CaOS

(Calcium Operating System)

메모리 관리

2007.9.3 v0.05 문서 작성

김기오

www.asmlove.co.kr

메모리 관리

1. 메모리 관리 개요

메모리를 동적으로 할당하고 해제하기 위해서 페이지 단위 메모리 관리와 바이트 단위 메모리 관리가 필요하다. 현재는 먼저 페이지 관리 기능을 구현하였다. 프로세서의 페이징 기능을 사용하면서부터 물리 메모리는 페이지 단위로 접근되므로 페이지 관리 기능이 먼저 필요하기 때문이다.

견고한 메모리 관리를 위해서 오랜 시간 다양한 방법으로 테스트를 해야 한다. 현재는 개발 중이고 많은 테스트를 하지 않아서 하드코딩되거나 구현되지 못한 부분이 있다.

가장 먼저 setup_memory 함수가 실행된다. setup_memory 함수에서는 커널이 부팅되면서 사용한 페이지에 대한 정보를 관리하는 bootmem_init 함수와, 페이지 디렉토리와 페이지 테이블을 설정하는 paging_init 함수를 호출한다. 그리고 시스템의 물리 메모리를 관리하기 위한 정보들을 설정하는 init_mem_map 함수와 init_node 함수를 실행한다.

즉, setup_memory 함수는 페이지 관리를 위한 기본 데이터와 환경을 준비하는 일을 하고 실제 페이지 할당과 해제는 alloc_pages, get_free_pages, free_pages 함수에서 담당한다.

2. 부팅 메모리 관리

커널의 부팅 과정에서 GDT/IDT나 페이지 디렉토리 등 고정적으로 사용하는 페이지가 있다. 페이지 관리를 시작하기 전에 이렇게 미리 사용된 페이지를 관리하고 있어야 한다. setup.asm에서 설정되는 GDT나 IDT는 항상 같은 페이지에 저장되고 페이지 관리 테이블인 mem_map 은 비어있는 페이지를 찾아서 저장된다.

주의할 것은 페이지 관리를 시작하기 전에 만들어지는 데이터는 연속된 페이지를 사용하도록 해야한다는 것이다. IDT를 0x0 번지에 만들었으면 GDT를 0x1000 번지에, 연속된 페이지에 저장해서 사이에 비어있는 페이지가 없도록 하는 것이 페이지 관리에 도움이 된다. 만약 중간중간에 빈 페이지가 있다면 빈 페이지를 찾아내는 것부터 어렵게 된다. 페이지 관리 준비가 끝난 후라면 상관이 없지만 그 전에는 유연한 페이지 관리를 하기가 어렵기 때문이다.

부팅 메모리를 관리하기 위해서 page.c 파일에는 unsigned char boot_bitmap[] 이라는 unsigned char 형 배열이 있다. 물리 메모리의 처음 1MB 영역 중에서 비디오 메모리로 예약된 0xA0000~0xFFFFF 영역을 제외하면 총 0xA0 (120) 개의 페이지를 사용할 수 있고, 하나의 페이지를 한 비트로 표현하면 20바이트가 된다. 비트가 0이면 비어있는 페이지이고 1이면 사용중인 페이지로 표시하게 된다.

