KELP 오렐리 디바이스 드라이버 소모임

Slab allocator & Buddy System

2007.12.21

김기오 (www.asmlove.co.kr)

슬랩 관리자의 기본 개념과 핵심 코드

버디 시스템의 기본 개념과 핵심 코드

- linux2.4 코드 중심으로

슬랩 할당자
Slab Allocator

슬랩 할당자
Slab Allocator

기본 개념

기본 개념

- 페이지 단위 메모리 관리에서 내부 단편화 문제
  - 4096byte의 페이지 크기
  - 4000byte 할당
  - 96byte는?
- 슬랩 할당자 이전 방식
  - 20~40%의 내부 단편화 생김
    - refer to Uresh Vahalia, UNIX Internals)
- 슬랩 할당자의 오버헤드 < 내부 단편화
  - 슬랩&캐시 디스크립터
  - 하드웨어 캐시 정렬을 위한 단편화
  - 페이지 내부에서 슬랩 크기보다 작은 내부 단편화

기본 개념

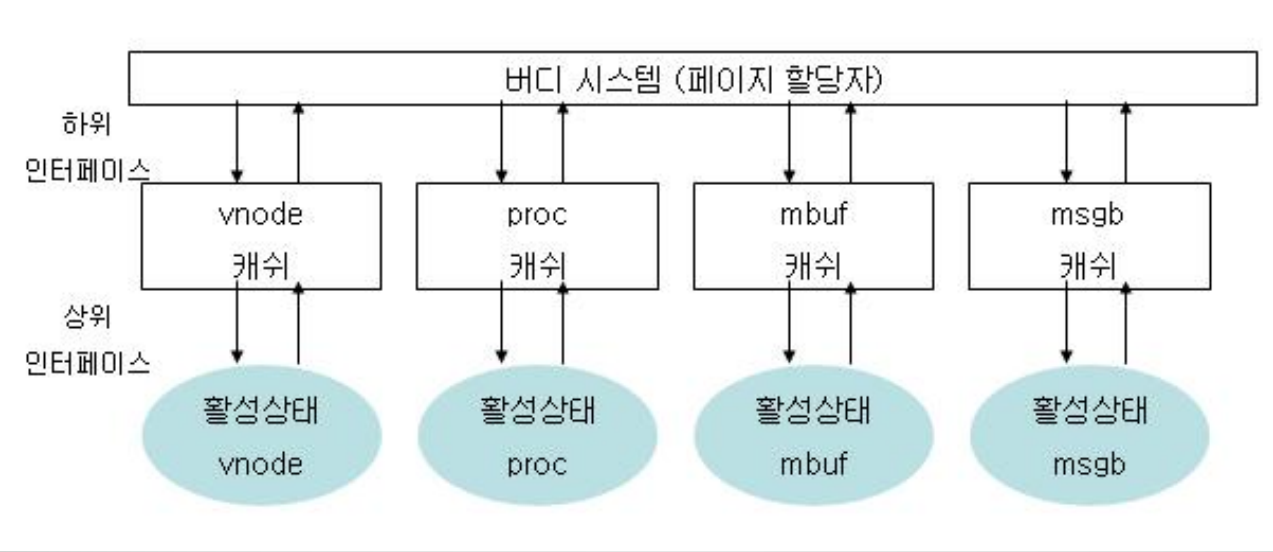
- Jeff Bonwick (Sun Micro. Inc.)의 논문
  - The Slab Allocator: An Object Caching Kernel Memory Allocator, 1994
  - 내부 단편화 감소
  - 메모리 할당 속도 증가
    - 메모리 할당/해제 오버헤드 줄임
    - 객체 생성/파괴 오버헤드 줄임
  - 하드웨어 캐시 활용 극대화

동적인 메모리 할당을 줄이고
미리 준비한 메모리를 반환하기 작전!

기본 개념

- 기본 정의
  - cache
    - 동일한 타입을 가진 객체들의 저장 장소
    - dentry cache 와 같이 저장되는 데이터 구조체와 같은 이름을 가짐
    - 슬랩 할당자에서 사용되는 기본적인 저장 단위
  - slab
    - 하나 이상의 페이지로 이루어진 객체들의 집합을 관리
    - 하나의 캐시는 여러 개의 슬랩으로 이루어짐
  - object
    - 슬랩 안에 저장되는 가장 작은 단위
    - dentry 와 같이 실제적으로 메모리 할당되는 메모리 조각

- 버디 시스템과 함께 사용됨
  - 페이지 단위 메모리 관리는 버디 시스템
  - 슬랩은 할당된 페이지에 작은 캐시를 저장해서 사용



기본 개념

기본 개념

- 하나의 캐시에 3가지 슬랩 연결 가능
  - 최초로 캐시가 생기면 slabs_free 연결
  - slabs_free, slabs_partial, slabs_full 연결 가능

