

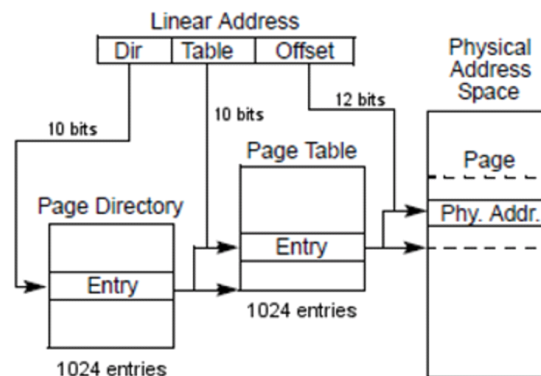
과제 #5 : Memory

○ 과제 목표

- 가상 메모리 Page Allocator 이해
- Page Fault Handler 구현
- Process 생성 시 Page directory 및 Page table 복사 구현

○ 기본 배경 지식

- 가상 메모리
 - ✓ 메모리 사용량이 늘어남에 따라 디스크의 일부를 마치 확장된 RAM처럼 사용할 수 있게 해주는 기법
 - ✓ 커널이 RAM에 적재된 메모리 블록 중 당장 쓰이지 않는 것을 디스크에 저장함으로써 사용가능한 메모리 영역을 늘림
 - ✓ 만일 디스크에 저장된 메모리 블록이 필요해지면 다시 RAM에 적재되며 다른 블록이 디스크에 저장됨
 - ✓ 프로세스의 할당된 가상 메모리는 실제 메모리 공간에 맵핑되어 RAM에 로드되어 사용됨
 - ✓ 현재 SSUOS의 가상메모리 구현은 가상 메모리주소를 page directory와 page table을 통해 실제 메모리주소로 변경하며 디스크의 SWAP 영역은 사용하지 않음
- 페이징
 - ✓ 가상 메모리를 모두 같은 크기의 블록으로 관리하는 기법
 - ✓ 가상 메모리의 일정한 크기를 가진 블록을 page라고 하며 실제 메모리의 블록을 frame라고 함
 - ✓ 가상 메모리 구현은 가상 메모리가 참조하는 page주소를 실제 메모리 frame주소로 변환하는 것
 - ✓ 가상 메모리 주소는 가상 메모리 page가 실제 메모리 frame으로 맵핑되어 있는 페이지 테이블을 통해서 실제 메모리 주소로 변경되어 메모리에 접근
 - ✓ SSUOS의 page 크기는 4KB 이며 밑의 그림과 같이 page directory와 page table로 나뉘는 2단계 페이지 테이블 구조를 사용



[그림 1] 가상 선형 주소의 변환 과정

- Page Fault

- ✓ 프로세스가 자신의 할당된 가상주소공간에는 맵핑되지 않은 메모리 페이지에 접근할 때 생기는 예외

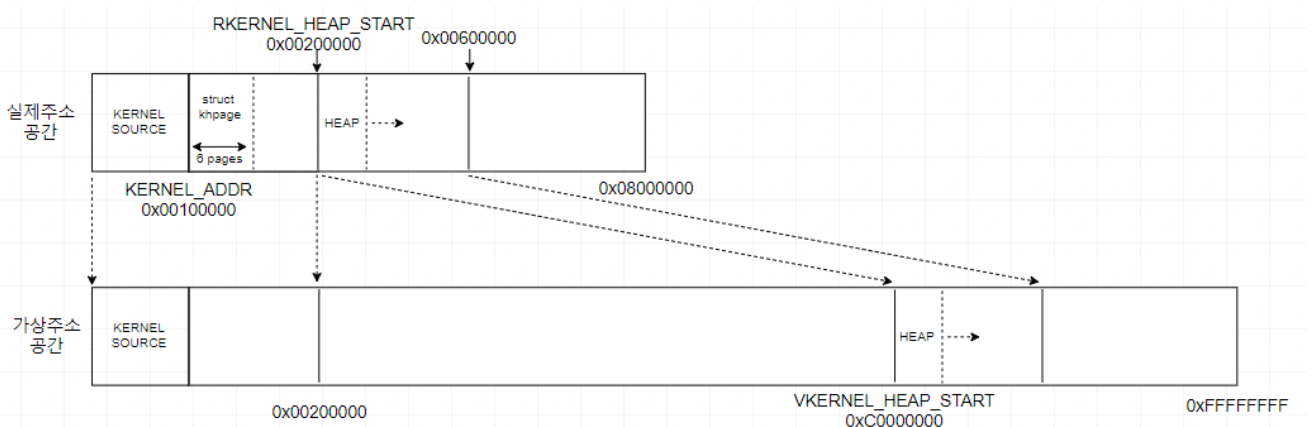
- ✓ 프로세스가 가상 메모리 주소에 접근할 때 자신의 페이지 테이블로 접근하여 실제 메모리 주소로 변경하는 중 해당 페이지가 페이지 테이블에 실제 주소로 맵핑되어 있는 정보가 없는 경우 발생
- ✓ Page fault가 발생한 페이지에 대한 맵핑 정보를 페이지 테이블에 추가해야 함
- ✓ Page fault 시 CR2 레지스터가 page fault가 발생한 32 bit 선형 주소를 포함함