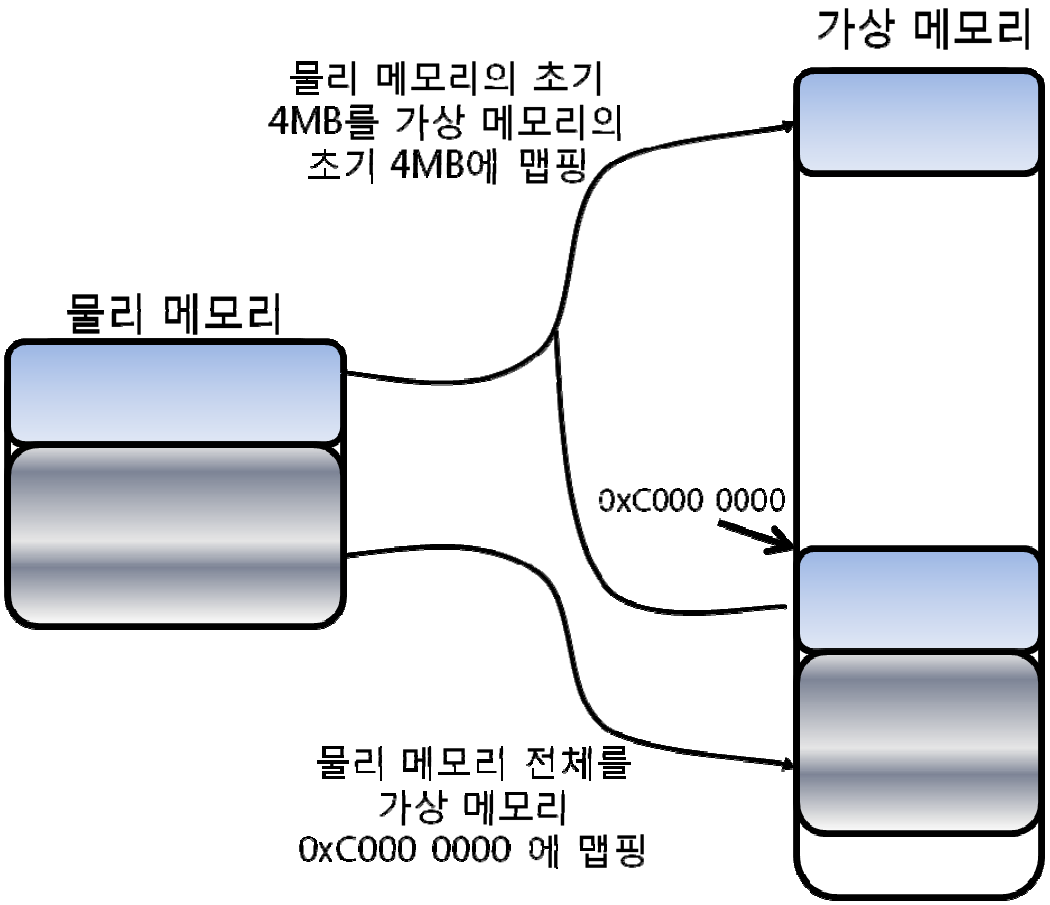
bootmem_init 함수에서는 boot_bitmap 배열을 모두 0으로 초기하고 setup.asm에서 GDT/IDT와 페이지 디렉토리, 페이지 테이블, TSS 로 사용된 0~7번 페이지의 비트를 1로 셋팅한다. 이 페이지

들은 항상 고정적으로 사용되기 때문에 하드코딩으로 할당시킨다.

alloc_bootmem 함수는 비어있는 페이지를 찾아서 해당 비트를 1로 셋팅하고 페이지의 시작 주소를 반환한다. 만약 여러 개의 페이지가 필요하다면 alloc_bootmem 함수를 여러 번 호출해서 연속된 페이지를 얻어서 첫번째로 얻은 페이지의 주소를 사용하면 된다.

3. 페이징 설정

setup.asm 파일에서는 하위 4MB 영역에 대한 페이지 디렉토리와 페이지 테이블을 설정했다. paging_init 함수에서는 나머지 물리 메모리에 대한 페이징을 설정한다. setup.asm 에서는 물리 메모리의 4MB를 가상 메모리 초기 4MB에 맵핑하고 동시에 가상 메모리의 0xC000 0000 에 맵핑한다. 그리고 paging_init 함수에서 나머지 4MB 이후 물리메모리를 0xC040 0000 이후에 맵핑한다.



결국 모든 물리 메모리를 0xC000 0000으로 맵핑하고 커널 모드일 때만 물리 메모리에 접근할 수 있게 한다. 또 물리 메모리 주소를 계산하는 것도 가상 메모리 주소 값에서 0xC000 0000 을 빼면 되고, 반대로 물리 메모리 주소에서 0xC000 0000을 더하면 가상 메모리 주소가 된다.

```
void paging_init(unsigned int phy_mem_size)
{
        int l;
        pte_t *pte;
        pgd_t *pgd;
        unsigned long addr;
        int pgd_count;

#ifdef DEBUG
        caos_printf("Init Page Global Dir₩n");
#endif

커널이 사용하고 있는 페이지 디렉토리의 주소
        swapper_pg_dir = (pgd_t *)SWAPPER_PG_DIR_ADDR;

        // PRINT PAGE DIR ENTRY!!
        //
#ifdef DEBUG
        caos_printf("## PGD check : %x -> 0x23=accessed, su, R/W, Pre₩n", swapper_pg_dir[0] );


        caos_printf("## Physical memory 0~4MB is mapping to 0xC0000000~0xC0400000₩n");
        caos_printf("0xB8000 -> 0xC00B8000, First char of scrren C->A₩n");
        do { char *t=(char *)0xC00b8000; *t = 'A'; } while (0);
#endif

0x301번 엔트리부터 물리 메모리에 매핑한다. 만약 물리 메모리가 32MB이면 페이지 테이블이 8개가 필요할 것인데 1개는 이미 setup.asm에서 설정했으므로 7개만 설정하면 된다. 따라서 7개의 페이지 디렉토리를 만들고 0x301~0x307번 엔트리들에 페이지 디렉토리의 주소를 저장하면 된다. 페이지 디렉토리의 주소를 저장할 때는 프로세서가 직접 읽어서 사용할 주소 값이므로 물리 메모리 주소 값으로 기록해야 한다.
        //
        // swapper_pg_dir[300] is already allocated for 0~4MB in setup.asm
        // This is mapping PAGE DIR for 4MB~ end of physical memory
        //
        pgd = swapper_pg_dir+0x301;
        pte_start = (unsigned long)alloc_bootmem();
```

```
        addr = pte_start + (_PAGE_RW | _PAGE_PRESENT);

        pgd_count = phy_mem_size/4;
        for (i=1; i<pgd_count; i++) {
                set_pgd(pgd, __pgd(addr));
#ifdef DEBUG
                caos_printf("swapper[%d]=%x ", i, addr);
#endif
                addr = (unsigned long)alloc_bootmem() + (_PAGE_RW | _PAGE_PRESENT);
                pgd++;
        }

#ifdef DEBUG
        for (i=0x2ff; i<0x309; i++)
                caos_printf("swapper_pg_dir[%d]=%x   ", i, swapper_pg_dir[i]);
        caos_printf("₩n");
#endif
```