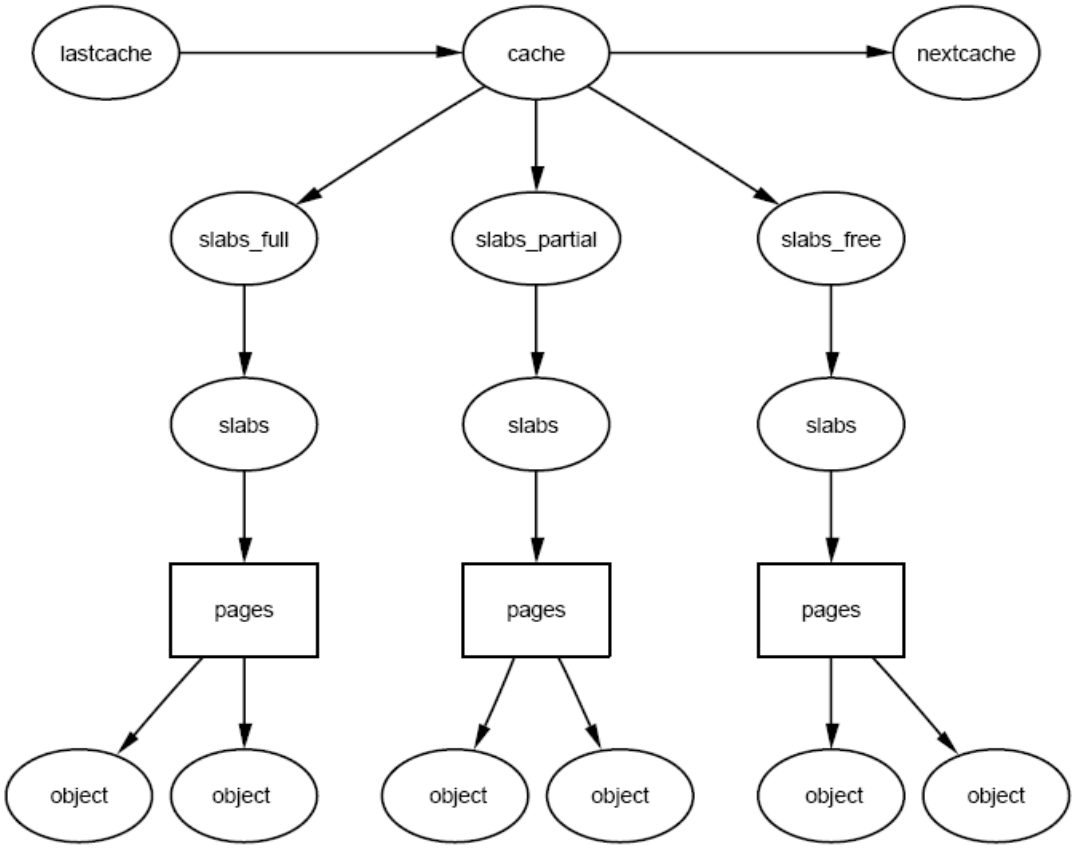
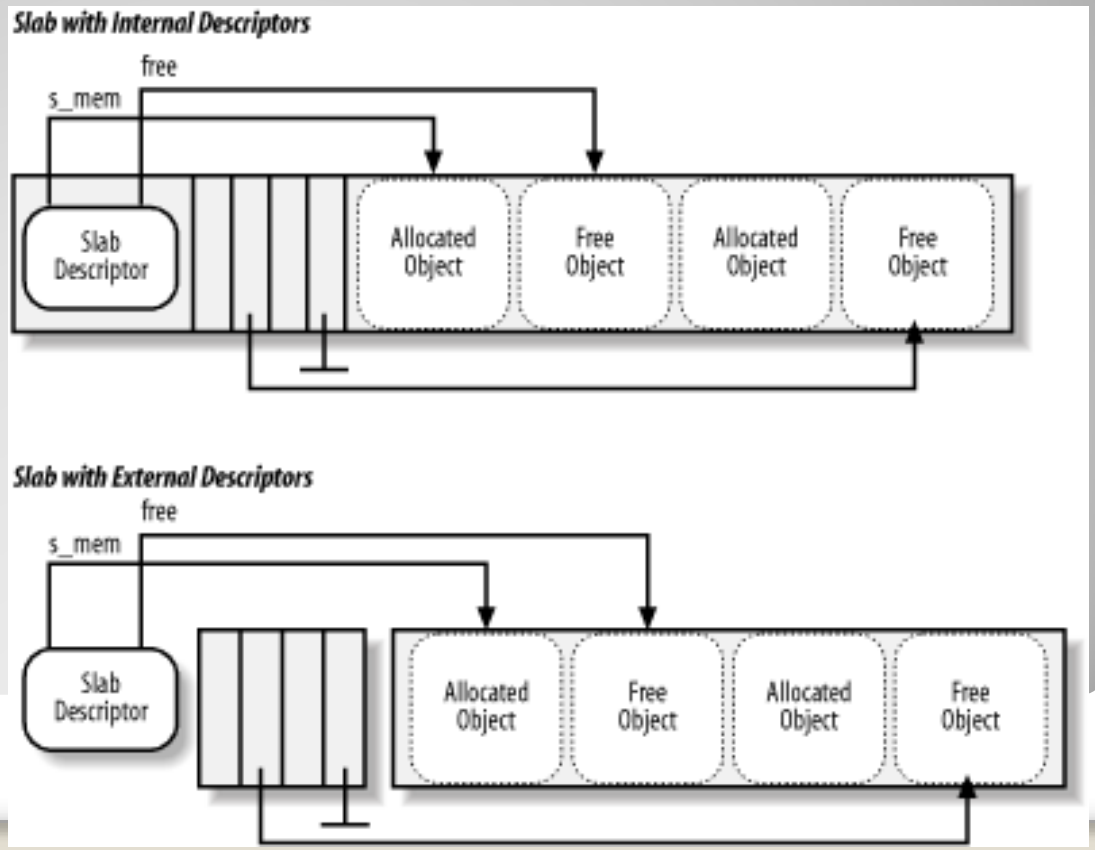