○ 과제 내용

- page allocator 구현 사항

- ✓ 현재 page allocator는 paging.h에 정의되어 있는 RAM의 RKERNEL_HEAP_START(2MB)부터 6MB까지 1024개의 4KB 짜리 page를 관리함
- ✓ 1024개의 page들을 위한 구조체는 RAM의 1MB부터 6개의 page에 저장됨
- ✓ palloc() 함수는 사용가능한 page하나를 할당 해 주는 것으로 함수를 통해 가상 주소를 갖는 메모리 블록을 리턴 받을 수 있음
- ✓ palloc() 함수 에서 리턴 되는 메모리 블록의 주소는 paging.h에 정의되어 있는 VKERNEL_HEAP_START (3GB)부터 리턴 됨

- 가상 메모리 맵



[그림 2] SSUOS의 가상 메모리 맵

- ✓ 가상 메모리 주소 공간의 0 ~ 2MB까지는 실제 메모리 공간과 동일하게 맵핑되며 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
- ✓ 현재 SSUOS의 생성되는 모든 프로세스는 커널 프로세스로 가상 커널 영역의 메모리들을 모두 공유함
- ✓ SSUOS에서 생성되는 모든 프로세스는 [그림 2]와 동일한 가상 메모리 맵을 가짐

- page fault handler 구현

- ✓ 현재 SSUOS는 init_paging() 함수를 통해서 2MB까지만 page table을 초기화 하여 사용
- ✓ palloc_get_page() 함수를 사용하여 새로운 페이지를 할당 받으면 3GB 이후의 메모리 공간을 리턴하기 때문에 page table에 해당 공간의 내용이 없어서 page fault가 발생
- ✓ page fault handler는 page fault 발생 시 호출되며 page fault가 발생한 메모리 주소를 cr2 레지

스터를 통해 알 수 있음

- ✓ cr2 레지스터로 page fault가 발생 한 메모리 주소의 page directory 인덱스를 확인하여 page table 주소를 얻고 다시 page fault가 발생 한 메모리 주소에서 page table 인덱스를 확인 하여 해당 메모리가 속한 페이지의 실제 메모리 페이지 주소와 RW, PRESENT 비트를 설정
- ✓ 만일 page directory 탐색 중 page table 또한 존재하지 않으면 palloc_get_page() 함수를 사용하여 page 하나를 할당 받고 page table로 사용

- 프로세스 생성 시 page directory 및 page table 복사 구현

- ✓ 새로운 프로세스가 생성될 때는 프로세스가 사용할 page directory와 page table을 생성
- ✓ page directory와 page table로 사용할 page는 palloc_get_page() 함수를 사용하여 할당
- ✓ 생성되는 프로세스는 부모 프로세스와 동일한 가상 메모리 맵을 유지하기 위해 page directory와 page table의 내용을 부모 프로세스의 내용과 동일하게 복사
- ✓ 부모 프로세스의 page directory에서 내용이 존재하는 항목을 찾아서 page table 만들고 다시 부모 프로세스의 page table에서 내용이 존재하는 항목을 찾아서 새로 만든 page table에 추가

○ 과제 수행 방법

- (1) page fault handler 구현

- ✓ ssuos/src/kernel/mem/paging.c 의 pf_handler() 함수 구현
- ✓ cr2 레지스터에 저장되어 있는 page fault를 발생시킨 주소를 fault_addr 변수에 저장
- ✓ 해당 변수를 통해서 page directory 인덱스와 page table 인덱스를 계산
- ✓ 현재 프로세스의 page directory에 인덱스를 참고 해서 해당 page table 항목에 page fault를 발생 시킨 메모리 주소의 실제 메모리 주소를 추가
- ✓ 현재 pid 0번 프로세스는 ssuos/src/kernel/arch/Main.c 의 초기화 작업들 후 page fault를 테스트 하기 위한 palloc_pf_test 함수 호출
- ✓ palloc_pf_test는 ssuos/src/kernel/mem/palloc.c 에 구현되어 있음
- ✓ ‘(1) Page Fault Handler 구현’ 이후 Main.c의 main_init() 함수 내부의 palloc_pf_test() 함수 호출 아래의 while(1); 삭제 후 (2) 수행

