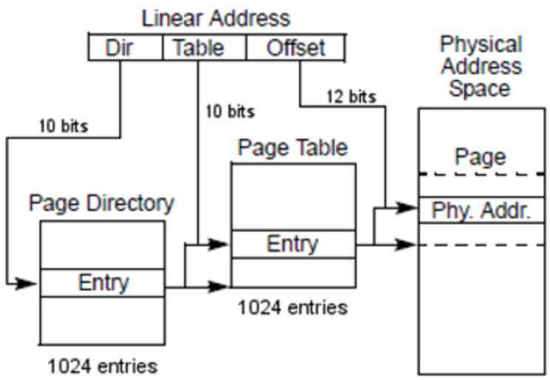
과제 #5 : Virtual Memory

○ 과제 목표

- 가상 메모리 구조 이해
- Page Allocator

○ 기본 배경 지식

- 가상 메모리
  - ✓ 디스크의 일부를 마치 확장된 주기억장치처럼 사용할 수 있게 해주는 기법
  - ✓ 커널이 주기억장치에 적재된 블록 중 일부를 디스크 공간에 저장함으로써 사용가능한 메모리 영역을 늘림
  - ✓ 만일 디스크에 저장된 블록이 필요하면 다시 주기억장치에 적재
  - ✓ 프로세스의 할당된 가상 메모리 공간은 물리 메모리 공간에 맵핑되며 매핑 정보는 테이블형태로 저장되어 주로 주기억장치에 저장됨
  - ✓ 현재 SSUOS에서 가상메모리 구현은 가상 메모리 주소를 기반으로 page directory와 page table을 통해 물리 메모리 주소로 변경이 가능함. 그러나, 디스크 영역을 가상 메모리로 사용하지 않음
- SSUOS 페이징 기법
  - ✓ 가상 메모리를 모두 같은 크기의 블록으로 관리하는 기법
  - ✓ 가상 메모리의 일정한 크기를 가진 블록을 page라고 하며 물리 메모리의 블록을 page frame라고 함
  - ✓ 가상 메모리 주소 변환은 가상 메모리가 참조하는 page주소를 물리 메모리 page frame주소로 변환하는 것으로 시스템 마다 다르게 구현되어 있음
  - ✓ SSUOS에서는 [그림 1]과 같이 2단계 페이지 테이블(페이지 디렉토리 및 페이지 테이블)을 이용하여 가상 주소를 물리 주소로 변환
  - ✓ SSUOS에서 한 개의 page 크기는 4KB로 이루어져 있으며, 가상 주소는 총 32비트로 페이지 디렉토리를 가리키는 10비트, 페이지 테이블을 가리키는 10비트, 오프셋 12비트로 이루어짐.
  - ✓ 참고로 오프셋이 12비트이면 page frame의 크기도 4KB를 의미함



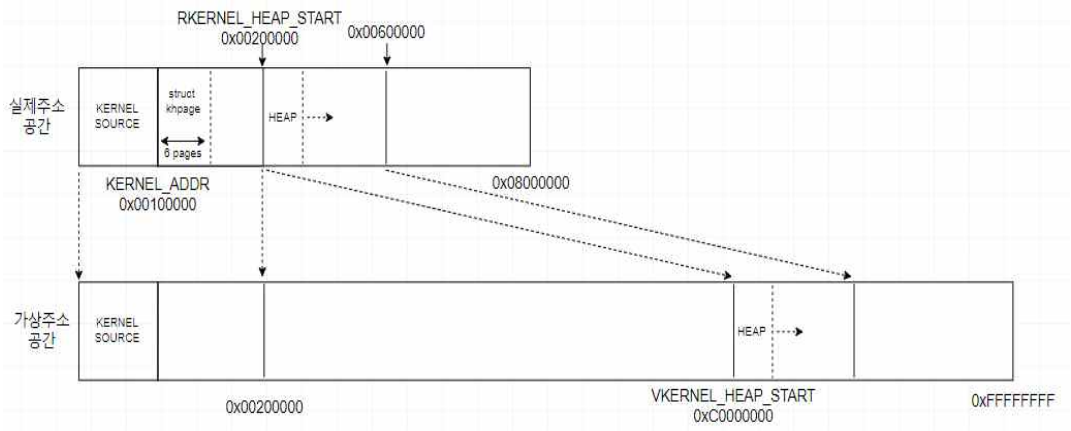
[그림 1] 가상 선형 주소의 변환 예

- SSUOS의 메모리 맵

✓ 기존 SSUOS 가상 메모리 주소 공간의 0 ~ 2MB까지는 실제 메모리 상에서 동일하게 맵핑되며 paging,c 의 init_paging() 함수에서 page table을 2MB까지 초기화함

