Chapter 3 : Memory Management

Choi Yong-jae
bestjae@naver.com

Overview

- Overview
  - Management of physical pages in memory.
  - The buddy system to allocate memory in large chunks.
  - The slab, slub, and slob allocators to allocate smaller chunks of memory.
  - The vmalloc mechanism to allocate non-contiguous blocks of memory.
  - The address space of processe

- Two types of machine that manage physical memory
  - UMA machines(uniform memory access)
    - Each processor (in a symmetric multiprocessor system) is able to access each memory area equally quickly
  - NUMA machines(non-uniform memory access)
    - Local RAM is available to each CPU of the system to support particularly fast access

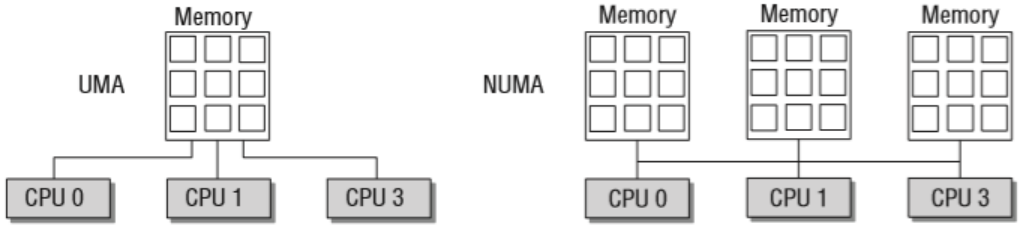
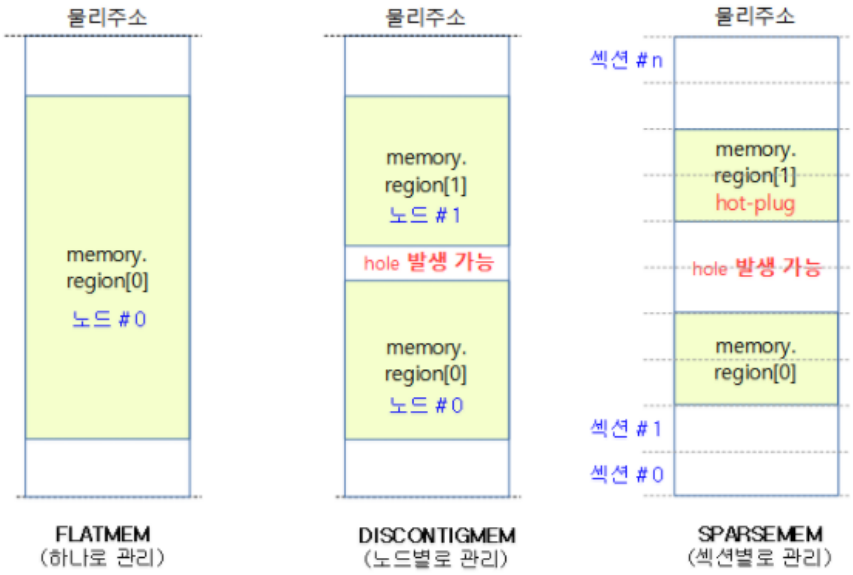


Figure 3-1: UMA and NUMA systems.

Overview

- Memory management
  - FLATMEM, DISCONTIGMEM, SPARSEMEM
  - Focuses on the UMA case
  - Real NUMA systems
    - CONFIG_NUMA

```
#ifdef CONFIG_FLATMEM
#define pfn_to_nid(pfn)      (0)
#endif

#ifdef CONFIG_SPARSEMEM

/*
 * SECTION_SHIFT              #bits spa
 *
 * PA_SECTION_SHIFT      physical addr
 * PFN_SECTION_SHIFT          pfn to/fr
 */
```



