과제 #6 : Virtual Memory

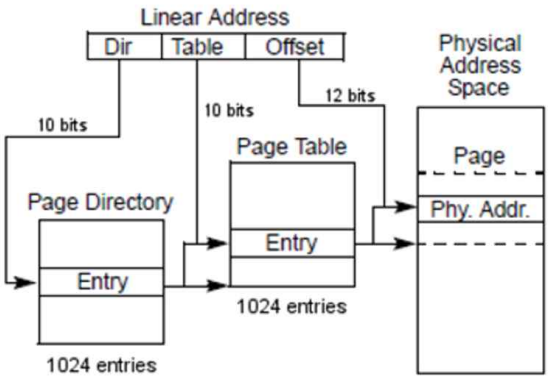
○ 과제 목표

- 가상메모리 Page Allocator 구현

○ 기본 배경 지식

- 가상 메모리
  - ✓ 메모리 사용량이 늘어남에 따라 디스크의 일부를 주기억 장치처럼 사용할 수 있게 해주는 기법
  - ✓ 커널이 주기억 장치에 적재된 메모리 블록 중 당장 쓰이지 않는 것을 디스크에 저장함으로써 사용 가능한 메모리 영역을 늘림
  - ✓ 만일 디스크에 저장된 메모리 블록이 필요해지면 다시 주기억 장치에 적재되며 다른 블록이 디스크에 저장됨
  - ✓ 현재 SSUOS의 가상메모리 구현에서는 가상주소를 page directory와 page table을 통해 실 주소로 변경하여 사용하지만 메모리 공간 할당 시 실주소와 같은 가상주소를 변환하여 사용. 또한 디스크의 SWAP 영역은 사용하지 않음

- 페이징
  - ✓ 가상 메모리를 모두 같은 크기의 블록으로 관리하는 기법
  - ✓ 가상 메모리의 일정한 크기를 가진 블록을 page라고 하며 실 메모리의 블록을 page frame라고 함
  - ✓ 가상 메모리 구현은 가상 메모리가 참조하는 page 주소(가상주소)를 실 메모리 page frame 주소(실주소)로 변환하는 것
  - ✓ SSUOS의 page frame의 크기는 4KB로 기본적으로 설정되어 있음



[그림 1] 가상주소의 실주소 변환 과정

○ 과제 내용

- 다운로드 받은 SSUOS는 이전에 구현한 메모리 구조와 page frame 구조(이하 이전 메모리 구조라고 함)이기 때문에 과제의 내용을 다시 구현해야 함

- 기존 SSUOS 메모리 구조

✓ 기존 page allocator는 paging.h에 정의되어 있는 주기억장치의 RKERNEL_HEAP_START(2MB)부터 6MB까지 1024개의 page frame(page frame 블록 한개당 4KB)를 관리하고 있음

✓ 각 page frame의 구조체(총 1024개)는 주기억장치의 1MB부터 6개의 page frame에 저장하고 있음

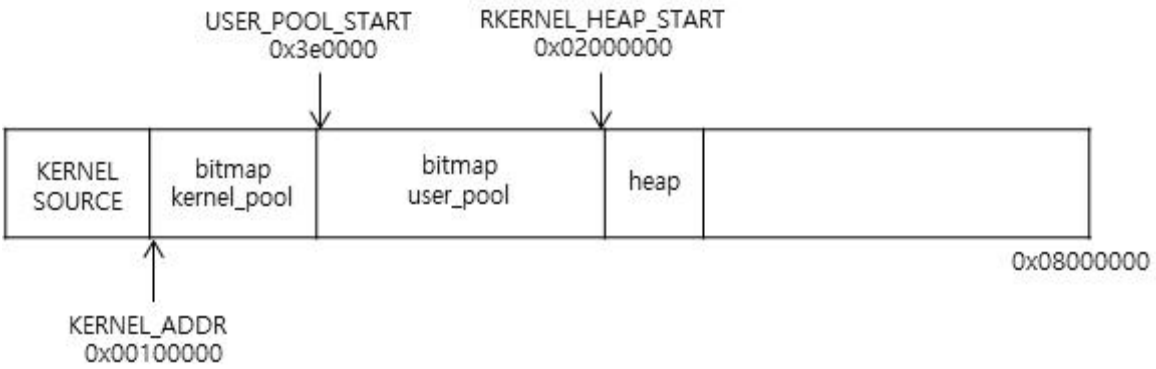
✓ palloc()는 사용 가능한 page frame 하나를 할당해 주는 것으로 성공하면 해당 메모리 블록(page frame)의 가상주소를 리턴하였음