✓ Page allocator를 통해서 할당되는 page들은 실제주소 2MB 이후부터 4KB씩 할당되는데 리턴되는 주소는 가상주소인 VKERNEL_HEAP_START(3GB) 이후와 매핑되어 있음

✓ 현재 SSUOS의 생성되는 모든 프로세스는 커널 프로세스로 가상 커널 영역의 메모리들을 모두 공유함



[그림 2] 기존 SSUOS의 메모리 맵

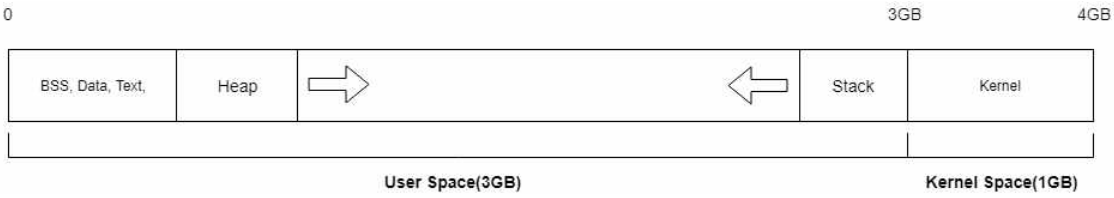
○ 과제 내용

- 커널 프로세스를 위한 새로운 메모리 공간

✓ [그림 3]와 같이 32비트 기반의 UNIX 시스템의 가상 메모리 주소 공간은 각 프로세스는 4GB의 크기이며, 하위 3GB 공간은 사용자 프로세스가 상위 1GB 공간은 커널 프로세스가 사용

✓ [그림 4]은 구현할 SSUOS의 메모리 구조로 사용자 프로세스와 커널 프로세스의 개념은 없는 구조로 모두 커널 프로세스가 전체 4GB 주소 공간을 모두 사용함. Bochs 에뮬레이터를 사용하는 SSUOS의 경우 각 프로세스를 위한 Text, Data, Stack 공간은 별도로 존재하지 않음