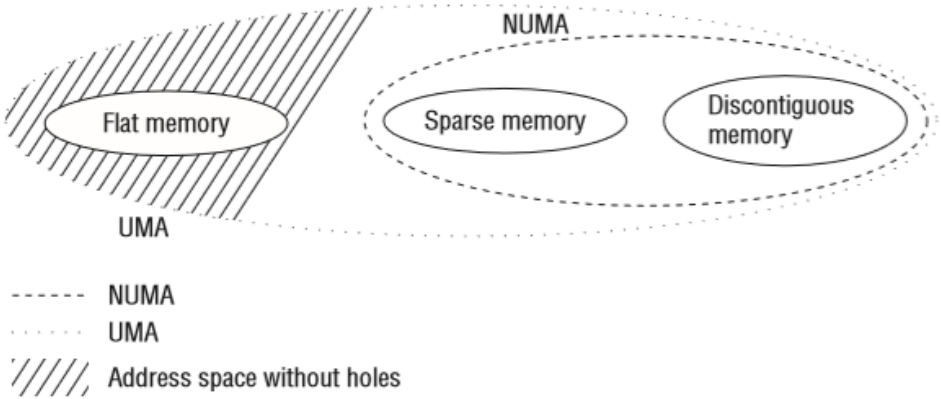


Figure 3-2: Overview of possible memory setups for flat, sparse, and discontiguous memory on UMA and NUMA machines.

Organization in the (N)UMA Model

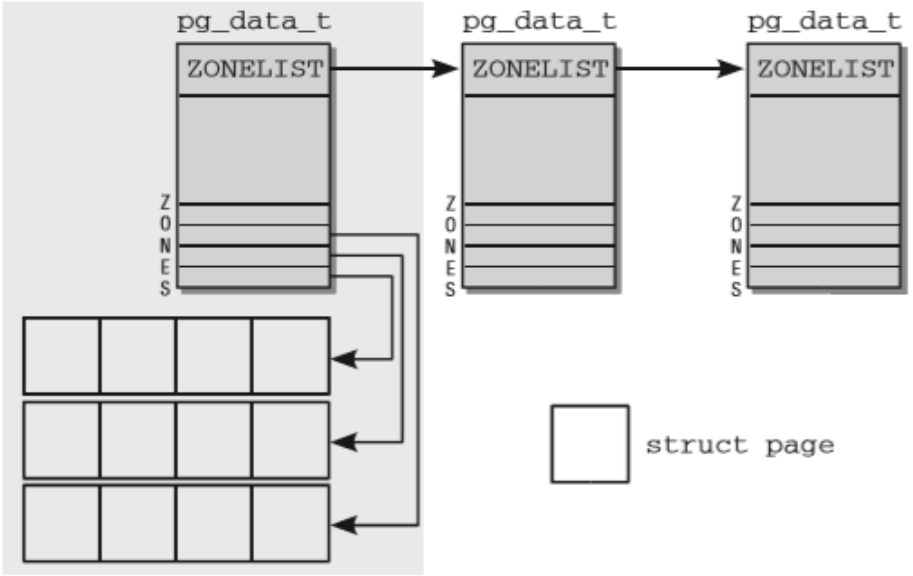


Figure 3-3: Memory partitioning in NUMA systems.

- UMA systems
  - A single NUMA node is introduced to help manage the entire system memory
  - RAM memory is divided into **nodes**
    - Nodes - Struct pg_data_t
  - Each node is split into **zones**