- 새로운 page allocator 구현

✓ palloc()는 사용 가능한 page frame 하나를 user pool이나 kernel pool에서 할당해 주는 것으로 성공하면 해당 메모리 블록(page frame)의 실주소를 리턴함

✓ 이를 위하여 실메모리를 kernel pool(시작주소 :KERNEL_ADDR)과 user pool(시작 주소: USER_POOL_START)로 나누어서 구현해야 함

✓ kernel pool 및 user pool은 각 bitmap을 사용하여 사용 가능한 page frame을 관리해야 함



[그림 2] bitmap을 사용하여 구현해야 하는 메모리 구조

○ 과제 수행 내용

- (1) kernel pool과 user pool의 구현

✓ 전체 실메모리 크기 : 0x08000000

✓ kernel pool의 시작 주소 : KERNEL_ADDR 0x00100000

✓ user pool의 시작 주소 : USER_POOL_START 0x3e0000

✓ kernel heap의 시작 주소 : RKERNEL_HEAP_START 0x02000000

- (2) void* palloc_get_multiple_page(enum palloc_flags flags, size_t page_cnt) 함수 구현

✓ return 값 : 할당된 page frame의 실주소

✓ cnt : kernel pool과 user pool에서 cnt만큼 page frame을 할당

✓ flag : enum형으로 kernel mode(0), user mode(1)로 설정

- (3) 프로세스 생성 시 page directory 및 page table 생성 수정 구현

✓ ssuos/src/kernel/mem/paging.c의 pd_copy(), pt_copy() 수정

- 기존 구현에는 pd_create(pid_t pid)는 새로 생성될 프로세스의 page directory로 사용할 page frame를 할당 받은 후 pd_copy() 호출하고 성공 시 page directory의 가상주소를 리턴함

- 기존 pd_copy(uint32_t *from, uint32_t *to)는 pt_copy(uint32_t *pd, uint32_t *dest_,