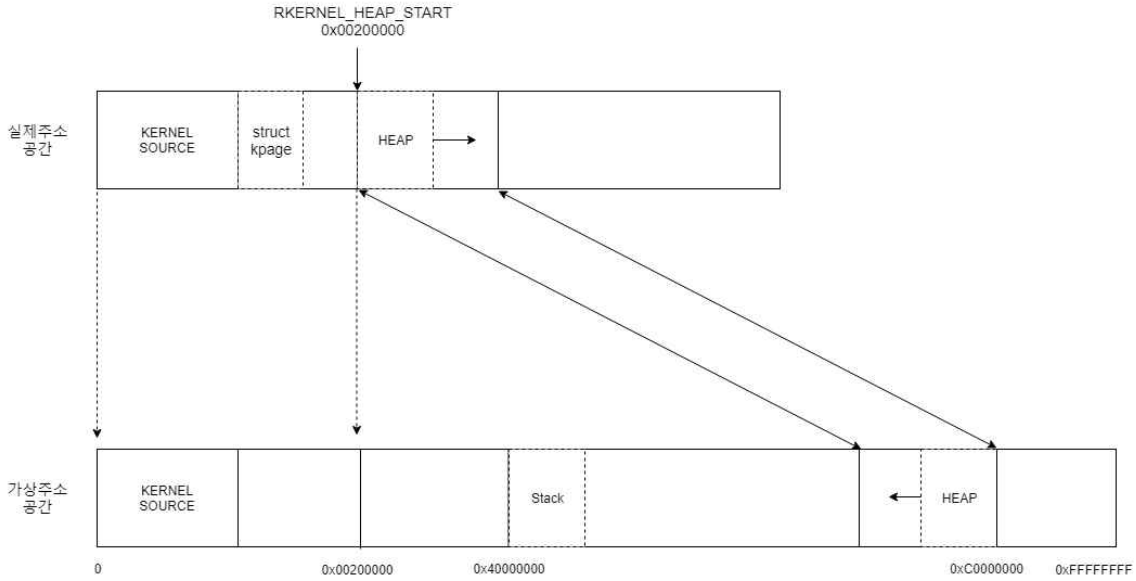
- 1G 영역부터 Stack이 3G 영역부터 Heap 이 사용됨



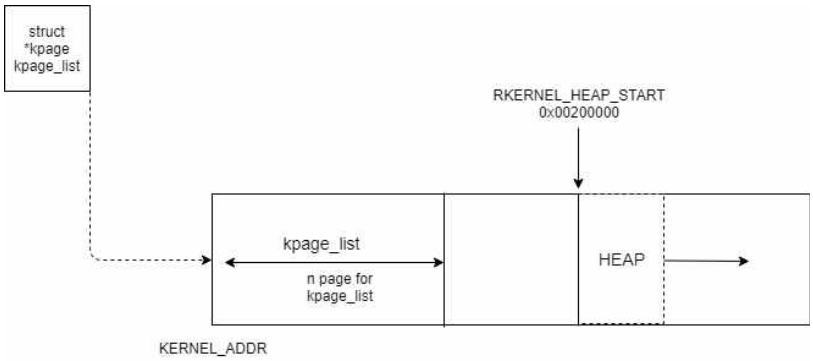
[그림 3] UNIX 시스템의 가상 메모리 구조

[그림 4] 새로운 SSUOS의 메모리 맵

- 가상 메모리 맵



[그림 5] SSUOS 가상 메모리 맵



[그림 6] kpage_list 할당

- palloc.c 구현

- ✓ 현재 Page Allocator는 paging.h에 정의되어 있는 주기억 장치 상 RKERNEL_PAGE_START(2MB)부터 6MB까지 4KB 크기의 PAGE_POOL_SIZE개의 page을 관리함, 주기억 장치 상 1MB 지점부터 1개의 page의 정보를 저장하는 kpage 구조체를 저장. 참고로 kpage 구조체는 type, *vaddr, nalloc, pid 등으로 이루어짐.
- ✓ palloc_get_page()는 사용가능한 page 하나를 커널에게 할당 해 주는 것으로 가상 주소를 갖는 메모리 블록을 리턴 받을 수 있음
- ✓ Heap의 경우, palloc_get_multiple() 함수에서 리턴되는 메모리 블록의 주소는 paging.h에 정의되어 있는 VKERNEL_HEAP_START(3GB) 지점부터 리턴 가능하며 heap이 할당 받았던 메모리 블록 주소와 연속되게 할당되어짐
- ✓ Stack의 경우, palloc_get_multiple() 함수에서 리턴되는 메모리 블록의 주소는 paging.h에 정의되어 있는 (VKERNEL_STACK_ADDR) (1GB)부터 지점부터 리턴 가능하며 각 프로세스가 Stack의 시작주소를 받을 때는 항상 같은 주소(VKERNEL_STACK_ADDR)의 메모리 블록을 리턴 받음.

✓ 가상 메모리 주소 공간의 0 ~ 2MB까지는 실제 메모리 공간과 동일하게 맵핑되어 있기 때문에 paging.c 의 init_paging() 함수에서 page table을 2MB까지 초기화함
✓ Page allocator를 통해서 할당되는 page들은 실제주소 2MB 이후부터 4KB씩 할당되는데 리턴 되는 주소는 가상주소인 VKERNEL_HEAP_START(3GB) 이후와 매핑
✓ SSUOS에서 생성되는 모든 프로세스는 [그림 5]와 동일한 가상 메모리 맵을 가짐

