound_pad_1
  - Do not use it.
- _compound_pad_2
  - Do not use it.
- deferred_list
  - It is used to detach a huge page from the second tail page (page[2]) and suspend it for a while instead of unmapping it immediately.

- Note: thp: introduce deferred_split_huge_page() (https://github.com/torvalds/linux/commit/9a982250f773cc8c76f1eee68a770b7cbf2faf78#diff-ff0276c422430aff67678952b177fe3d)

include/linux/mm_types.h -3/4-

```
                struct {        /* Page table pages */
                        unsigned long _pt_pad_1;        /* compound_head */
                        pgtable_t pmd_huge_pte; /* protected by page->ptl */
                        unsigned long _pt_pad_2;        /* mapping */
                        union {
                                struct mm_struct *pt_mm; /* x86 pgds only */
                                atomic_t pt_frag_refcount; /* powerpc */
                        };
#if ALLOC_SPLIT_PTLOCKS
                        spinlock_t *ptl;
#else
                        spinlock_t ptl;
#endif
                };
                struct {        /* ZONE_DEVICE pages */
                        /** @pgmap: Points to the hosting device page map. */
                        struct dev_pagemap *pgmap;
                        void *zone_device_data;
                        /*
                         * ZONE_DEVICE private pages are counted as being
                         * mapped so the next 3 words hold the mapping, index,
                         * and private fields from the source anonymous or
                         * page cache page while the page is migrated to device
                         * private memory.
                         * ZONE_DEVICE MEMORY_DEVICE_FS_DAX pages also
                         * use the mapping, index, and private fields when
                         * pmem backed DAX files are mapped.
                         */
                };

                /** @rcu_head: You can use this to free a page by RCU. */
                struct rcu_head rcu_head;
        };
```

5) Pages for page tables (PGD, PUD, PMD, PTE)

- _pt_pad_1
  - Do not use it.
- pmd_huge_pte
  - It contains a Page descriptor pointer that points to the PTE table for HUGE.
- _pt_pad_2
  - Do not use it.
- *pt_mm
  - x86 pgds only

- *pt_frag_refcount
  - powerpc only
- *ptl or ptl
  - The page table is a spinlock.
  - If the spinlock_t size is included in an unsigned long unit, use PTL, otherwise use *PTL to assign and point to spinlock_t.

6) Zone Devices Page

- *pgmap
  - Contains a dev_pagemap struct pointer for the zone device.
- hmm_data
  - It contains driver data for HMM device memory.
- _zd_pad_1
  - Do not use it.

include/linux/mm_types.h -4/4-

```
        union {         /* This union is 4 bytes in size. */
                /*
                 * If the page can be mapped to userspace, encodes the number
                 * of times this page is referenced by a page table.
                 */
                atomic_t _mapcount;

                /*
                 * If the page is neither PageSlab nor mappable to userspace,
                 * the value stored here may help determine what this page
                 * is used for.  See page-flags.h for a list of page types
                 * which are currently stored here.
                 */
                unsigned int page_type;

                unsigned int active;            /* SLAB */
                int units;                      /* SLOB */
        };

        /* Usage count. *DO NOT USE DIRECTLY*. See page_ref.h */
        atomic_t _refcount;

#ifdef CONFIG_MEMCG
        unsigned long memcg_data;
#endif

        /*
         * On machines where all RAM is mapped into kernel address space,
         * we can simply calculate the virtual address. On machines with
         * highmem some memory is mapped into kernel virtual memory
         * dynamically, so we need a place to store that address.
         * Note that this field could be 16 bits on x86 ... 🙂
         *
         * Architectures with slow multiplication can define
         * WANT_PAGE_VIRTUAL in asm/page.h
```

```
         */
#if defined(WANT_PAGE_VIRTUAL)
        void *virtual;                  /* Kernel virtual address (NULL
if
                                           not kmapped, ie. highmem) */
#endif /* WANT_PAGE_VIRTUAL */

#ifdef LAST_CPUPID_NOT_IN_PAGE_FLAGS
        int _last_cpupid;
#endif
} _struct_page_alignment;
```

Other-1

The _mapcount, page_type, active, and units below are defined as union types that are selected and used according to the page type.

- _mapcount
  - Mapping counts used by users
  - Each time it is mapped to multiple user spaces, the counter is incremented.
    - If a shared page is shared by multiple user processes, it will be multi-mapped, and the mapping counter will be incremented by the number.
- page_type
  - Except for the user mapping page and the slab page, the rest of the kernel pages specify the page type and store and use the following flags.
    - PG_buddy
    - PG_ballon
    - PG_kmemcg
    - PG_table
- active
  - slab: Used in the allocator.
- units
  - slob: Used in the allocator.

