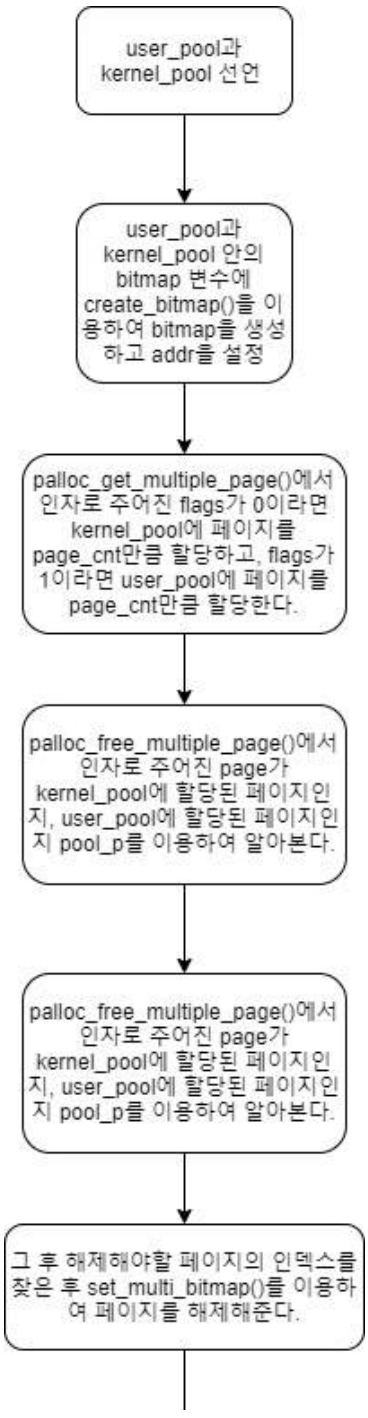
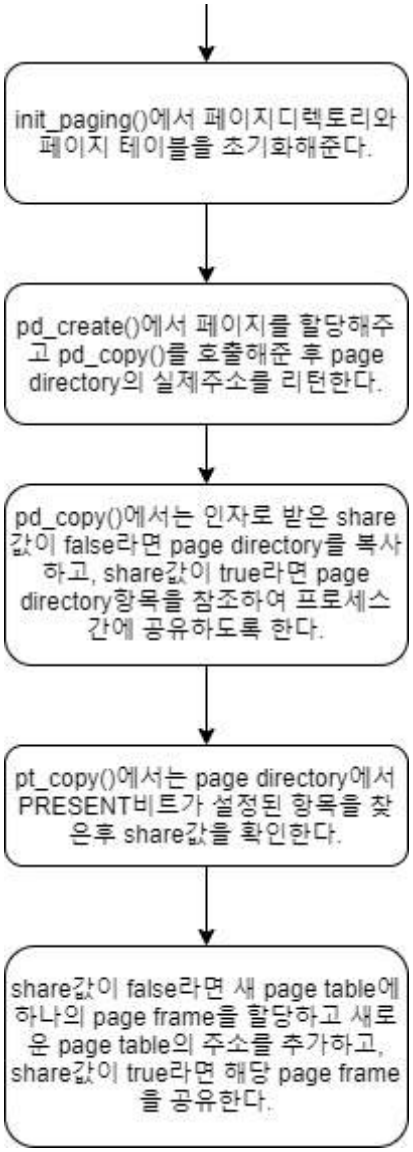
전자정보공학부 20160433 김민정

1. 개요

- 가상메모리랑 컴퓨터에서 주기억 장치 안의 프로그램 양이 많아질 때, 사용하지 않는 프로그램을 보조 기억 장치 안의 특별한 영역으로 옮겨서, 주기억 장치처럼 사용할 수 있는 공간이다. 프로세스에 할당된 가상메모리공간은 물리 메모리 공간에 맵핑이 되며 맵팽된 정보는 테이블 형태로 저장되어 주기억장치에 저장이 된다. 실 메모리를 kernel pool과 user pool로 나누어 구현한 후 각각 bitmap을 사용하여 사용가능한 page frame을 관리하는 Page Allocator 구현해본다.

