# 과제 #5 : Inverted Page Table

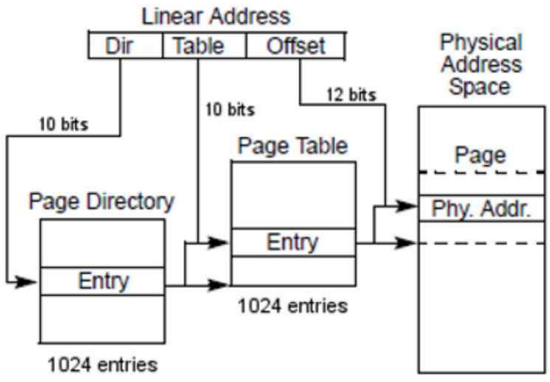
○ 과제 목표

- 가상 메모리 Page Allocator 이해
- Level Hash 구현
- 역페이지 테이블 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 실제메모리주소로 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

- 역페이지 테이블

✓ 많은 프로세스를 생성하면 페이지 테이블이 차지하는 메모리의 양이 많아지는 것을 해결하기 위한 방법

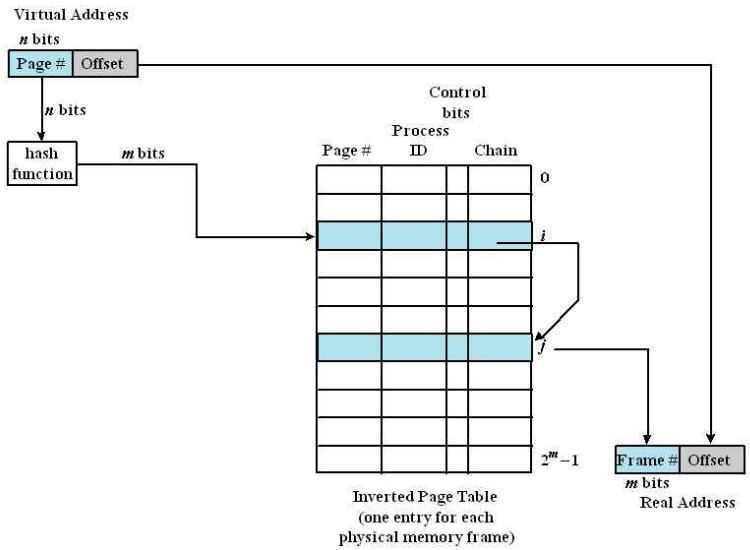
✓ 전체 시스템에서 하나의 테이블을 가지며 physical memory의 각 frame마다 하나의 entry를 할당

✓ 가상주소 중 페이지번호 부분은 간단한 해시함수(hash function)를 통해 특정 해시값(hash value)으로 사상

✓ 해시값은 역페이지 테이블에 대한 인덱스로 사용

✓ 프로세스의 수나 지원되는 가상 페이지 수와 상관없이 주기억장치의 일정 부분만이 테이블 저장에 쓰임

✓ 하나 이상의 가상주소들이 동일한 해시테이블 항목으로 사상될 수 있기 때문에, 오버플로를 관리하기 위한 연결(chaining) 기법이 사용




[그림 2] 역 페이지 테이블 구조

- Level Hash

✓ 메모리 쓰기 성능 향상을 위한 해싱 기법

✓ Top-level과 Bottom-level의 2-level hash table로 구성되며 Top-level은 Bottom-level의 2배 크기 테이블을 가짐