- (2) 프로세스 생성 시 page directory 및 page table 복사 구현

- ✓ ssuos/src/kernel/arch/Main.c의 main_init 함수 안의 palloc_pf_test() 함수 호출 밑의 while(1); 삭제
- ✓ ssuos/src/kernel/mem/paging.c의 pd_create(), pt_copy() 함수 구현
- ✓ pd_create() 함수는 새로 생성될 프로세스의 page directory로 사용할 page를 할당 받은 후 pt_copy() 함수 호출
- ✓ 할당 받아 page directory로 사용하는 page를 실제 메모리 주소로 변환하여 리턴
- ✓ pt_copy() 함수는 source page directory에서 항목들을 검사해 PRESENT 비트가 설정되어 있는 항목들을 찾아 pt_copy() 함수를 호출
- ✓ pt_copy() 함수는 page table로 사용할 page를 새로 할당 받은 후 인자로 받은 destination directory의 인자로 받은 idx 번째 항목에 추가
- ✓ source directory에 idx번째 항목인 source page table 의 항목들을 검사하여 PRESENT 비트가 설정되어 있는 항목을 새로 할당받은 page table 복사
- ✓ ‘(2) 프로세스 생성 시 page directory 및 page table 복사 구현’ 이후 Main.c의 main_init() 함수 내부의 sema_self_test() 함수 호출 아래의 while(1); 삭제 후 (3) 수행

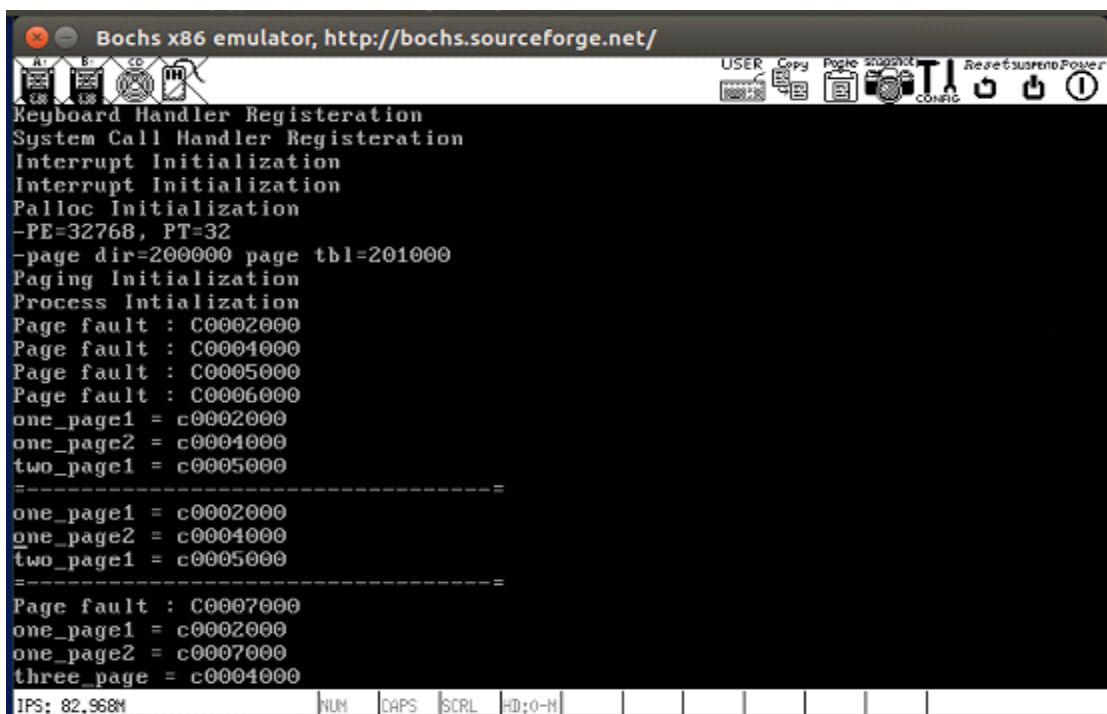
- (3) page fault handler 구현 보강

- ✓ ssuos/src/kernel/arch/Main.c의 main_init 함수 안의 sema_self_test() 함수 호출 밑의 while(1); 삭제
- ✓ 셸 프로세스가 ps 및 uname 의 명령어를 통해서 새로운 프로세스를 만들 경우에는 (1)에서 구현 했던 page fault handler를 좀 더 보완해야 함
- ✓ (1) ~ (2) 과정에서 구현할때 0번 프로세스에서 새로운 프로세스를 생성하는 경우만 생각해서 구현하였다면 0번 프로세스가 아닌 셸 프로세스에서 새로운 프로세스를 생성하는 상황에 대한 구현이 빠져있음
- ✓ 로그인 이후 셸 프로세스에서 ps 및 uname 명령어 실행 시 에러 발생 없이 원활히 수행되면 (3) page fault handler 구현 보강 완료