2.상세설계

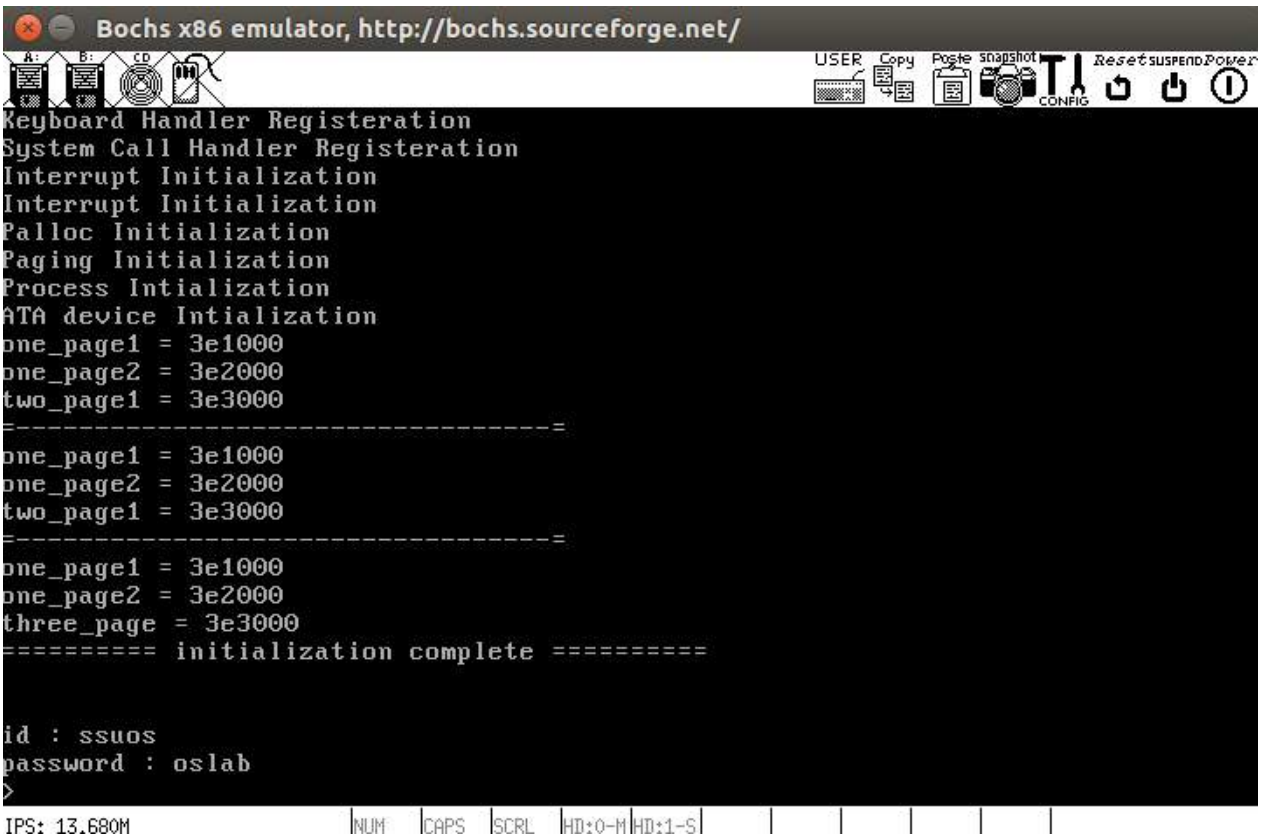




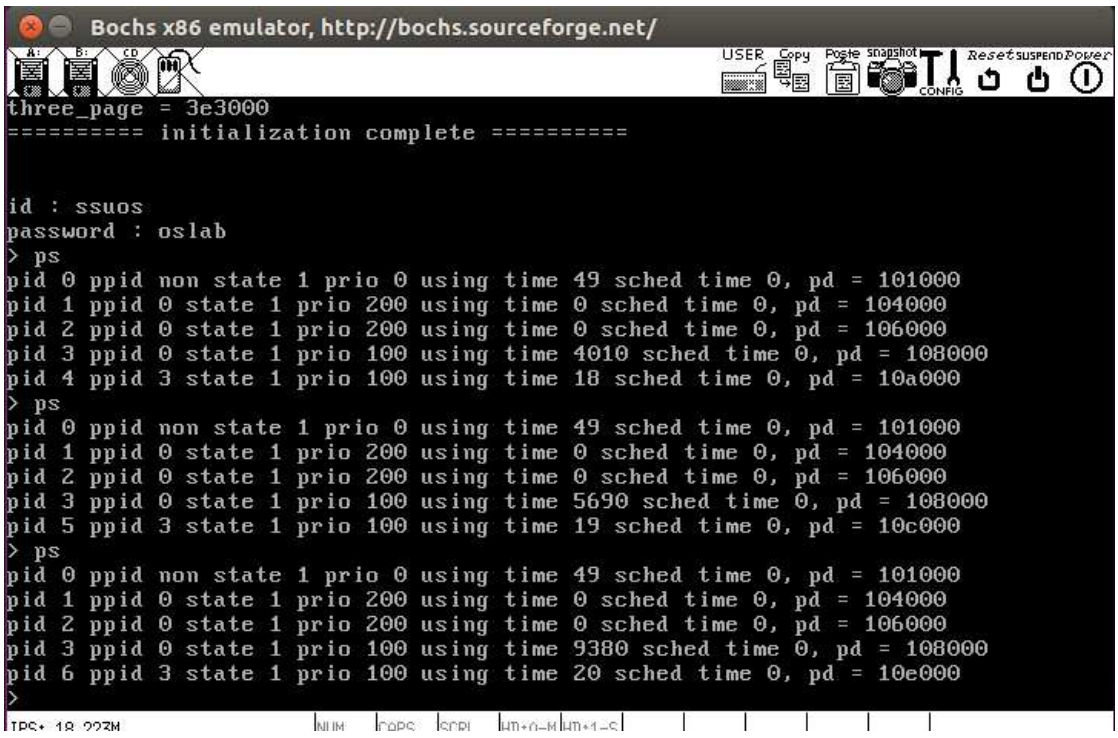
-bool pool_p (struct memory_pool *pool, void *page); //페이지가 어느 memory_pool에 속해있는지 알려주는 함수
-void init_palloc (void); //memory_pool 초기화 함수
-void* palloc_get_multiple_page(enum palloc_flags flags, size_t page_cnt); //인자로 주어진 cnt만큼의 페이지를 할당해주는 함수
-void* palloc_get_one_page (enum palloc_flags flags); //페이지 하나 할당해주는 함수
-uint32_t *palloc_free_multiple_page(void *pages, size_t page_cnt){ //인자로 주어진 cnt만큼의 페이지를 해제해주는 함수
-uint32_t *palloc_free_one_page (void *pages);//한 페이지 해제해주는 함수

3. 실행결과

1) [Case1 결과] 프로그램 수행시 결과화면 일부



2) [Case2 결과] 프로그램 수행 시 결과화면의 일부



4. 소스코드

-palloc.c

```
struct memory_pool user_pool; //user_pool 선언
struct memory_pool kernel_pool; //kernel_pool선언
bool pool_p ( struct memory_pool *pool, void *page); //페이지가 어느 memory_pool에 속해있는지 알려주는 함수

/* Initializes the page allocator. */
//
void init_palloc (void)
{

        size_t bm_pages = DIV_ROUND_UP (bitmap_struct_size (1), PGSIZE);

        lock_init (&user_pool.lock);
        lock_init (&kernel_pool.lock);
        user_pool.bitmap = create_bitmap((RKERNEL_HEAP_START-USER_POOL_START)/PAGE_SIZE,USER_POOL_START,bm_pages * PGSIZE); //user_pool에 bitmap생성
        kernel_pool.bitmap = create_bitmap((USER_POOL_START-KERNEL_ADDR)/PAGE_SIZE, KERNEL_ADDR,bm_pages * PGSIZE); //kernel_pool에 bitmap생성
        user_pool.addr = USER_POOL_START + bm_pages * PGSIZE; //user_pool의 addr 설정
        kernel_pool.addr = KERNEL_ADDR + bm_pages * PGSIZE; //kernel_pool의 addr 설정

}



/* Obtains and returns a group of PAGE_CNT contiguous free pages.
 */
void* palloc_get_multiple_page(enum palloc_flags flags, size_t page_cnt){
        struct memory_pool* tmp_pool;
        void *pages;
        size_t page_idx;
        size_t tm;
        if (page_cnt == 0)
                return NULL;

        //flags에 따라 kernel mode인지 user_mode인지 판단
        if(flags==0){//kernel mode라면
                tmp_pool = &kernel_pool;