unit32_t idx)를 호출하여 새로운 page table을 위한 page를 할당

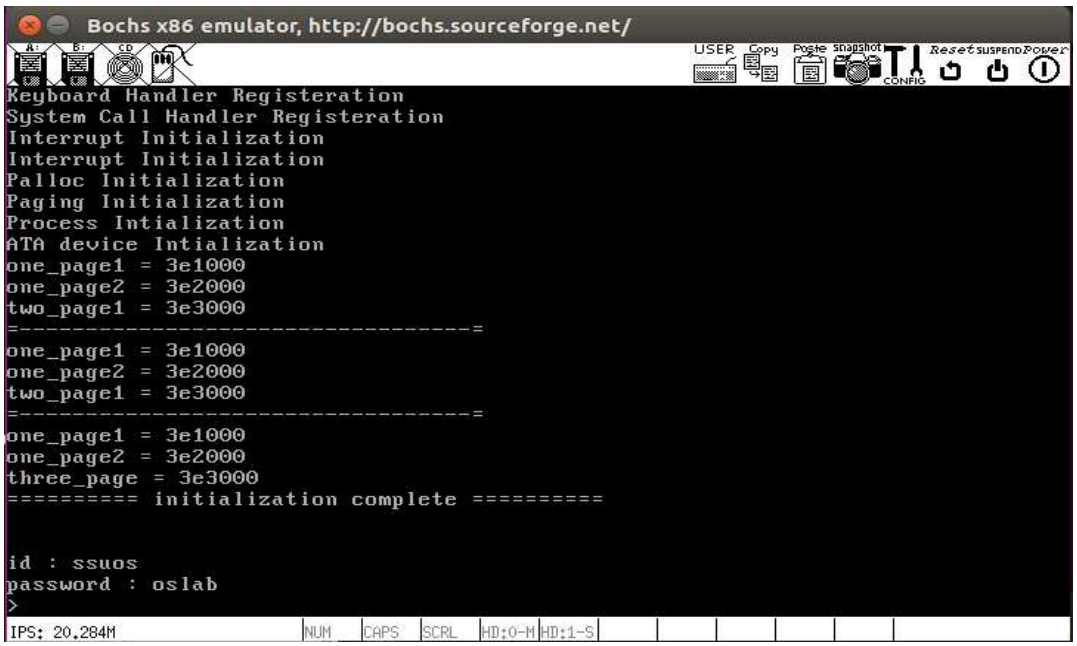
- 새로운 pd_copy(uint32_t *pd, uint32_t *dest_pd, uint32_t idx, uint32_t* start, uint32_t* end, bool share)를 구현하여 프로세스 간 해당 page directory 항목을 공유할 수 있게 해야 함
  - bool share (T/F 또는 1/0) : 0인 경우 해당 page directory를 복사하고 1인 경우 해당 page directory 항목을 프로세스 간에 공유함
  - 새로운 pd_create()는 pd_copy() 호출하고 성공 시 page directory의 실주소를 리턴해야 함
- 새로운 pt_copy(uint32_t *pd, uint32_t *dest_pd, uint32_t idx, uint32_t* start, uint32_t* end, bool share)를 구현하여 프로세스 간 page table을 공유할 수 있게 해야 함
  - 새로운 pt_copy()는 source page directory에서 전체 항목들을 검사해 PRESENT 비트가 설정된 항목들을 찾고 share가 F(0)로 설정된 경우 새로운 page table을 하나의 page frame을 할당하고 destination directory의 인자로 받은 idx 번째 항목에 새로운 page table의 주소를 추가. share가 T(1) 인 경우 해당 페이지 page frame을 공유함

- (4) page frame 삭제 기능 구현

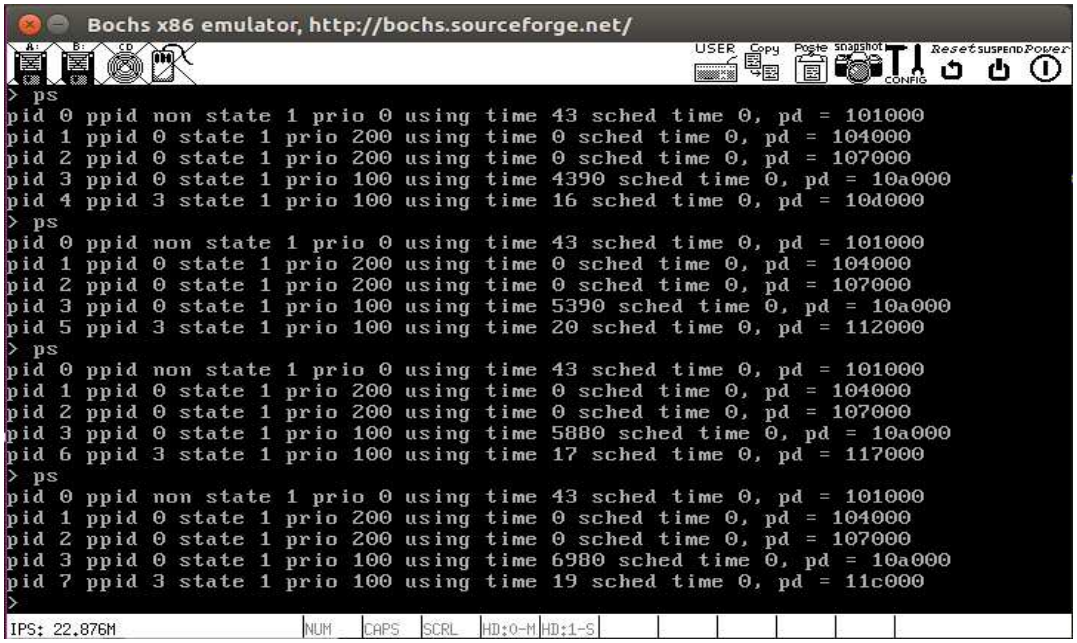
✓ uint32_t *palloc_get_multiple_page_page (enum palloc_flags flags ,size_t page_cnt) 구현