○ 과제 수행 방법

- (1) Heap 구현
✓ ssuos/src/kernel/mem/palloc.c 의 heap page 할당 관련 함수들 구현
✓ Heap 영역을 할당 받을 경우, palloc_get_multiple() 함수의 리턴 값으로 VKERNEL_HEAP_START를 기준으로 할당받을 페이지로 주소변환 후 리턴함. 단, 할당할 영역을 0으로 초기화
✓ palloc_free_multiple() 함수를 완성하여 성공적으로 page 반환을 해야함
✓ va_to_ra() 및 ra_to_va() 함수를 완성
✓ 현재 pid 0번 프로세스는 ssuos/src/kernel/arch/Main.c 의 초기화 작업들 후 page fault
✓ **'(1) Heap 구현' 이후, Main.c 의 main_init() 함수 내부의 palloc_pf_test() 함수 호출 아래의 while(1); 삭제후 (2) 수행**

- (2) Stack 구현
✓ ssuos/src/kernel/mem/palloc.c 의 stack page 할당 관련 함수들 구현
✓ 부모가 자식 프로세스의 스택할당을 위해 해당 스택주소를 할당받을 때, 부모 프로세스의 page table에 스택주소 추가
✓ 스택 주소를 추가한 후에, pd_copy() 함수를 통해서 부모가 가지고 있는 자식프로세스의 스택주소가 자식프로세스의 pt로 복사함
✓ Stack 영역을 할당 받을 경우, palloc_get_multiple() 함수의 리턴 값으로 VKERNEL_STACK_ADDR로 할당받을 페이지로 주소변환 후 리턴함. 단, 할당할 영역을 0으로 초기화
✓ 스택은 할당받은 VKERNEL_STACK_ADDR 주소 기준으로 밑으로 사용되기 때문에 하위 2 page를 0으로 초기화함
✓ palloc_free_multiple() 함수를 완성하여 성공적으로 page 반환을 해야함
✓ **'(2) Stack 구현' 이후, Main.c 의 main_init() 함수 내부의 sema_self_test() 함수 호출 아래의 while(1); 삭제 후 (3) 수행**

- (3) shell 프로세스의 Stack 주소 값을 프린트하기
✓ printk() 함수를 이용하여 shell_proc의 proc 구조체에 저장되어있는 stack 주소값 출력하기
✓ 올바른 Stack 주소가 찍히는지 확인
✓ Stack 주소값은 구현에 따라 다를 수 있음

○ 과제 수행 시 주의 사항

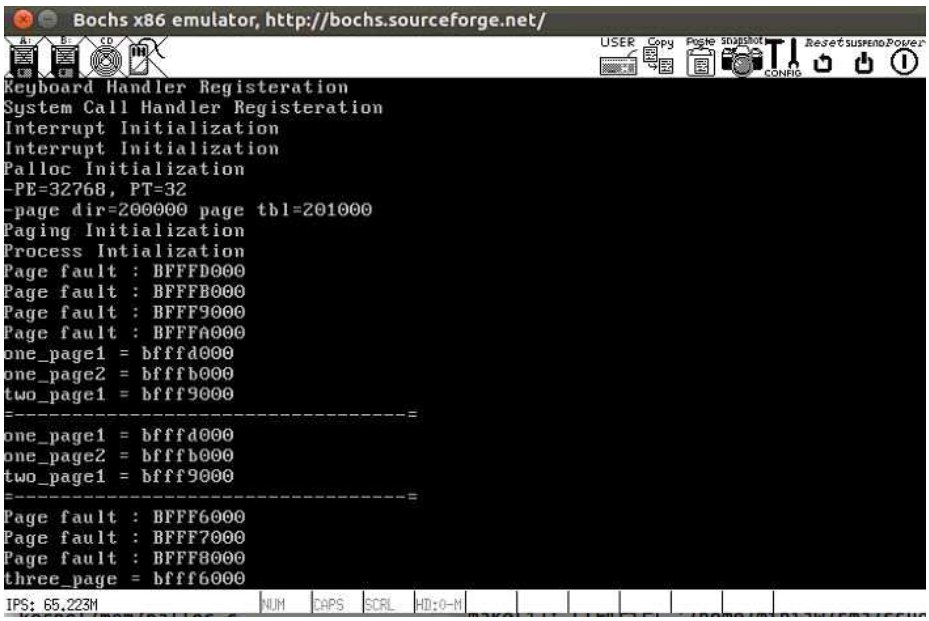
✓ 구현해야 할 함수(palloc_get_multiple(), va_to_ra(), ra_to_va(), palloc_free_multiple(), proc_create(), child_stack_reset()) 외 다른 코드는 수정 불가
✓ main_init() 함수 안의 while(1); 문은 테스트를 위한 문장이므로 과제 수행을 하면서 지워야 함
✓ page fault 처리 시 에러가 발생하는 메모리 주소가 포함되는 page 1개만 page table에 추가 할 것
- **다음에 page fault가 날 page들을 미리 page table의 항목을 채워 넣지 말 것**