        }
        else if(flags == 1){//user mode라면
                tmp_pool = &user_pool;
        }
```

```
        lock_acquire (&tmp_pool->lock);
        page_idx = find_set_bitmap(tmp_pool->bitmap, 0, page_cnt, FALSE);//bitmap에서 비어있는 인덱스 찾아 할
당
        lock_release (&tmp_pool->lock);

        if (page_idx != BITMAP_ERROR){
                pages = (size_t *)tmp_pool->addr + (page_idx * PAGE_SIZE/sizeof(page_idx)); //페이지 주소 찾기
        }
        else
                pages = NULL;

        if (pages != NULL)
        {
                memset (pages, 0, PGSIZE * page_cnt);
        }

        return pages;
}

/* Obtains a single free page and returns its address.
 */
void* palloc_get_one_page (enum palloc_flags flags)
{
        return palloc_get_multiple_page (flags, 1); //1페이지 할당할때
}

/* Frees the PAGE_CNT pages starting at PAGES. */
uint32_t *palloc_free_multiple_page(void *pages, size_t page_cnt){ //여러 페이지 해제해주는 함수

        struct memory_pool* tmp_pool;
        size_t page_idx;


        if (pages == NULL || page_cnt == 0)
                return NULL;

        if(pool_p(&kernel_pool, pages)){ //kernel_pool이라면

                tmp_pool = &kernel_pool;
        }

        else if(pool_p(&user_pool, pages)){ //user_pool이라면
```

```
            tmp_pool = &user_pool;
        }

        page_idx = pg_no(pages) - pg_no(tmp_pool->addr); //해제해야할 인덱스 찾기

        if(bitmap_all (tmp_pool->bitmap, page_idx, page_cnt))
            set_multi_bitmap (tmp_pool->bitmap, page_idx, page_cnt, FALSE); //비트맵에서 해당하는 인덱스 찾아 페이지 해제해주기

        return tmp_pool->addr;

}

/* Frees the page at PAGE. */
uint32_t *palloc_free_one_page (void *pages){
        palloc_free_multiple_page (pages, 1); //한페이지 해제할때
}

bool pool_p( struct memory_pool *pool, void *page) //해제해야할 페이지가 user_pool에 있는지 kernel_pool에 있는지 페이지 주소로 판단하는 함수
{
        return (pg_no (page) >= pg_no (pool->addr) && pg_no (page) < (pg_no (pool->addr) + bitmap_size (pool->bitmap)));

}
```