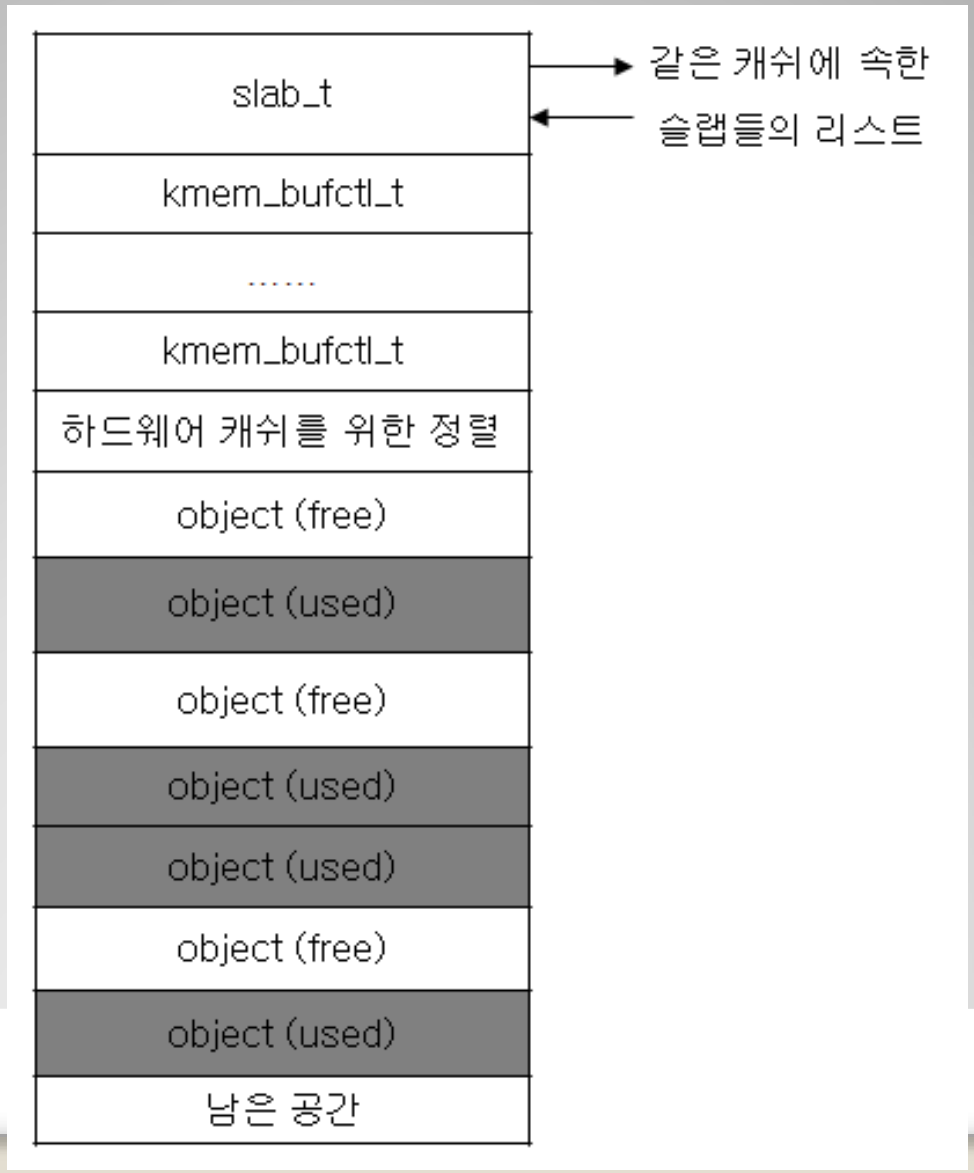
Figure 3.1: Cache Structure for the Slab Allocator

- 슬랩 디스크립터의 저장 방식
  - 슬랩 디스크립터가 객체와 같은 페이지
    - 객체가 작아서 페이지 내부 공간이 남을 때
  - 슬랩 디스크립터가 객체와 다른 페이지
    - 객체가 커서 페이지 내부 공간 부족



기본 개념

- 기본 구조


slab_t
같은 캐쉬에 속한
슬랩들의 리스트
kmem_bufctl_t
.....
kmem_bufctl_t
하드웨어 캐쉬를 위한 정렬
object (free)
object (used)
object (free)
object (used)
object (used)
object (free)
object (used)
남은 공간

기본 개념

- /proc/slabinfo
  - $ man slabinfo 매뉴얼 페이지
  - cache-name: 캐시 이름
  - num-active-objs: 사용중인 객체 수
  - total-objs: 총 객체 수
  - num-active-slabs: 사용중 객체를 하나라도 가지고 있는 슬랩의 수
  - total-slabs: 총 슬랩 수
  - num-pages-per-slab: 슬랩 한 개에 필요한 페이지 수
  - etc…

```
% cat /proc/slabinfo
slabinfo - version: 1.1
kmem_cache            60      78    100    2    2    1
blkdev_requests     5120    5120     96  128  128    1
mnt_cache             20      40     96    1    1    1
inode_cache         7005   14792    480 1598 1849    1
dentry_cache        5469    5880    128  183  196    1
filp                 726     760     96   19   19    1
buffer_head        67131   71240     96 1776 1781    1
vm_area_struct      1204    1652     64   23   28    1
...
size-8192              1      17   8192    1   17    2
size-4096             41      73   4096   41   73    1
...
```