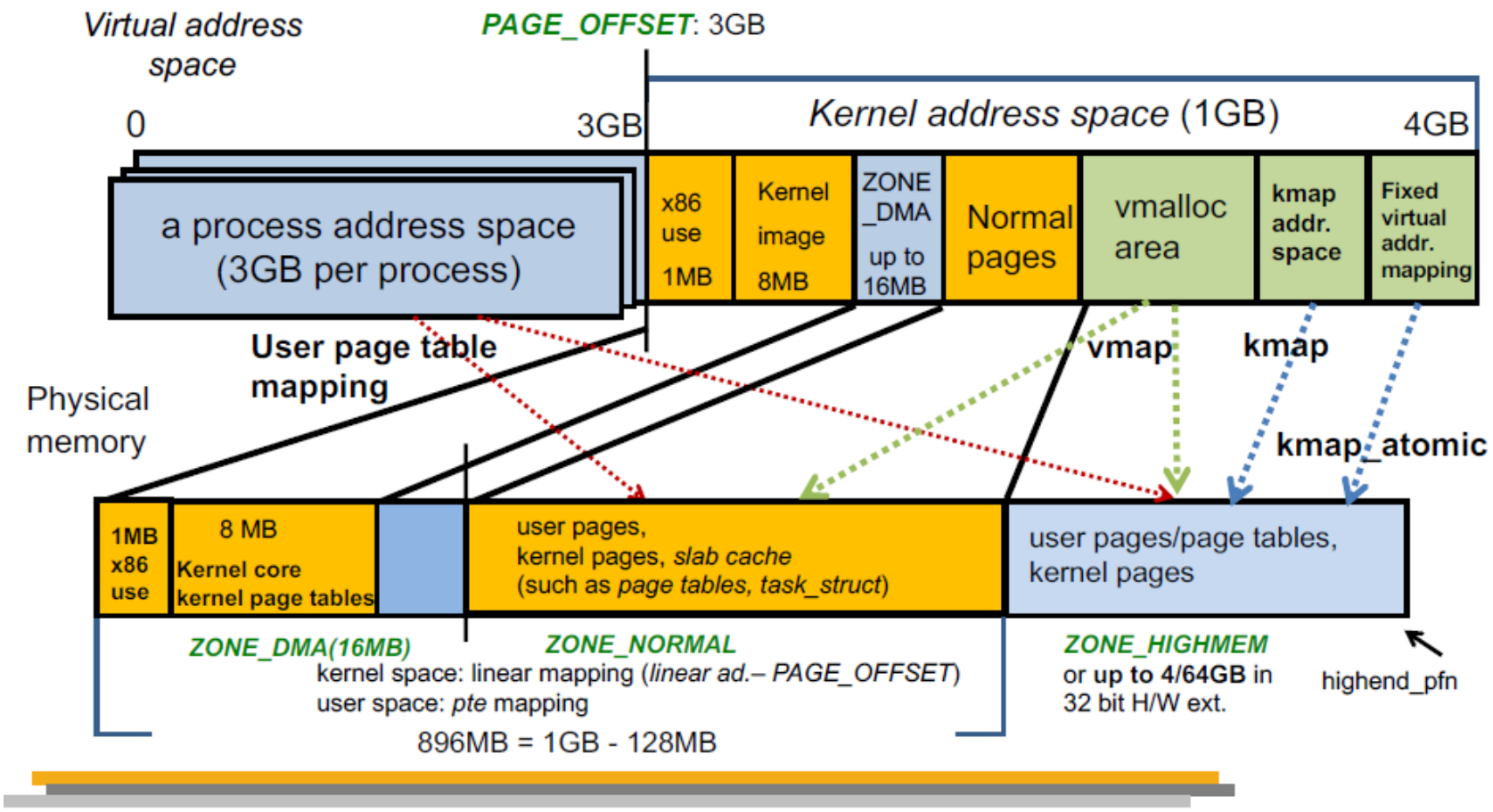
Organization in the (N)UMA Model

- Zones
  - Mmzone.h-enum zone_type
    - ZONE_DMA
    - ZONE_DMA32
    - ZONE_NORMAL
    - ZONE_HIGHMEM
  - ZONE_MOVABLE
    - Prevent fragmentation
  - Each zone is associated with an array, page frames
  - Kernel attempts memory allocations currently running CPU in NUMA system

```
enum zone_type {
#ifdef CONFIG_ZONE_DMA
    /*
     * 모든 주소지정 가능 메모리에(ZONE_NORMAL)
     * DMA를 수행할 수 없는 장치가 없을때 사용
     */
    ZONE_DMA,
#endif
#ifdef CONFIG_ZONE_DMA32
    /*
     * x86_64는 4GB 아래 DMA 영역만 수행할 수있는
     * 비트 장치도 지원하므로 DMA가 두개필요
     */
    ZONE_DMA32,
#endif
    /*
     * 정상적인 주소지정이 가능한 메모리
     * DMA장치가 모든주소 지정 가능 메모리의 전송을 지원하는경우
     * ZONE_NORMAL에서 DMA작업 수행 가능
     */
    ZONE_NORMAL,
#ifdef CONFIG_HIGHMEM
    /*
     * 자체 주소 공간에 부분을 매핑하여
     * 커널에 의해서만 주소 지정 가능한 메모리 영역
     */
    ZONE_HIGHMEM,
#endif
    ZONE_MOVABLE,
#ifdef CONFIG_ZONE_DEVICE
    ZONE_DEVICE,
#endif
    __MAX_NR_ZONES

};
```

Organization in the (N)UMA Model

- Virtual & Physical Map (32bit machine)