Other-2

- _refcount
  - The reference counter should not be accessed directly by the user.
- memcg_data
  - When using a memory cgroup on a system with CONFIG_MEMCG kernel options, it contains a obj_cgroup struct pointer that can be used, and depending on the type of allocation, it uses the following two flags:
    - MEMCG_DATA_OBJCGS
    - MEMCG_DATA_KMEM
- *virtual
  - If you use the WANT_PAGE_VIRTUAL kernel option, in some 32-bit architectures that use highmem, if the highmem page is mapped to the kernel address space, it will contain the mapped kernel virtual address.
  - In the case of ARM32, HASHED_PAGE_VIRTUAL is used.
    - When mapping a highmem page, we don't use this member because we use a separate allocation of page_address_map structs.

- 64-bit systems use highmem, so you don't need to use this member.
- _last_cpupid
  - There are times when you need to include the CPU and PID in the flags member of the page, but on a 32-bit system you can't include both in 32-bit. In this case, use the LAST_CPUPID_NOT_IN_PAGE_FLAGS kernel option to store the last used CPU and PID in this member. They are used for NUMA balancing.
  - 64-bit systems use the flags member to store the last CPU and PID information, so there is no need to use this member.

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) | Sentence C – Current post
- Memory Model -3- (Sparse Memory) (http://jake.dothome.co.kr/sparsemem/) | 문c
- Memory Model -4- (APIs) (http://jake.dothome.co.kr/mem_map_page) | 문c
- 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

- Memory: the flat, the discontiguous, and the sparse(2019) (https://lwn.net/Articles/789304/) | LWN.net
- Generic page-pool recycle facility? (2016) | Jesper Dangaard Brouer – Download pdf (http://people.netfilter.org/hawk/presentations/MM-summit2016/generic_page_pool_mm_summit2016.pdf)

---

6 thoughts to “Memory Model -2- (mem_map)”

***

2019-04-12 18:10 (http://jake.dothome.co.kr/mem_map/#comment-208486)

I don't think I understood it correctly.
When you know the page structure, how do you know the physical memory that the page structure points to, i.e. the page frame#는

RESPONSE (/MEM_MAP/?REPLYTOCOM=208486#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2019-04-15 11:25 (http://jake.dothome.co.kr/mem_map/#comment-208628)

Hello? Moon Young-il of Munc Blog.

You can use the page_to_pfn() API to get the page frame number (pfn) from a pointer to the page structure pointing to that page.
An API that does the opposite is the pfn_to_page() API.

Keep up the good work.

RESPONSE (/MEM_MAP/?REPLYTOCOM=208628#RESPOND)

****
2020-05-17 09:00 (http://jake.dothome.co.kr/mem_map/#comment-249307)

Hello good article, I am reading it carefully.
There is no MM-7B, MM-9A picture.

RESPONSE (/MEM_MAP/?REPLYTOCOM=249307#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-06-22 11:18 (http://jake.dothome.co.kr/mem_map/#comment-257080)

I've included the picture again.
Thanks for letting us know.

RESPONSE (/MEM_MAP/?REPLYTOCOM=257080#RESPOND)

****
2020-09-22 20:08 (http://jake.dothome.co.kr/mem_map/#comment-277674)

Hello. Do you have any idea how memory is initialized when Linux boots? (mem_map arrays, etc.) Even if I tried to allocate an array like that, I'd need a memory allocation policy. I'm curious how to initialize it at the very beginning.

RESPONSE (/MEM_MAP/?REPLYTOCOM=277674#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-09-24 07:00 (http://jake.dothome.co.kr/mem_map/#comment-278068)

안녕하세요?

리눅스 부팅 후 사용되는 대표적인 메모리 할당 시스템은 buddy, slab, per-cpu, cma, … 등이 있습니다.
이를 초기화하여 운용하기 전에는 memblock을 사용하여 영역 정보만을 관리하고 있습니다.

메모리 할당자를 사용하기 전에 memblock을 통해 미리 필요한 영역을 reserve하여 등록하고, 사용할 때에는 원시적으로 직접 해당 주소 영역에 직접 기록하는 방법을 사용합니다. 그런 후 추후 페이지 할당자인 buddy 시스템을 준비할 때 그 reserve 영역을 피한 나머지 메모리 영역만 buddy 시스템에 free 페이지로 등록하여 사용합니다.

감사합니다.

응답 (/MEM_MAP/?REPLYTOCOM=278068#RESPOND)

댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

< Fixmap (http://jake.dothome.co.kr/fixmap/)

페이지 테이블 API – ARM32 > (http://jake.dothome.co.kr/pt-api/)

문c 블로그 (2015 ~ 2024)