- kmem_cache
  - kmem_cache의 구조체의 크기 = 100bytes
  - 차지하는 메모리 공간 100 x 78 = 2pages
  - 한 개의 슬랩에 필요한 페이지 = 1page
  - 78개의 객체 = 2pages = 2slabs
  - 2개의 슬랩 모두 사용중인 객체를 가짐

```
% cat /proc/slabinfo
slabinfo - version: 1.1
kmem_cache            60      78    100    2    2    1
blkdev_requests     5120    5120     96  128  128    1
mnt_cache             20      40     96    1    1    1
inode_cache         7005   14792    480 1598 1849    1
dentry_cache        5469    5880    128  183  196    1
filp                 726     760     96   19   19    1
buffer_head        67131   71240     96 1776 1781    1
vm_area_struct      1204    1652     64   23   28    1
...
size-8192              1      17   8192    1   17    2
size-4096             41      73   4096   41   73    1
...
```

기본 개념

슬랩 할당자
Slab Allocator

데이터 구조

데이터 구조

- struct kmem_cache
  - 캐시의 디스크립터
  - /mm/slab.c
  - name: 이름
  - num: 한 슬랩에 들어가는 객체의 수
  - objsize: 객체의 크기
  - gfporder: 한 슬랩의 페이지 개수 (log값)
  - next: 캐시 디스크립터의 이중 연결 리스트
  - struct kmem_list3
    - slabs_partial, slabs_full, slabs_free 연결 리스트
    - free_objects: 여유 객체의 개수
  - slabp_cache: 외부 슬랩 디스크립터의 포인터

데이터 구조

- struct slab
  - 슬랩 디스크립터
  - /mm/slab.c
  - list: slabs_full, slabs_partial, slabs_free에서 같은 종류 끼리의 리스트
  - inuse: 현재 슬랩에서 사용중인 객체의 개수
  - free: 첫번째 여유 객체의 번호
  - s_mem: 첫번째 객체의 포인터 (사용중or여유)

데이터 구조

- struct kmem_bufctl_t
  - typedef unsigned int kmem_bufctl_t;
  - 슬랩에 저장된 객체 수만큼 배열로 존재
  - 배열의 시작 주소를 계산하는 매크로

```
kmem_bufctl_t *slab_bufctl(struct slab *slabp)
{
return (kmem_bufctl_t *)(slabp+1);
}
```

  - 항상 다음 여유 객체의 인덱스가 저장됨

- kmem_bufctl_t 의 초기화
  - slab->free = 0
    - 첫번째 여유 객체의 인덱스는 0
  - kmem_bufctl_t[0] = 1
    - 첫번째 여유 객체인 0번 객체 다음의 여유 객체는 1번 객체
  - … kmem_bufctl_t[i] = i+1
  - kmem_bufctl_t[cache->num] = BUFCTL_END
    - 더 이상의 객체가 없음

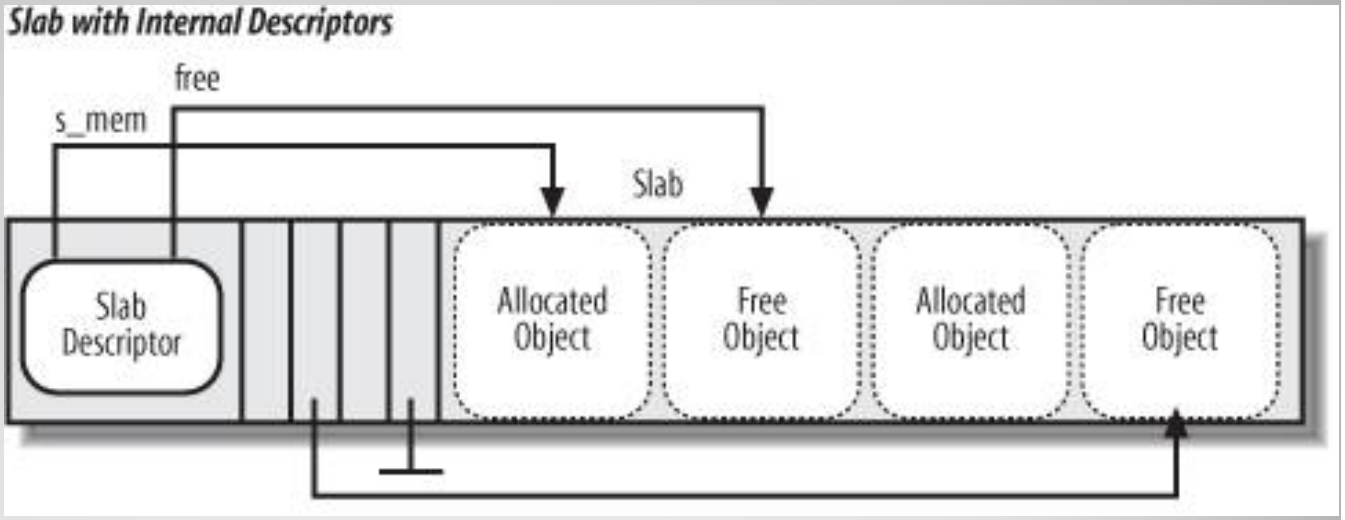
데이터 구조

데이터 구조

- 새 객체 할당 과정
  - 사용중 객체 개수 카운터 증가
  - free 번째 객체를 할당해줌
  - free 번째 kmem_bufctl_t 에서 다음 여유 객체의 번호 읽기
  - free 번째 kmem_bufctl_t 에 사용중 표시
  - free 값을 다음 여유 객체의 번호로 바꿈
  - linux 2.4.19 – kmem_cache_alloc 코드 일부

```
slabp->inuse++
next = slab_bufctl(slabp)[slabp->free];
slab_bufctl(slabp)[slabp->free] = BUFCTL_FREE;
slabp->free = next;
```