Organization in the (N)UMA Model

- Node Management

```
typedef struct pglist_data {
    // 노드에서 영역의 데이터 구조를 보유하는 배열
    struct zone node_zones[MAX_NR_ZONES];
    //현재 존에서 사용가능한 공간이 없을때 할당에 사용되는 대체노드존
    struct zonelist node_zonelists[MAX_ZONELISTS];
    int nr_zones; //노드의 다른 영역 수
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    // physical 메모리 페이지 인스턴스 배열 포인터
    struct page *node_mem_map;
#ifdef CONFIG_PAGE_EXTENSION
    struct page_ext *node_page_ext;
#endif
#endif
#ifndef CONFIG_NO_BOOTMEM
    struct bootmem_data *bdata; // 부팅 메모리 할당기 활용 -3.4.3
#endif
```

Organization in the (N)UMA Model

- Node Management

```
#ifdef CONFIG_MEMORY_HOTPLUG
    /*
     * Must be held any time you expect node_start_pfn, node_present_pages
     * or node_spanned_pages stay constant.  Holding this will also
     * guarantee that any pfn_valid() stays that way.
     *
     * pgdat_resize_lock() and pgdat_resize_unlock() are provided to
     * manipulate node_size_lock without checking for CONFIG_MEMORY_HOTPLUG.
     *
     * Nests above zone->lock and zone->span_seqlock
     */
    spinlock_t node_size_lock;
#endif
    unsigned long node_start_pfn; // 노드의 첫번째 페이지 프레임
    unsigned long node_present_pages; /* total number of physical pages */
    unsigned long node_spanned_pages; /* total size of physical page
                         range, including holes */
    int node_id;    // Global 노드 식별자
    wait_queue_head_t kswapd_wait; // 스왑 데몬 작업구조
    wait_queue_head_t pfmemalloc_wait;
    struct task_struct *kswapd; /* Protected by
                       mem_hotplug_begin/end() */
    int kswapd_order;
    enum zone_type kswapd_classzone_idx;
```

Organization in the (N)UMA Model

- Node Management

```
#ifdef CONFIG_NUMA_BALANCING
    // 누마 밸런싱을 위해
    /* Lock serializing the migrate rate limiting window */
    spinlock_t numabalancing_migrate_lock;

    /* Rate limiting time interval */
    unsigned long numabalancing_migrate_next_window;

    /* Number of pages migrated during the rate limiting time interval */
    // 마이그레이션 페이지 수
    unsigned long numabalancing_migrate_nr_pages;
#endif
    /*
     * This is a per-node reserve of pages that are not available
     * to userspace allocations.
     */
    unsigned long       totalreserve_pages;

#ifdef CONFIG_NUMA
    /*
     * zone reclaim becomes active if more unmapped pages exist.
     */
    unsigned long       min_unmapped_pages;
    unsigned long       min_slab_pages;
```

Organization in the (N)UMA Model

- Node Management

```
    //page 회수를 위한 lru 정책
    /* Fields commonly accessed by the page reclaim scanner */
    struct lruvec       lruvec;

    /*
     * The target ratio of ACTIVE_ANON to INACTIVE_ANON pages on
     * this node's LRU.  Maintained by the pageout code.
     */
    unsigned int inactive_ratio;

    unsigned long       flags;

    ZONE_PADDING(_pad2_)

    /* Per-node vmstats */
    struct per_cpu_nodestat __percpu *per_cpu_nodestats;
    atomic_long_t       vm_stat[NR_VM_NODE_STAT_ITEMS];
} pg_data_t;
```

```
/*
 * zone->lock and the zone lru_lock are two of the hottest locks in the kernel.
 * So add a wild amount of padding here to ensure that they fall into separate
 * cachelines.  There are very few zone structures in the machine, so space
 * consumption is not a concern here.
 */
#if defined(CONFIG_SMP)
struct zone_padding {
    char x[0];
} ____cacheline_internodealigned_in_smp;
```

Organization in the (N)UMA Model

- Node State Management
  - the kernel keeps a bitmap that provides state information for each node

```
enum node_states {
    N_POSSIBLE,     /* The node could become online at some point */
    N_ONLINE,       /* The node is online */
    N_NORMAL_MEMORY,    /* The node has regular memory */
#ifdef CONFIG_HIGHMEM
    N_HIGH_MEMORY,      /* The node has regular or high memory */
#else
    N_HIGH_MEMORY = N_NORMAL_MEMORY,
#endif
#ifdef CONFIG_MOVABLE_NODE
    N_MEMORY,       /* The node has memory(regular, high, movable) */
#else
    N_MEMORY = N_HIGH_MEMORY,
#endif
    N_CPU,      /* The node has one or more cpus */
    NR_NODE_STATES
};
```