✓ 2개의 Top-level 버킷은 하나의 Bottom-level 버킷을 공유하며, 전체적으로 완전 이진 트리의 일부분 형태의 구조가 됨

✓ 하나의 버킷은 4개의 슬롯을 가짐

✓ 2개의 해시 함수를 사용하여 해시 함수에 대응하는 Top-level의 위치 중 비어있는 아무 위치에 아이템 삽입

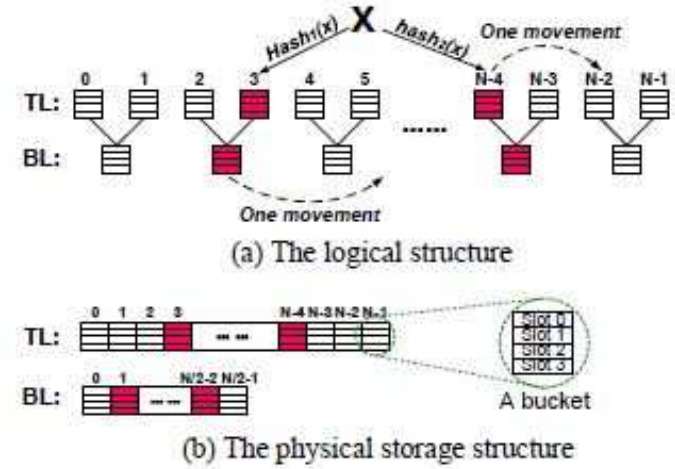
✓ 해시 함수에 대응하는 모든 Top-level이 가득 차있다면 Top-level에 대응하는 Bottom-level의 아무 위치에 아이템 삽입

✓ 삽입할 위치의 모든 슬롯이 가득 차있다면 Top-level에 이미 삽입된 아이템 중 하나를 다른 Top-level로 이동 후 아이템 삽입

✓ Top-level to Top-level 이동이 불가능하다면 Bottom-level에 이미 삽입된 아이템 중 하나를 다른 Bottom-level로 이동 후 아이템 삽입

✓ 삭제 연산 시 해시 함수에 대응하는 위치의 Top-level 버킷 2개와 Top-level 버킷에 대응하는 각

각의 Bottom-level 버킷, 총 4개의 버킷을 탐색하여 아이템을 찾아 삭제



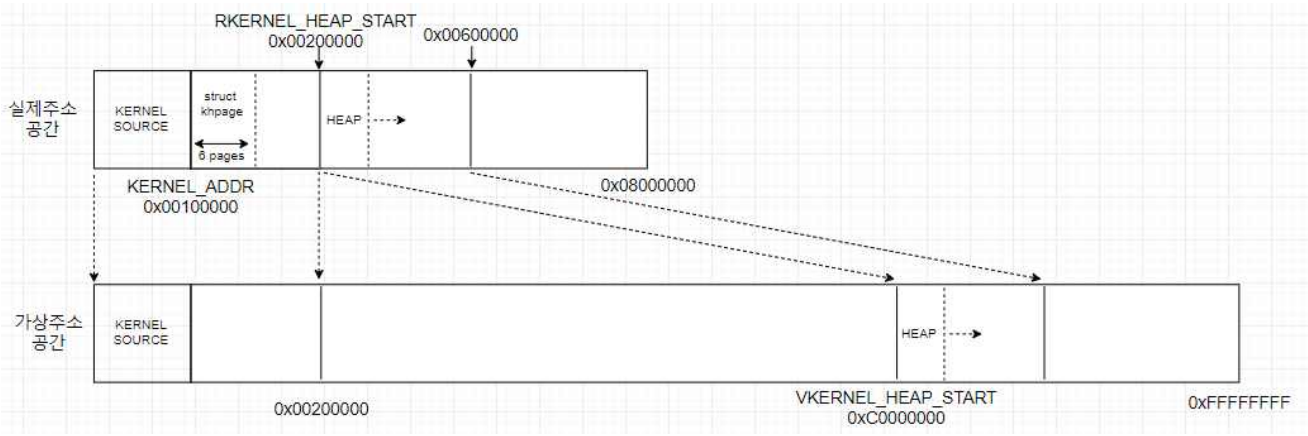

[그림 3] Level Hash의 구조

- ✓ [그림 3]에서 X는 Hash Table에 들어갈 원소
- ✓ X에 Hash1(X), Hash2(X) 두가지 해시 함수 적용
- ✓ 해시함수가 적용된 3번, N-4번 인덱스의 Top-level 버킷 중 아무 위치에 삽입
- ✓ 3번, N-4번 인덱스의 Top-level 버킷이 모두 차 있다면 대응하는 Bottom-level 버킷 중 아무 위치에 삽입
- ✓ 대응하는 Bottom-level 버킷도 모두 차 있다면 3번, N-4번 인덱스의 기존 아이템 중 하나를 다른 버킷으로 이동 시도
- ✓ 이때 이동할 원소의 다른 위치는 해당 값에 해시함수를 적용한 위치여야 함
- ✓ Top-level의 모든 원소가 이동 불가하다면 대응하는 Bottom-level에서 이동 시도
- ✓ Bottom-level에서도 이동이 불가능하다면 Resizing 진행