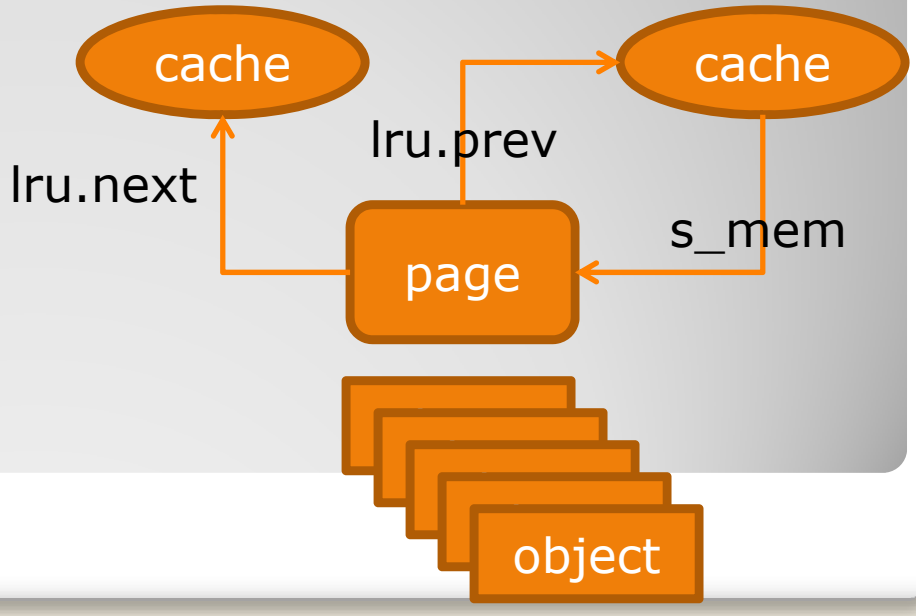
- kmem_bufctl_t 사용 예
  - 새 객체 할당 전 free == 2
  - 현재 할당해 줄 여유 객체의 번호 => 2
  - slab_bufctl(slabp)[2] == 4
  - 다음 여유 객체 번호 갱신 slabp->free = 4



데이터 구조

데이터 구조

- page 구조체와 캐시
  - page.lru : 버디 시스템에서 사용되는 리스트
  - 페이지가 캐시에 할당되면 lru는 사용되지 않음
  - lru에 페이지에 저장된 캐시 디스크립터의 포인터 저장
  - lru.next : 캐시 디스크립터 주소
  - lru.prev : 슬랩 디스크립터 주소



- 페이지 디스크립터와 캐시/슬랩 디스크립터의 관계
  - static inline void page_set_cache(struct page *page, struct kmem_cache *cache)
  - static inline struct kmem_cache *page_get_cache(struct page *page)
  - static inline void page_set_slab(struct page *page, struct slab *slab)
  - static inline struct slab *page_get_slab(struct page *page)

데이터 구조

- General caches
  - kmalloc()에서 일반적인 메모리 할당에 사용됨
  - 특정 객체의 캐시가 아님
  - kmalloc(10) -> 32바이트가 할당됨

```
/* Size description struct for
   general caches. */
typedef struct cache_sizes {
        size_t                  cs_size;
        kmem_cache_t
        *cs_cachep;
        kmem_cache_t
        *cs_dmacachep;
} cache_sizes_t;
```

데이터 구조

```
 static cache_sizes_t cache_sizes[] = {
#if PAGE_SIZE == 4096
        {   32,     NULL, NULL},
#endif
        {   64,     NULL, NULL},
        {  128,     NULL, NULL},
        {  256,     NULL, NULL},
        {  512,     NULL, NULL},
        { 1024,     NULL, NULL},
        { 2048,     NULL, NULL},
        { 4096,     NULL, NULL},
        { 8192,     NULL, NULL},
        {16384,     NULL, NULL},
        {32768,     NULL, NULL},
        {65536,     NULL, NULL},
        {131072,    NULL, NULL},
        {   0,      NULL, NULL}
};
```

cs_size 메모리 블록의 크기
cs_cachep 일반 메모리 블록을 위한 캐쉬
ca_dmacachep DMA용 메모리 블록을 위한 캐쉬

슬랩 할당자
Slab Allocator

함수 코드

함수 코드

- 주요 함수의 코드 이해
  - kmem_cache_create
  - kmem_cache_alloc
  - kmalloc/kfree
  - 부록 문서에~

함수 코드

- 주요 함수의 코드 이해
  - kmem_getpages()
  - kmem_freepages()
  - 부록 문서에~

- slab? slob?
  - /mm 디렉토리에 slab.c와 slob.c가 있음
  - 임베디드 환경에서 slab을 사용하지 않도록 커널 옵션을 설정하면 slob을 사용하게 됨
  - 고전적인 K&R/UNIX 할당자를 사용함
  - 슬랩 할당자와 동일한 인터페이스
  - 좀더 작은 코드와 적은 메모리 소비
  - 슬랩 할당자보다는 단편화가 많으므로 소형 시스템에만 사용됨

잠깐 쉬어가요~

버디 시스템
Buddy System

버디 시스템
Buddy System

기본 개념

기본 개념

- 외부 단편화
  - 페이지 단위 메모리 관리 시스템의 부작용
  - 중간중간 빈 페이지가 흩어져 있어서 연속된 페이지 할당 불가
  - 페이지를 2의 배수로 묶어서 덩어리로 관리
  - 연속된 페이지 쌍이 생길 때 마다 더 큰 덩어리로 뭉치기
  - 이런 페이지 덩어리의 쌍이 Buddy(단짝?)