- kernel pool과 user pool에 인자로 주어진 cnt만큼 page frame을 제거
- return 값 : 할당된 page frame의 시작 주소
- flag : enum형으로 kernel mode(0), user mode(1)로 설정

- (5) 과제 수행 결과
- [Case 1], [Case2]와 같은 결과가 나오도록 구현해야 함



[Case1 결과]프로그램 수행시 결과화면 일부



[Case 2 결과] 프로그램 수행 시 결과화면의 일부

○ 구현 시 주의할 점

- mem/palloc.c에서 void palloc_pf_test(void) 함수의 주석 삭제
- RKENEAL_START 주소 값을 0x02000000으로 변경
- kernel/mem/palloc.c에서의 함수명은 그대로 두고 인자를 수정
- 메모리 주소를 하드코딩으로 page frame을 설정하지 말 것
- 구현 시 필요 변수 임의로 선언 및 구현된 함수를 수정 및 사용 가능

```
void palloc_pf_test(void)
{
    // uint32_t *one_page1 = palloc_get_one_page(user_area);
    // uint32_t *one_page2 = palloc_get_one_page(user_area);
    // uint32_t *two_page1 = palloc_get_multiple_page(user_area,2);
    // uint32_t *three_page;
    // printk("one_page1 = %x\n", one_page1);
    // printk("one_page2 = %x\n", one_page2);
    // printk("two_page1 = %x\n", two_page1);

    // printk("=---------------------------------=\n");
    // palloc_free_one_page(one_page1);
    // palloc_free_one_page(one_page2);
    // palloc_free_multiple_page(two_page1,2);

    // one_page1 = palloc_get_one_page(user_area);
    // two_page1 = palloc_get_multiple_page(user_area,2);
    // one_page2 = palloc_get_one_page(user_area);

    // printk("one_page1 = %x\n", one_page1);
    // printk("one_page2 = %x\n", one_page2);
    // printk("two_page1 = %x\n", two_page1);

    // printk("=---------------------------------=\n");
    // palloc_free_multiple_page(one_page2, 3);
    // one_page2 = palloc_get_one_page(user_area);
    // three_page = palloc_get_multiple_page(user_area,3);

    // printk("one_page1 = %x\n", one_page1);
    // printk("one_page2 = %x\n", one_page2);
    // printk("three_page = %x\n", three_page);

    // palloc_free_one_page(one_page1);
    // palloc_free_one_page(three_page);
    // three_page = (uint32_t*)((uint32_t)three_page + 0x1000);
    // palloc_free_one_page(three_page);
    // three_page = (uint32_t*)((uint32_t)three_page + 0x1000);
    // palloc_free_one_page(three_page);
    // palloc_free_one_page(one_page2);
}
```

[그림 3] 구현 후 palloc_pf_test 함수 주석 해제

○ 과제제출

- 2019년 11월 26일 (화) 23시 59분까지 제출
- 배점 기준
  - ✓ 보고서 10%
    - 개요 2%
    - 상세 설계 명세(기능 명세 포함) 5%
    - 실행 결과 3%
  - ✓ 소스코드 90%
    - 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
    - 실행 여부 85% ((1)번 15점, (2)번 15점, (3)번 20점, (4)번 15점, (4)번 20점)
- 최소 구현사항
  - ✓ 과제 수행 내용 (1), (2), (4)