Organization in the (N)UMA Model

- Memory Zones
  - Several sections separated by ZONE_PADDING
    - zone structures are very frequently accesse
  - Different CPUs try to access structure elements at the same time.
    - Using Lock - prevent them interfering with each, and giving rise to errors and inconsistencies

```
struct zone {
        /* Fields commonly accessed by the page allocator */
        unsigned long           pages_min, pages_low, pages_high;

        unsigned long           lowmem_reserve[MAX_NR_ZONES];

        struct per_cpu_pageset  pageset[NR_CPUS];

        /*
         * free areas of different sizes
         */
        spinlock_t              lock;
        struct free_area        free_area[MAX_ORDER];

        ZONE_PADDING(_pad1_)

        /* Fields commonly accessed by the page reclaim scanner *
        spinlock_t              lru_lock;
        struct list_head        active_list;
        struct list_head        inactive_list;
        unsigned long           nr_scan_active;
        unsigned long           nr_scan_inactive;
        unsigned long           pages_scanned;     /* since last
        unsigned long           flags;             /* zone flags, see below */

        /* Zone statistics */
        atomic_long_t           vm_stat[NR_VM_ZONE_STAT_ITEMS];

        int prev_priority;

        ZONE_PADDING(_pad2_)
        /* Rarely used or read-mostly fields */

        wait_queue_head_t       * wait_table;
        unsigned long           wait_table_hash_nr_entries;
        unsigned long           wait_table_bits;

        /* Discontig memory support fields. */
        struct pglist_data      *zone_pgdat;
        unsigned long           zone_start_pfn;

        unsigned long           spanned_pages;  /* total size, including holes */
        unsigned long           present_pages;  /* amount of memory (excluding hol

        /*
         * rarely used fields:
         */
        char                    *name;
} ____cacheline_maxaligned_in_smp;
```

Organization in the (N)UMA Model

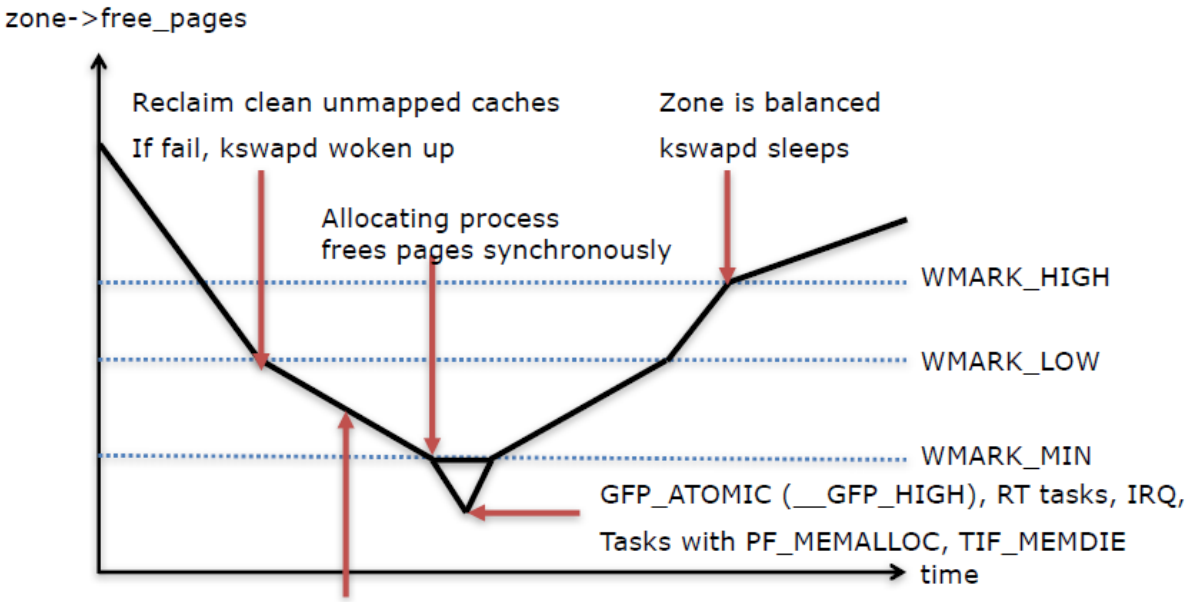
- Memory Zones
  - Several sections separated by ZONE_PADDING
    - zone structures are very frequently accesse
  - Different CPUs try to access structure elements at the same time.
    - Using Lock - prevent them interfering with each, and giving rise to errors and inconsistencies

```
struct zone {
        /* Fields commonly accessed by the page allocator */
        /* zone watermarks, access with *_wmark_pages(zone) macros */
        unsigned long watermark[NR_WMARK];
        unsigned long           lowmem_reserve[MAX_NR_ZONES];

    struct per_cpu_pageset __percpu *pageset; // for page Hot-n-Cold

        /*
         * free areas of different sizes
         */
        spinlock_t              lock;
        struct free_area        free_area[MAX_ORDER];

        ZONE_PADDING(_pad1_)

        /* Fields commonly accessed by the page reclaim scanner *
        spinlock_t              lru_lock;
        struct list_head        active_list;
        struct list_head        inactive_list;
        unsigned long           nr_scan_active;
        unsigned long           nr_scan_inactive;
        unsigned long           pages_scanned;     /* since last
        unsigned long           flags;             /* zone flags, see below */

        /* Zone statistics */
        atomic_long_t           vm_stat[NR_VM_ZONE_STAT_ITEMS];

        int prev_priority;

        ZONE_PADDING(_pad2_)
        /* Rarely used or read-mostly fields */

        wait_queue_head_t       * wait_table;
        unsigned long           wait_table_hash_nr_entries;
        unsigned long           wait_table_bits;

        /* Discontig memory support fields. */
        struct pglist_data      *zone_pgdat;
        unsigned long           zone_start_pfn;

        unsigned long           spanned_pages;  /* total size, including holes */
        unsigned long           present_pages;  /* amount of memory (excluding ho

      ZONE_PADDING( pad3 )
      /* Zone statistics */
      atomic_long_t       vm_stat[NR_VM_ZONE_STAT_ITEMS];
     */
        char                    *name;
} ____cacheline_maxaligned_in_smp;
```