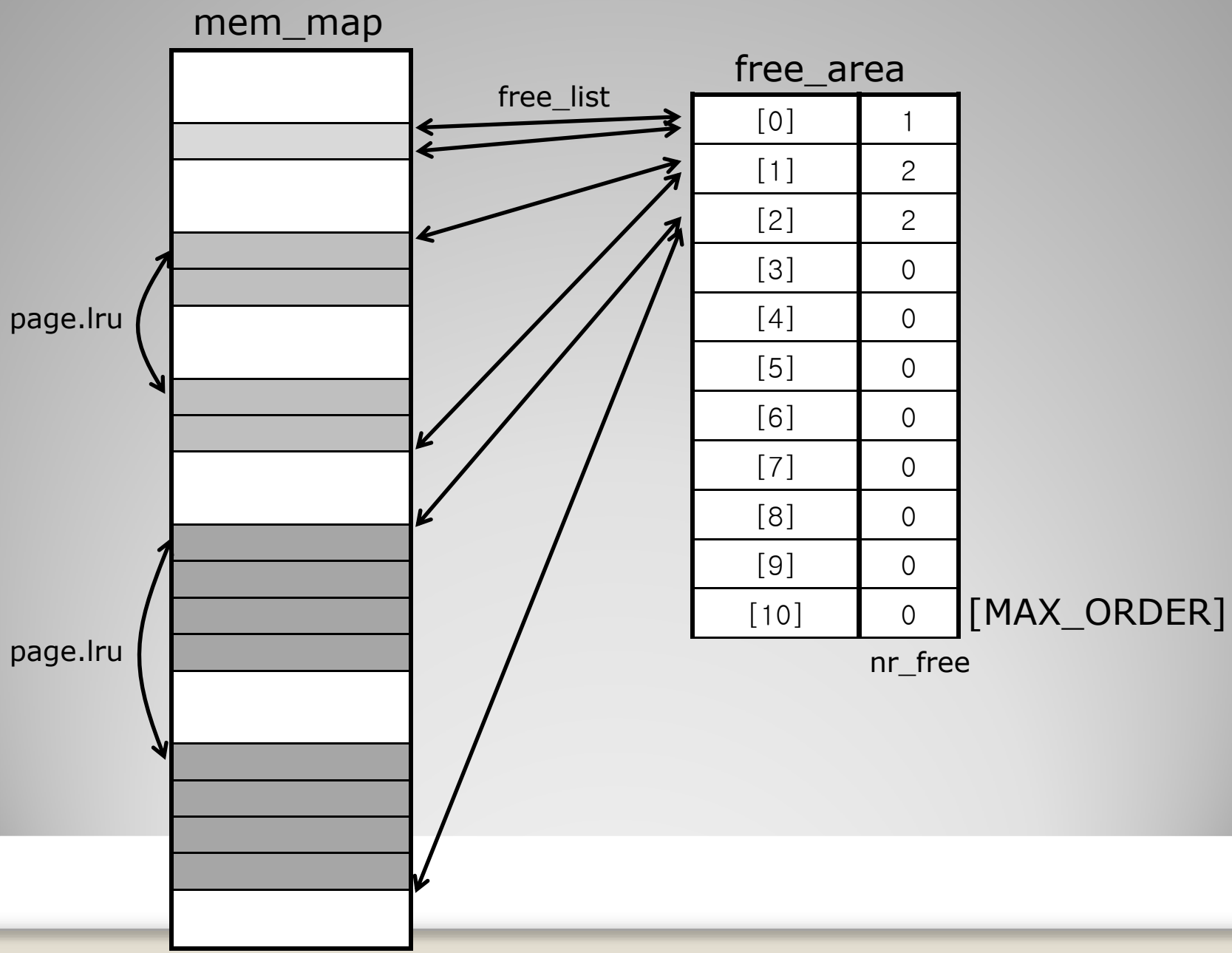
기본 개념

- 페이지 디스크립터
  - struct page
    - count : 페이지를 참조하는 프로세스의 수
    - list : 페이지 리스트
    - flags : 상태 플래그
    - typedef struct page mem_map_t;
  - mem_map_t mem_map[]
    - 페이지 개수만큼 배열로 만들어서 모든 페이지를 관리

기본 개념

- 연속된 페이지 블록의 리스트 관리 데이터
  - 동일한 개수의 페이지로 구성된 블록끼리 리스트로 관리
  - 각 메모리 존마다 구분
  - 보통 2^0~2^9개의 페이지 블록으로