페이지 테이블에는 페이지 프레임의 물리 주소를 쓴다. 물리 메모리가 32MB이면 총 8*1024개의 페이지 프레임이 있을 것이고 그 중 1024개는 setup.asm에서 설정했으므로 여기에서는 나머지 7*1024개의 페이지에 대한 설정을 해야 한다. 메모리 주소로 따지면 0x40000부터 0x1FFF000까지의 페이지 주소가 써질 것이다.

```
        //
        // mapping PAGE TABLE for 4MB ~ end of physical memory
        //
        pte = (pte_t *)pte_start;    // page table at 0x5000;
        addr = (4*0x100000) | _PAGE_RW | _PAGE_PRESENT; // 4MB +

        pgd_count = phy_mem_size/4 - 1;
        for (i=0; i<PTRS_PER_PTE*pgd_count; i++) {// fill out 7 tables
                set_pte(pte, __pte(addr));
                addr += PAGE_SIZE;
                pte++;
        }
```

페이징 설정에 문제가 없는지 확인하기 위해서 메모리에 값을 쓰고 읽어보는 테스트를 한다. 1MB 단위로 특정 값을 써보고 같은 값이 읽혀지는지 확인하면 주소 지정이나 페이징 설정에 문제가 없는지 간단하게나마 확인해볼 수 있다. 예를 들어 물리 메모리가 32MB이면, 0x400000(4MB)부터 1MB씩 증가하면서 0x1F00000까지 값을 써본다.

```
#ifdef DEBUG
        caos_printf("TEST PHY mapping..");
        do {
                char *pt;
                for (pt = (char *)0xC0400000; pt<0xc0000000+phy_mem_size*0x100000;
pt+=0x100000) {
                        *pt = 'a';
                        caos_printf("TEST ADDR=%x ", (unsigned long)pt);
                        if (*pt == 'a')
                                caos_printf("OK₩n");
                        else
                                caos_printf("FAIK₩n");
                }
        } while (0);
        caos_printf("OK₩n");
#endif
}
```

가상 메모리 주소와 페이지 관리 테이블에 있는 페이지 디스크립터를 매핑하는 함수들이다. 페이지의 가상 주소를 알면 __pa() 매크로를 이용해서 물리 주소를 알 수 있고 시스템에 있는 페이지 프레임 중에서 몇번째 페이지인지 알 수 있다.

```
extern mem_map_t *mem_map;
```

가상 주소를 페이지 디스크립터 page_t의 주소로 바꿔준다.

```
page_t *virt_to_page(unsigned long vaddr)
{
        unsigned int pfn = __pa(vaddr) >> PAGE_SHIFT;

#ifdef DEBUG
        caos_printf("virt (%x) to page #(%x)₩n", vaddr, pfn);
#endif
        return mem_map+pfn;
}
```

페이지 디스크립터를 페이지의 가상 주소로 바꿔준다.

```
unsigned long page_to_virt(page_t *pg)
```

```
{
        unsigned int pfn = pg - mem_map;
#ifdef DEBUG
        caos_printf("page #(%x) to virt (%x)₩n", pfn, __va(pfn<<PAGE_SHIFT));
#endif
        return (unsigned long)__va(pfn << PAGE_SHIFT);
}
```