Organization in the (N)UMA Model

- Zone Watermark
  - WMARK_MIN, WMARK_LOW, WMARK_HIGH

Watermark level	*Lowmem (DMA, Normal)*	*Highmem*
WMARK_MIN	*min_free_kbytes* * *zone_pages* / lowmem_pages	zone_pages / 1024 (32~128 pages)
WMARK_LOW	1.25 * min	*1.25 * min*
WMARK_HIGH:	1.5 * *min*	*1.5 * min*



Organization in the (N)UMA Model

- Calculation of Zone Watermark
  - The size of the available RAM. It is stored in the global variable min_free_kbytes
  - Filling the watermarks in the data structure is handled by init_per_zone_pages_min()

mm/page_alloc.c

```
void setup_per_zone_pages_min(void)
{
        unsigned long pages_min = min_free_kbytes >> (PAGE_SHIFT - 10);
        unsigned long lowmem_pages = 0;
        struct zone *zone;
        unsigned long flags;
...
        for_each_zone(zone) {
                u64 tmp;

                tmp = (u64)pages_min * zone->present_pages;
                do_div(tmp,lowmem_pages);
                if (is_highmem(zone)) {
                        int min_pages;

                        min_pages = zone->present_pages / 1024;
                        if (min_pages < SWAP_CLUSTER_MAX)
                                min_pages = SWAP_CLUSTER_MAX;
                        if (min_pages > 128)
                                min_pages = 128;
                        zone->pages_min = min_pages;
                } else {
                        zone->pages_min = tmp;
                }

                zone->pages_low = zone->pages_min + (tmp >> 2);
                zone->pages_high = zone->pages_min + (tmp >> 1);
        }
}
```

```
static void __setup_per_zone_wmarks(void)
{
    unsigned long pages_min = min_free_kbytes >> (PAGE_SHIFT - 10);
    unsigned long lowmem_pages = 0;
    struct zone *zone;
    unsigned long flags;

    /* Calculate total number of !ZONE_HIGHMEM pages */
    for_each_zone(zone) {
        if (!is_highmem(zone))
            lowmem_pages += zone->managed_pages;
    }

    for_each_zone(zone) {
        u64 tmp;

        spin_lock_irqsave(&zone->lock, flags);
        tmp = (u64)pages_min * zone->managed_pages;
        do_div(tmp, lowmem_pages);
        if (is_highmem(zone)) {
```