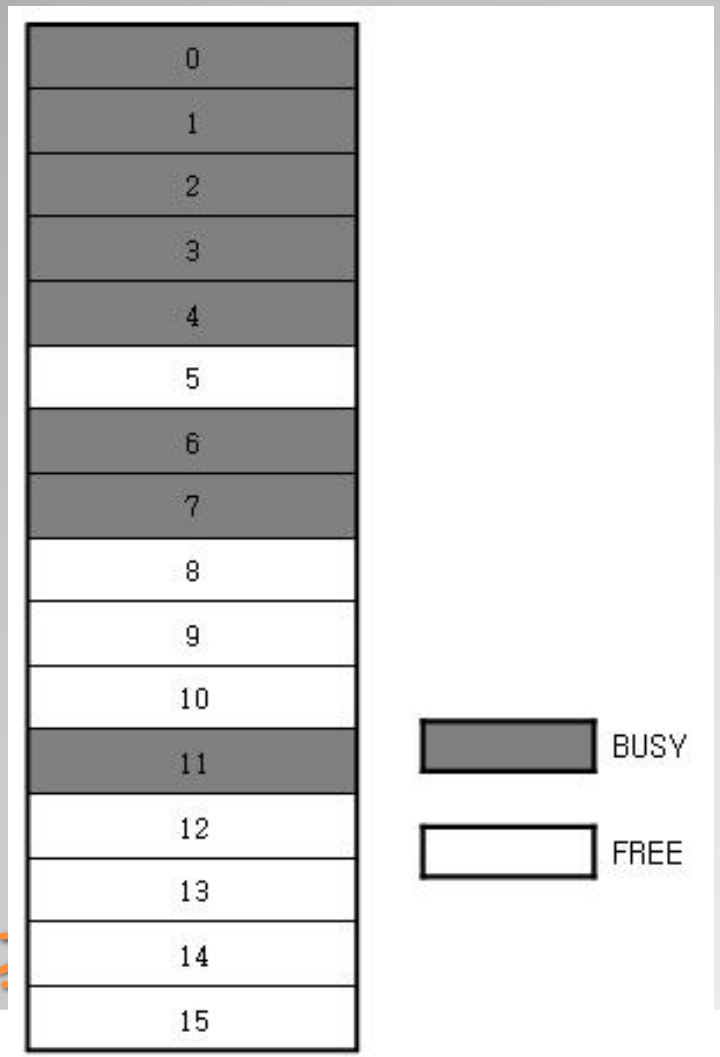
```
struct free_area {
    struct list_head free_list;
    unsigned long nr_free;
};
struct zone {
…
    struct free_area free_area[MAX_ORDER];
…
}
```


mem_map
free_list
free_area
[0] 1
[1] 2
[2] 2
[3] 0
[4] 0
[5] 0
[6] 0
[7] 0
[8] 0
[9] 0
[10] 0
[MAX_ORDER]
nr_free
page.lru
page.lru

버디 시스템
Buddy System

동작 예제

- 초기 상태



free_list 상태
- order(0) : 5, 10
- order(1) : 8 [8,9]
- order(2) : 12 [12, 13, 14, 15]

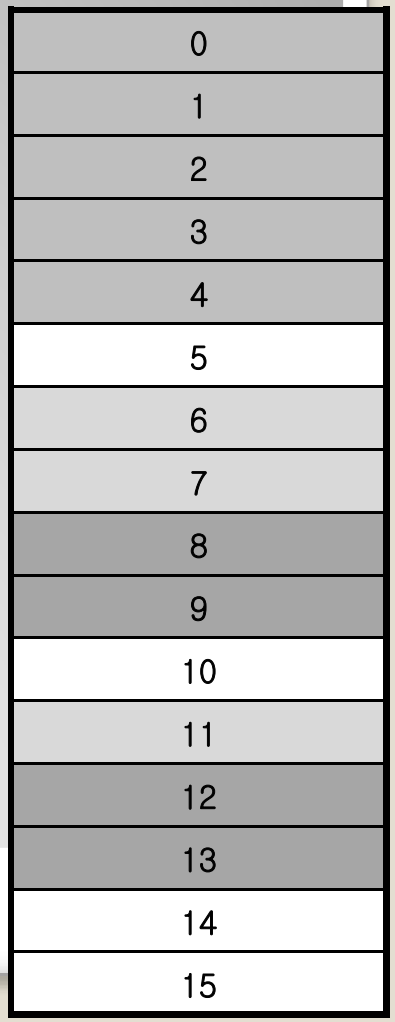
동작 예제

- 2페이지 할당
  - order(1) 리스트에서 #8 버디 할당

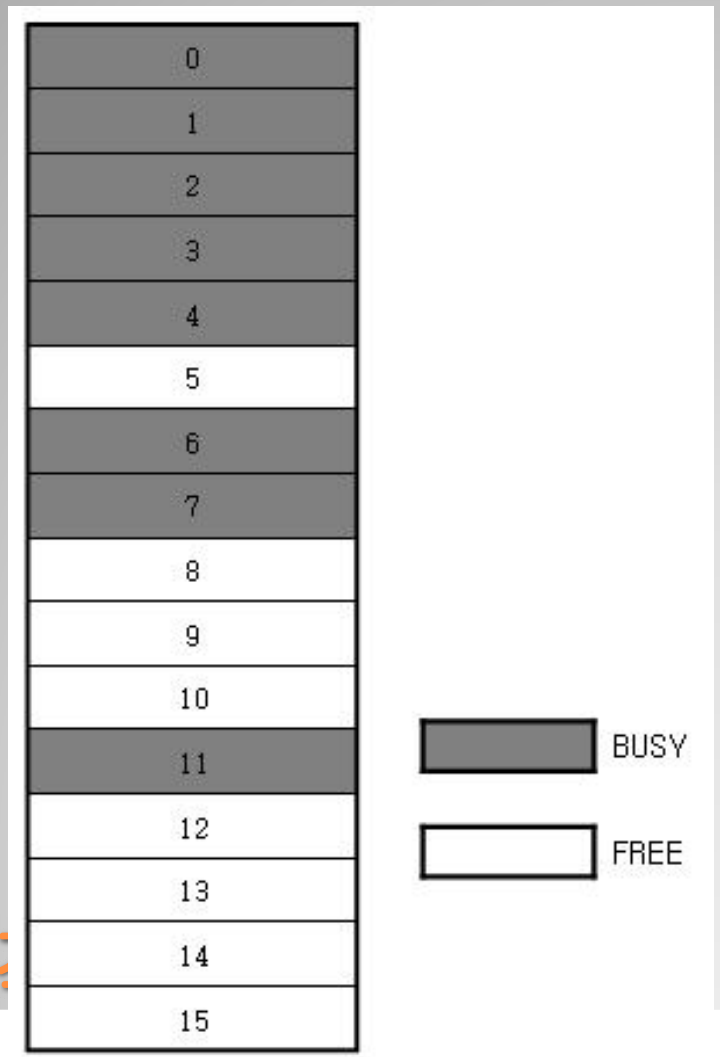
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