○ 과제 내용

- 페이지 테이블 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 4] 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

○ 과제 수행 방법

- (1) Level Hash 구현

- ✓ **Level Hash**의 삽입, 삭제 기능 구현
- ✓ 기존 **Level Hash**의 **resizing** 기능(확장 및 축소)은 고려하지 않음
- ✓ **Top-level**의 버킷 개수(**capacity**)는 **256**개로 고정
- ✓ 아이템 삽입 여부를 표시하기 위해 **token** 사용, 아이템이 존재하면 **1**, 존재하지 않으면 **0**
- ✓ 아이템 삽입 시 **2**개의 해시 함수를 사용하여 대응하는 **2**개의 **Top-level** 버킷에서 첫 번째 해시 함수 결과의 슬롯을 차례대로 확인, 이후 두 번째 해시 함수 결과의 슬롯을 차례대로 확인하여 비어있는 위치에 저장
- ✓ **Top-level** 버킷이 가득 차있다면 대응하는 **Bottom-level** 버킷에서 첫 번째 해시 함수 결과의 슬롯을 차례대로 확인, 이후 두 번째 해시 함수 결과의 슬롯을 차례대로 확인하여 비어있는 위치에 저장
- ✓ 삽입 가능한 위치가 모두 차있다면 **Top-level** 버킷 아이템 중 하나를 다른 **Top-level** 버킷으로 이동 후 아이템 삽입
- ✓ **Top-level** 버킷 아이템 중 이동 가능한 아이템이 없다면 **Bottom-level** 버킷 아이템 중 하나를 다른 **Bottom-level** 버킷으로 이동 후 아이템 삽입
- ✓ 아이템 삽입이 성공하면 해당 슬롯의 **token**을 **1**로 변경
- ✓ 아이템 삭제 시 테이블 인덱스와 **key**값으로 탐색 후 해당 슬롯의 **token**을 **0**으로 변경

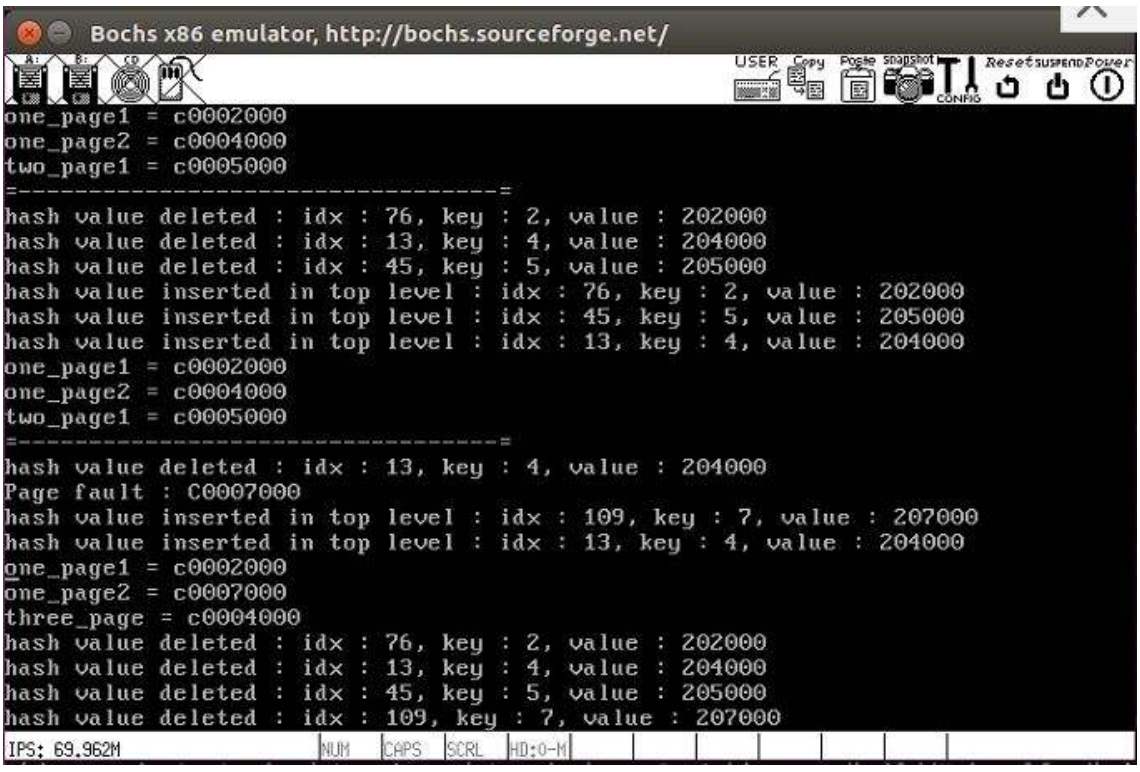
- (2) Level Hash를 사용한 Inverted Page Table 구현