Organization in the (N)UMA Model

- Hot-N-Cold Pages
  - The pageset element of struct zone is used to implement a hot-n-cold allocator
  - The useful data are held in per_cpu_pages

```
struct per_cpu_pages {
    int count;      /* number of pages in the list */
    int high;       /* high watermark, emptying needed */
    int batch;      /* chunk size for buddy add/remove */

    /* Lists of pages, one per migrate type stored on the pcp-lists */
    struct list_head lists[MIGRATE_PCPTYPES];
};
```

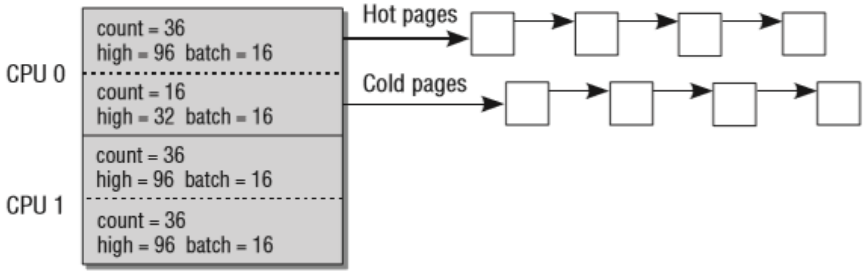


Figure 3-6: Per-CPU cache on a dual-processor system.

<mmzone.h>

```
struct per_cpu_pageset {
        struct per_cpu_pages pcp[2];    /* 0: hot.  1: cold */
} ____cacheline_aligned_in_smp;
```

```
struct per_cpu_pageset {
    struct per_cpu_pages pcp;
#ifdef CONFIG_NUMA
    s8 expire;
#endif
#ifdef CONFIG_SMP
    s8 stat_threshold;
    s8 vm_stat_diff[NR_VM_ZONE_STAT_ITEMS];
#endif
};
```

Organization in the (N)UMA Model

- Page Frames
  - The smallest unit of system memory
  - Keep struct page as small as possible
    - the memory of systems even with a moderate RAM configuration is broken down into a very large number of page

<mm_types.h>

```
struct page {
...
        union {
                atomic_t _mapcount; /* Count of ptes mapped in mms,
                                     * to show when page is mapped
                                     * & limit reverse map searches.
                                     */
                unsigned int inuse; /* SLUB: Nr of objects */
        };
...
}
```

```
struct {

    union {
        /*
         * Count of ptes mapped in mms, to show when
         * page is mapped & limit reverse map searches.
         *
         * Extra information about page type may be
         * stored here for pages that are never mapped,
         * in which case the value MUST BE <= -2.
         * See page-flags.h for more details.
         */
        atomic_t _mapcount;

        unsigned int active;        /* SLAB */
        struct {                    /* SLUB */
            unsigned inuse:16;
            unsigned objects:15;
            unsigned frozen:1;
        };
        int units;                  /* SLOB */
    };
    /*
     * Usage count, *USE WRAPPER FUNCTION* when manual
     * accounting. See page_ref.h
     */
    atomic_t _refcount;
};
```

Organization in the (N)UMA Model

- Definition of page

```
struct page {
    /* First double word block */
    unsigned long flags;        /* Atomic flags, some possibly
                                 * updated asynchronously */
    union {
        //페 이 지  프 레 임 이  위 치 한  주 소 공 간
        struct address_space *mapping;  /* If low bit clear, points to
                                         * inode address_space, or NULL.
                                         * If page mapped as anonymous
                                         * memory, low bit is set, and
                                         * it points to anon_vma object:
                                         * see PAGE_MAPPING_ANON below.
                                         */
        void *s_mem;                    /* slab first object */
        atomic_t compound_mapcount; /* first tail page */
        /* page_deferred_list().next     -- second tail page */
    };

    /* Second double word */
    union {
        //매 핑  내 의 오 프 셋
        pgoff_t index;          /* Our offset within mapping. */
        void *freelist;         /* sl[aou]b first free object */
        /* page_deferred_list().prev    -- second tail page */
    };
```