✓ page fault 처리 시 에러가 발생하는 메모리 주소의 page directory 및 page table 인덱스를 계산하고 필요시 palloc()으로 할당할 것

- **메모리 주소를 하드코딩 하지 말 것**

✓ 구현 시 필요 변수 임의로 선언 및 구현되어 있는 함수들 사용가능

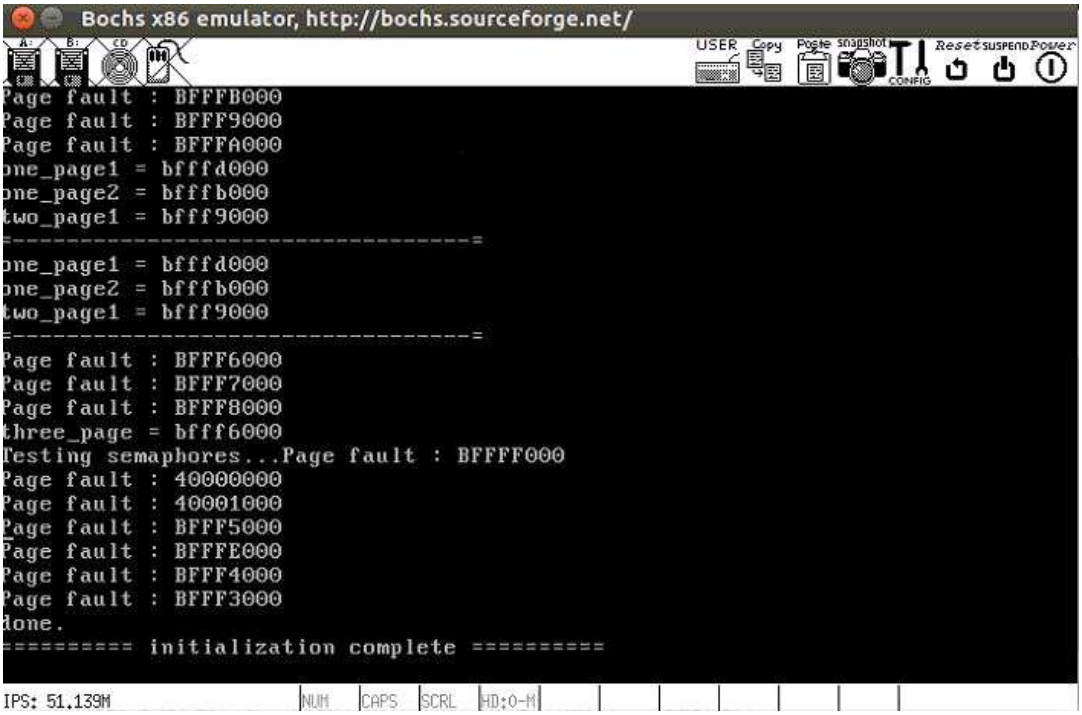
○ 과제 수행 결과



(1) Heap 구현 후

```
62      init_paging();
63      printk("%s", "Paging Initialization\n");
64
65      init_proc();
66      printk("%s", "Process Intialization\n");
67
68      intr_enable();
69
70      palloc_pf_test();
71  #ifdef SCREEN_SCROLL
72      refreshScreen();
73  #endif
74  //  while(1); // (1)
75      sema_self_test();
76      printk("========== initialization complete ==========\n\n");
77
78  while(1); //(2)
79  #ifdef SCREEN_SCROLL
80      refreshScreen();
81  #endif
82  }
83
```