- ✓ 기존의 **page** 할당 방법은 그대로 사용
- ✓ 할당받은 **page**의 가상 주소를 사용하여 해시 함수를 계산, 계산된 값을 테이블의 인덱스로 사용
- ✓ **page** 인덱스는 **key**로, **page**의 실제 주소는 **value**로 사용
- ✓ **page**를 할당받은 후 **page**의 인덱스와 실제 주소를 구현된 **Level Hash Table**에 삽입
- ✓ 삽입 성공 시 삽입 성공 문구, 삽입된 테이블의 인덱스, **key, value**값 출력
- ✓ **page** 해제 시 **hash table**에서 대응하는 **page** 정보를 **key**를 사용하여 찾고 삭제
- ✓ 삭제 성공 시 삭제 성공 문구, 삭제된 테이블의 인덱스, **key, value**값 출력

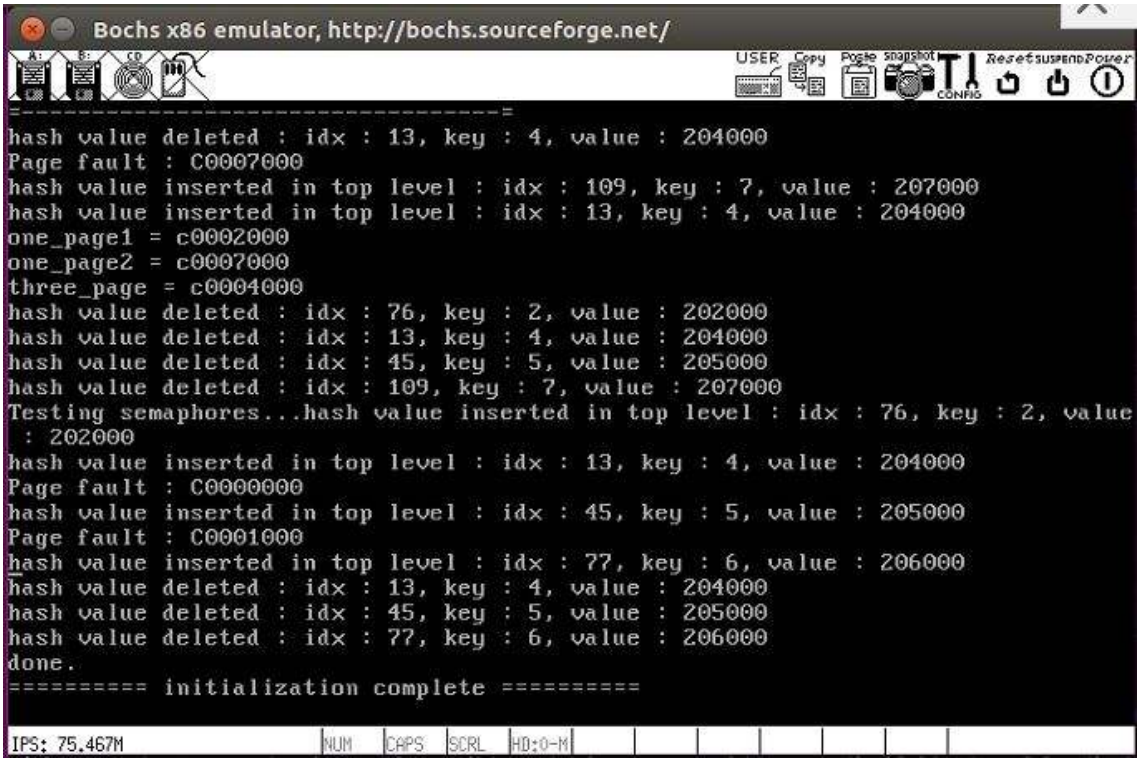
○ 과제 수행 시 주의 사항

- ✓ 구현해야 할 파일(palloc.c, hashing.c, hashing.h) 외 다른 코드는 수정 불가
- ✓ 기존의 페이지 할당 방식은 수정하지 않으므로 페이지 할당 부분 코드 삭제 불가
- ✓ Hash Table의 value로 사용할 실제 주소 변환을 위해 VH_TO_RH() 함수 사용
- ✓ main_init() 함수 안의 while(1); 문은 테스트를 위한 문장이므로 과제 수행을 하면서 지워야 함
- ✓ 구현 시 필요 변수 임의로 선언 및 구현되어 있는 함수들 사용가능