○ 과제 수행 시 주의 사항

- ✓ 구현해야 할 함수(pf_handler(), pd_create(), pt_copy()) 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 계산 하고 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



```
Bochs x86 emulator, http://bochs.sourceforge.net/
Keyboard Handler Registration
System Call Handler Registration
Interrupt Initialization
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=200000 page tbl=201000
Paging Initialization
Process Initialization
Page fault : C0002000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
Page fault : C0004000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
Page fault : C0005000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
Page fault : C0006000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
Page fault : C0007000
one_page1 = c0002000
one_page2 = c0004000
three_page = c0004000
IPS: 82.968M
```

(1) page fault handler 구현 후

```

65     init_proc();
66     printk("%s", "Process Initialization\n");
67
68     intr_enable();
69
70     palloc_pf_test();
71 #ifdef SCREEN_SCROLL
72     refreshScreen();
73 #endif
74 // while(1);
75
76     sema_self_test();
77     printk("==== initialization complete =====\n\n");
78
79
80 #ifdef SCREEN_SCROLL
81     refreshScreen();
82 #endif
83     while(1);
84 }

```

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

```

Bochs x86 emulator, http://bochs.sourceforge.net/
-PE=32768, PT=32
-page dir=200000 page tbl=201000
Paging Initialization
Process Initialization
Page fault : C0002000
Page fault : C0004000
Page fault : C0005000
Page fault : C0006000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
Page fault : C0007000
one_page1 = c0002000
one_page2 = c0007000
three_page = c0004000
Testing semaphores...Page fault : C0000000
Page fault : C0001000
done.
==== initialization complete =====

```

(3) PD 및 PT 복사 구현 후

```

70     palloc_pf_test();
71 #ifdef SCREEN_SCROLL
72     refreshScreen();
73 #endif
74 // while(1);
75
76     sema_self_test();
77     printk("==== initialization complete =====\n\n");
78
79
80 #ifdef SCREEN_SCROLL
81     refreshScreen();
82 #endif
83 // while(1);
84 }

```

(4) 구현 후 main_init 함수의 sema_self_test 함수 호출 밑의 while(1); 삭제

```

Bochs x86 emulator, http://bochs.sourceforge.net/
three_page = c0004000
Testing semaphores...Page fault : C0000000
Page fault : C0001000
done.
==== initialization complete =====

Page fault : C0008000
Page fault : C0009000
Page fault : C000a000
Page fault : C000B000
Page fault : C000C000
Page fault : C000D000
Page fault : C000E000

id : ssuos
password : oslab
> uname
Page fault : C000F000
Page fault : C0010000
Page fault : C0011000
Page fault : C0012000
SSUOS 0.1.02
made by OSLAB
modified by You
>
IPS: 59.426M
NUM CAPS SCRL HD:0-H

```

(5) Page Fault Handler 보강 이후 셸 프로세스에서 ‘uname’ 명령어 실행

(6) Page Fault Handler 보강 이후 셸 프로세스에서 ‘ps’ 명령어 실행

○ 과제 제출

- 2017년 10월 25일 (수) 23시 59분 59초까지 과제 게시판으로 제출

○ 최소 기능 구현

- (1) Page Fault Handler 구현
- (2) Process 생성 시 Page directory 및 Page table 복사 구현

○ 배점 기준

- 보고서 20점
 - ✓ 개요 3점
 - ✓ 상세 설계 명세(기능 명세 포함) 7점
 - ✓ 구현 방법 설명(코드 주석 포함) 8점
 - ✓ 실행 결과 2점
- 소스코드 80점
 - ✓ 컴파일 여부 10점(설계 요구에 따르지 않고 설계된 경우 0점 부여)
 - ✓ 실행 여부 70점
 - (1) Page Fault Handler 구현 20점
 - (2) Process 생성 시 Page directory 및 Page table 복사 구현 20점
 - (3) Page Fault Handler 보강 30점