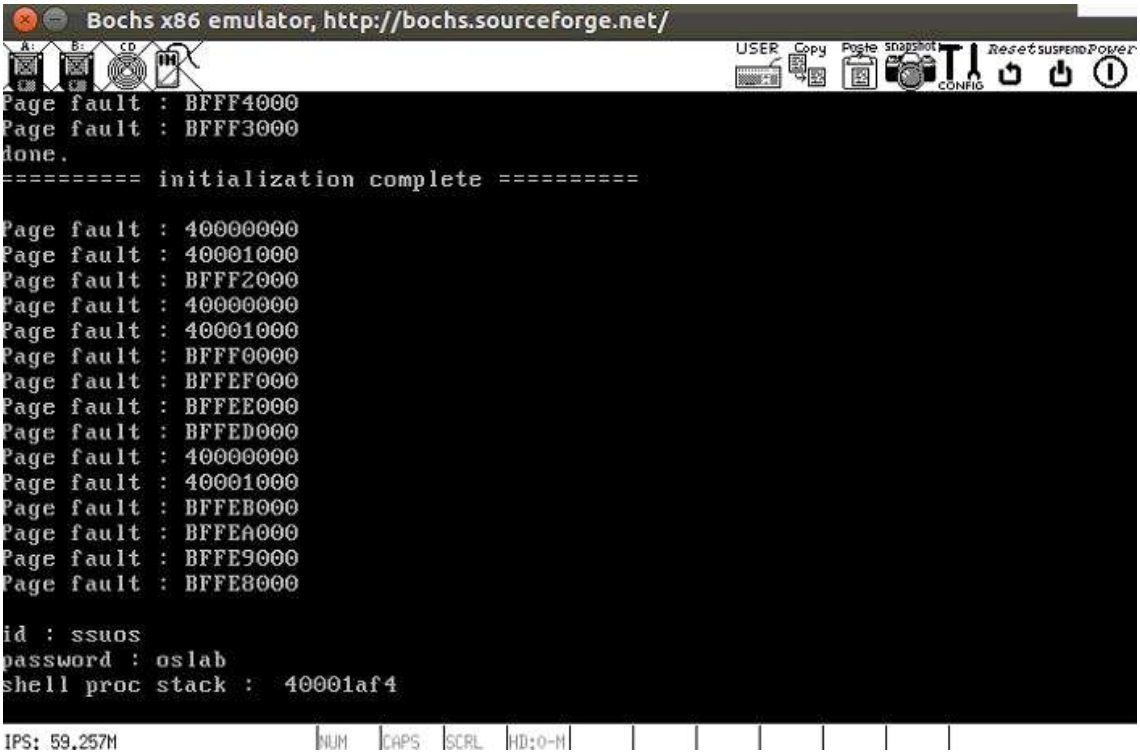
(2) Heap 구현 후 main_init 함수의 palloc_pf_test() 함수 호출 밑의 while(1); 주석처리



(3) Stack 구현 후

```
65      init_proc();
66      printk("%s", "Process Intialization\n");
67
68      intr_enable();
69
70      palloc_pf_test();
71  #ifdef SCREEN_SCROLL
72      refreshScreen();
73  #endif
74  //  while(1); // (1)
75      sema_self_test();
76      printk("========== initialization complete ==========\n\n");
77
78  //while(1); //(2)
79  #ifdef SCREEN_SCROLL
80      refreshScreen();
81  #endif
82  }
83
```

(4) Stack 구현후 main_init 함수의 sema_self_test 함수 호출 밑의 while(1); 주석처리



(5) shell process 의 Stack 주소값을 출력

○ 과제 제출

- 2018년 10월 29일 (월) 23시 59분 59초까지 과제 게시판으로 제출

- 최소 구현사항

✓ (1),(2) 구현

- 배점 기준

✓ 보고서 15%

- 개요 2%
- 상세 설계 명세(기능 명세 포함) 10%
- 실행 결과 3%

✓ 소스코드 85%

- 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
- 실행 여부 80%((1) 구현 35%, (2) 구현 35%, (3) 구현 10%)