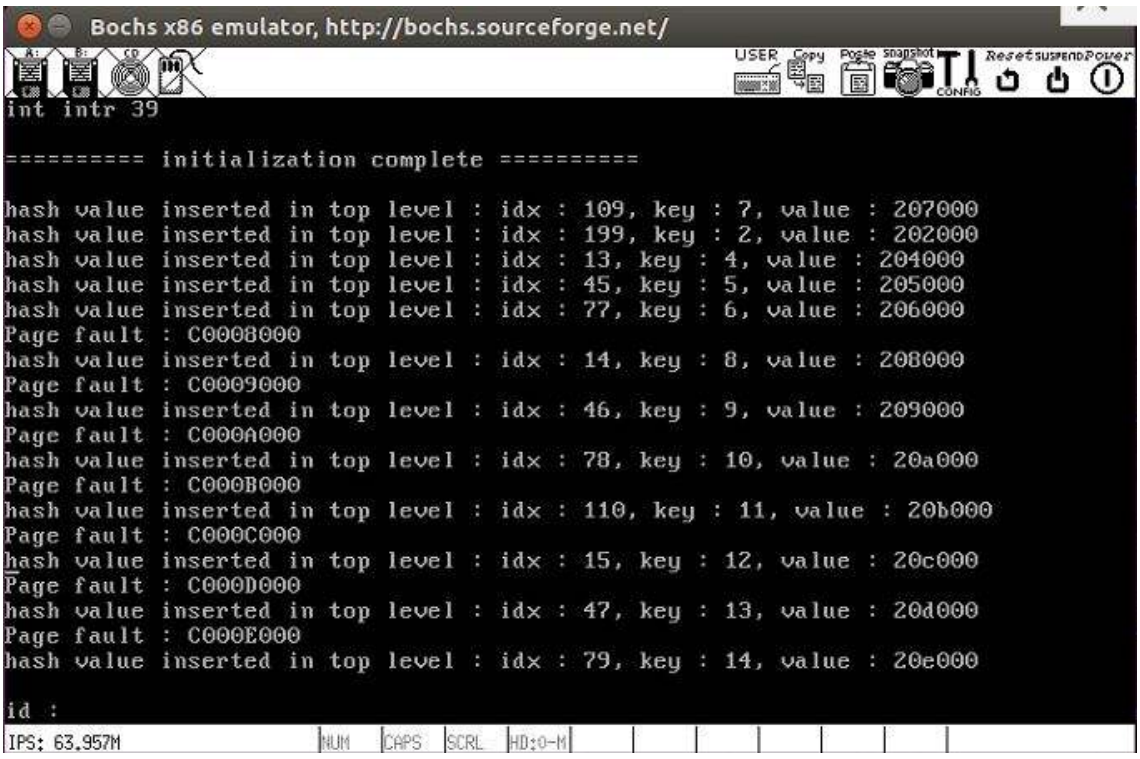
○ 과제 수행 결과



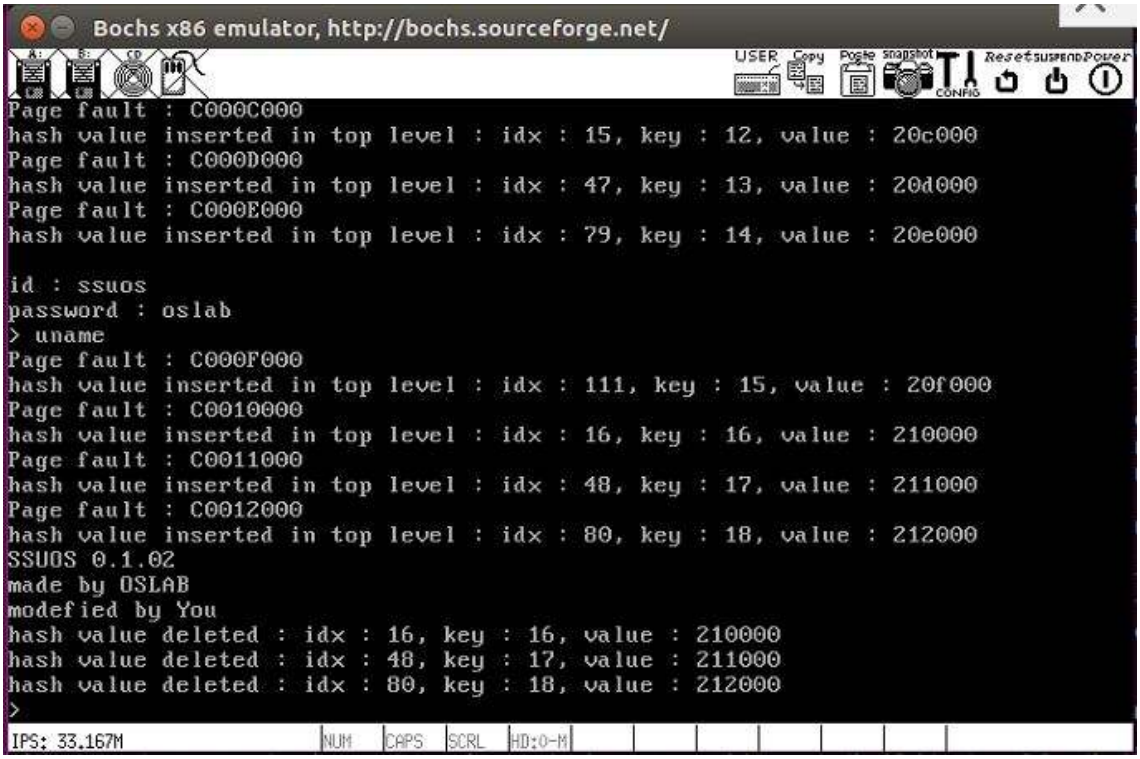

[그림 5] Inverted Page Table 구현 직후 실행결과



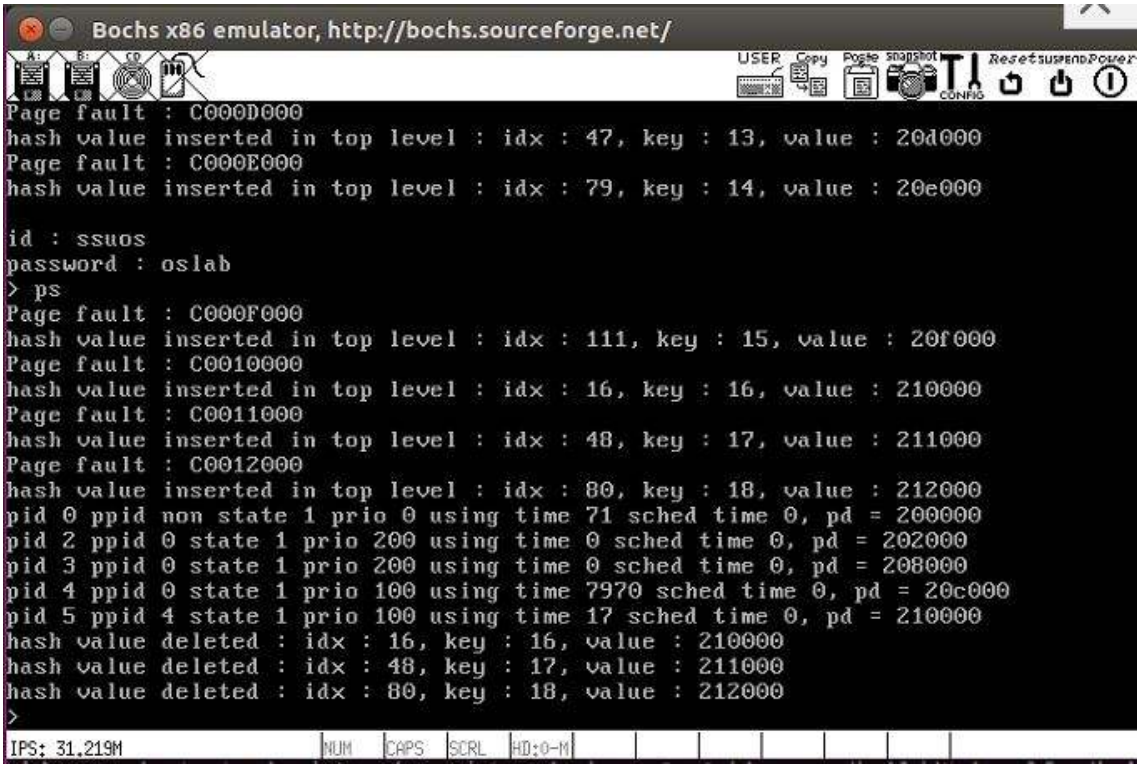

[그림 6] 과제 구현 후 main_init()에서 첫 번째 while(1)문 제거 후 실행결과




[그림 7] 과제 구현 후 main_init()에서 두 번째 while(1)문 제거 후 실행결과




[그림 8] while(1)문 제거 및 id, password 입력 후 uname 커맨드 입력 결과




[그림 9] while(1)문 제거 및 id, password 입력 후 ps 커맨드 입력 결과

```