Page Tables

- Page tables
  - Hierarchically linked page tables
    - To support the rapid and efficient management of large address spaces
  - To establish an association between the virtual address spaces of user processes and the physical memory of the system (RAM, page frames)
  - To make a uniform virtual address space available to each process

- Page table management
  - Architecture-dependent
    - all data structures and almost all functions to manipulate them are defined in architecture-specific files
  - Architecture-independent

Page Tables

- Data structures
  - Addresses in virtual memory are split into five parts as required by the structure of the four-level page tables
  - The length but also the way in which the address is split are different on the individual architectures

```
Cscope 태그 : PAGE_SHIFT
   #   줄   파일 이름 / 콘텍스트 / 줄
   1     8  arch/alpha/include/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 13
   2    16  arch/arc/include/uapi/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 14
   3    18  arch/arc/include/uapi/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 12
   4    27  arch/arc/include/uapi/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 13
   5   210  arch/arm/include/asm/assembler.h <<PAGE_SHIFT>>
              mov \rd, \rd, lsl #THREAD_SIZE_ORDER + PAGE_SHIFT
   6    14  arch/arm/include/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 12
   7    26  arch/arm64/include/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT CONFIG_ARM64_PAGE_SHIFT
   8    14  arch/avr32/include/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 12
   9   109  arch/avr32/kernel/entry-avr32b.S <<PAGE_SHIFT>>
              bfextu r1, r0, PAGE_SHIFT, PGDIR_SHIFT - PAGE_SHIFT
  10    21  arch/avr32/mm/copy_page.S <<PAGE_SHIFT>>
              sub r10, r11, -(1 << PAGE_SHIFT)
  11     8  arch/cris/include/asm/page.h <<PAGE_SHIFT>>
              #define PAGE_SHIFT 13
```

Page Tables

- Data structures
  - PGD, PUD, PMD, PTE
    - Page (Global, Upper, Middle) Directory
    - Page Table Entry
- Format of Page Tables
  - Pgd_t, pud_t, pmd_t, pte_t
  - pdg_val, pdg_index, pdg_present, pdg_none, pdg_clear, pdg_bad, pte_page
  - Pmd_offset, PAGE_ALIGN

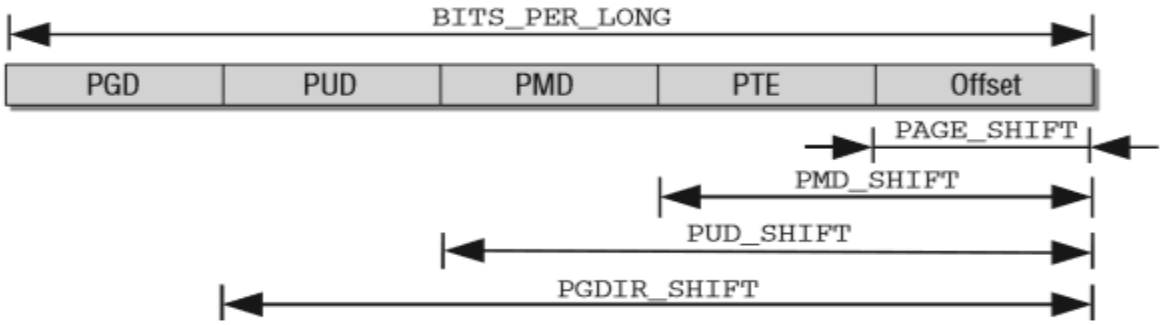


Figure 3-7: Breakdown of a virtual address.

Page Tables

- PTE-Specific Entries
  - Each final entry in the page table yields a pointer to the memory location of the page
  - holds additional information on the page
  - _PAGE_PRESENT, _PAGE_ACCESSED, _PAGE_DIRTY,_PAGE_DIRTY

Table 3-3: Functions for Processing the Architecture-Dependent State of a Memory Page

Function	Description
pte_present	Is the page present?
pte_read	May the page be read from within userspace?
pte_write	May the page be written to?
pte_exec	May the data in the page be executed as binary code?
pte_dirty	Is the page dirty; that is, have its contents been modified?
pte_file	Does the PTE belong to a nonlinear mapping?
pte_young	Is the access bit (typically _PAGE_ACCESS) set?
pte_rdprotect	Removes read permission for the page.
pte_wrprotect	Deletes write permission for the page.