-paging.c

```
/*
   page table 복사
 */
void pt_copy(uint32_t *pd, uint32_t * dest_pd, uint32_t idx, uint32_t* start, uint32_t *end, bool share){

        uint32_t *pt = pt_pde(pd[idx]);
        uint32_t *new_pt;
        uint32_t s, e, i;
        new_pt = palloc_get_one_page(kernel_area);
        dest_pd[idx] = (uint32_t)new_pt | (pd[idx] & ~PAGE_MASK_BASE & ~PAGE_FLAG_ACCESS);
        s = pte_idx_addr(start);
        e = pte_idx_addr(end);

        for(i = s; i<e; i++)
        {
            if(pt[i] & PAGE_FLAG_PRESENT) //source page directory에서 전체 항목들을 검사해 PRESENT 비트가 설정 된 항목들을 찾고
```

```
                {
                        //share가 F(0)로 설정된 경우 새로운 page table에 하나의 page frame을 할당하고
destination directory의 인자로 받은 idx 번째 항목에 새로운 page table의 주소를 추가.
                        if(share == FALSE){
                                uint32_t* pg = palloc_get_one_page(kernel_area);
                                new_pt[i] = (uint32_t)pg | (pd[i] & ~PAGE_MASK_BASE & ~
PAGE_FLAG_ACCESS);
                                memcpy_hard((void*)(pt[i] & PAGE_MASK_BASE), (void*)pg, PAGE_SIZE);
                        }
                        //share가 T(1) 인 경우 해당 페이지 page frame을 공유함
                        else{ //share == true
                                new_pt[i] = pt[i];
                        }
                }
        }
}

/*
   page directory 복사.
   커널 영역 복사나 fork에서 사용
 */
void pd_copy(uint32_t *pd, uint32_t * dest_pd, uint32_t idx, uint32_t* start, uint32_t *end, bool share){

        uint32_t i;

        for(i = 0; i < 1024; i++){
                if(pd[i] & PAGE_FLAG_PRESENT){
                        if(share== FALSE){//share = FALSE 일때,  page directory를 복사
                                pt_copy(pd, dest_pd, i, start, end, share);
                        }
                        else{ //share = TRUE 일때, page directory 항목을 프로세스간에 공유
                                dest_pd[i] = pd[i];
                        }
                }
        }
}

uint32_t* pd_create (pid_t pid){
        uint32_t *pd = palloc_get_one_page(kernel_area);
        //pd_copy()호출
        pd_copy((uint32_t*)read_cr3(), pd ,0 ,(uint32_t*)0, (uint32_t*)USER_POOL_START,TRUE);
        //성공시 page directory의 실 주소를 리턴
        return pd;
}
```