    palloc_pf_test();
#ifdef SCREEN_SCROLL
    refreshScreen();
#endif
    while(1); // OS_P5 구현 후 삭제 필요

    sema_self_test();
    printk("========== initialization complete ==========\n\n");


#ifdef SCREEN_SCROLL
    refreshScreen();
#endif
    while(1); // OS_P5 구현 후 삭제 필요
```

[그림 10] 과제 진행 중 해당 while(1)문 주석 처리 필요

○ 과제 제출

- 2019년 11월 5일 (화) 23시 59분 59초까지 과제 게시판 및 이메일로 제출

○ 필수 기능 구현

- (1) Level Hash의 삽입 기능 구현
- (2) Level Hash의 삭제 기능 구현

○ 기타 기능 구현

- (1) Level Hash의 삽입 기능 중 이동 기능 구현
- (2) Level Hash를 이용한 Inverted Page Table 구현

○ 배점 기준

- 보고서 10점
  - ✓ 개요 2점
  - ✓ 상세 설계 명세(기능 명세 포함) 5점
  - ✓ 실행 결과 3점

- 소스코드 90점
  - ✓ 컴파일 여부 5점(설계 요구에 따르지 않고 설계된 경우 0점 부여)
  - ✓ 실행 여부 85점
    - (1) Level Hash 삽입 기능 구현 15점
    - (2) Level Hash 삽입 기능 중 이동 기능 구현 20점
    - (3) Level Hash 삭제 기능 구현 15점
    - (4) Level Hash를 이용한 Inverted Page Table 구현 35점