동작 예제

- 또 다시 2페이지 할당
  - order(1) 리스트 검색
  - 비어있는 블록이 없으므로 order(2) 검색
  - #12 블록을 [12,13] 과 [14,15] 로 나눔
  - [14,15]는 order(1)의 free_list로 연결
  - [12,13] 블록은 할당
  - free_list 결과
    - order(0) : 5, 10
    - order(1) : 14 [14,15]

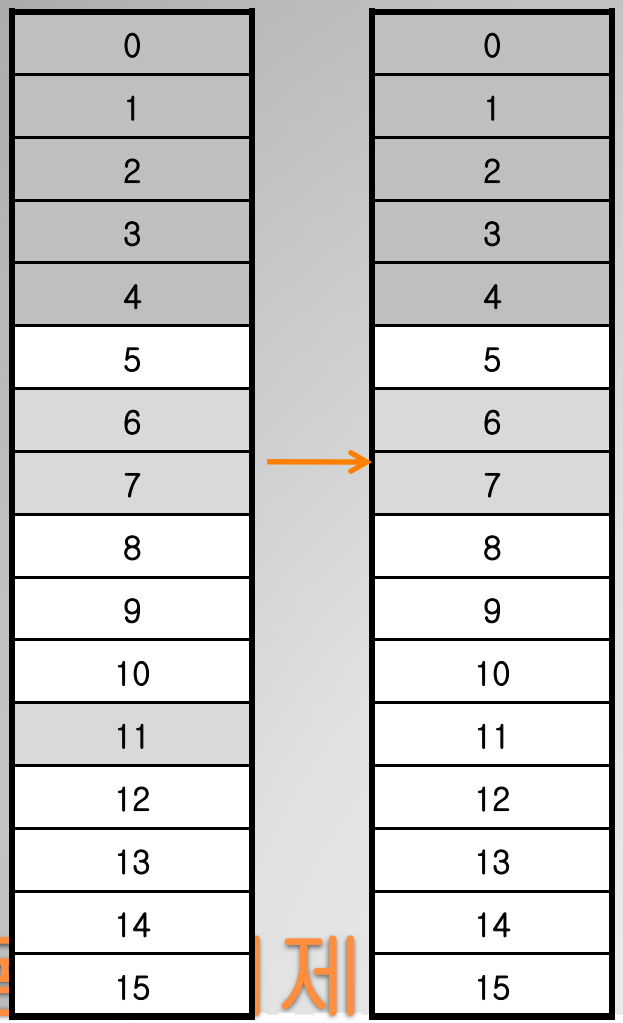


- 초기 상태



free_list 상태
- order(0) : 5, 10
- order(1) : 8 [8,9]
- order(2) : 12 [12, 13, 14, 15]

- #11 페이지 해제



- order(0) : 5, 10
- order(1) : 8 [8,9]
- order(2) : 12 [12, 13, 14, 15]
- order(3) :

- order(0) : 5, 10, 11
- order(1) : 8 [8,9]
- order(2) : 12 [12, 13, 14, 15]
- order(3) :

- order(0) : 5
- order(1) : 10 [10, 11] 8 [8,9]
- order(2) : 12 [12, 13, 14, 15]
- order(3) :

- order(0) : 5
- order(1) :
- order(2) : 8 [8,9,10,11] 12 [12, 13, 14, 15]
- order(3) :

- order(0) : 5
- order(1) :
- order(2) :
- order(3) : 8 [8~15]

- 버디 시스템
  - 기본 개념은 간단함
  - zone – numa – per_cpu_pages – mem_map 으로 연결되는 물리 메모리 관리 시스템의 이해가 필요함

동작 예제

버디 시스템
Buddy System

함수 코드

함수 코드

- 핵심 함수만 해석
  - __rmqueue
  - __free_one_page
  - 부록 문서에~

그 외 자료들
References

- References
  - Memory Management in Linux - Desktop Companion to the Linux Source Code
    - Abhishek Nayani
    - Mel Gorman & Rodrigo S. de Castro
    - 이 논문은 이 문서를 번역한 것 일뿐…
  - Understanding the Linux Kernel
    - 말이 필요없음
  - Intel 80386 Processor Manual
    - 인텔 프로세서의 페이지 관리 방식 이해
  - Functional Callgraph of the Linux VM
    - Martin Devera
    - 커널 2.4 버전이지만 커널의 동작 흐름
  - Process Management in Linux – Desktop Companion to the Linux Source Code
    - Kiran Divekar
    - 커널 2.4 버전이지만 프로세스 관리를 위한 모든 것을 이해할 수 있음

제 논문은 단지 차려진 밥상에서
밥을 먹은 것 뿐입니다~

감사합니다.