**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 한솔 / 20171705

개발 기간 : 11.06 ~ 11.26

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**페이지를 필요할 때마다 로드하는 기능, 그리고 메모리가 부족할 때 디스크에 잠시 덜 사용되는 페이지를 저장하고 그 메모리를 이용하는 기능을 구현한다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Page Table & Page Fault Handler

**프로그램 로드시에 모든 데이터를 메모리에 올리지 않고, 필요할 때마다 로드할 수 있는 기능을 담당한다.**

* 1. Disk Swap

**더 이상 로드할 수 있는 메모리가 없을 떄, 디스크의 특정 영역에 메모리에 있는 데이터를 옮겨두는 역할을 한다.**

* 1. Stack Growth

**처음에 할당된 스택영역을 넘어가는 크기의 영역을 할당하려할 때, 스택 영역을 추가로 할당하는 역할을 한다.**

* 1. **개발 내용**
* 아래 항목의 내용만 서술
  1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

**page fault는 접근하려는 가상주소에 해당하는 페이지가 프레임에 할당되어있지 않기 때문에 발생한다. page fault가 발생하면 page\_fault함수가 실행되어 이를 handling한다. 우선 stack 영역에 할당해야 하는 주소에서 page fault가 발생한 경우에는 단순히 stack 영역에 그 페이지를 할당해준다. stack영역에서 발생한 경우가 아니고 다른 유저 영역에서 발생한 경우에도 페이지를 할당해주는데, 할당가능한 frame이 존재하지 않는다면 clock\_algorithm을 통해 frame을 선정해서 swap disk로 evict한 후에 그 frame에 새롭게 페이지를 할당해준다.**

* 1. Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술

**할당할 수 있는 frame이 없는 경우에 page replacement algorithm을 사용해서 evict할 frmae을 선정하는데, frame을 하나씩 확인하면서 accessed bit를 확인한다. 1인 경우 이를 0으로 수정하고 다음 frame을 확인한다. 모든 frame을 확인한 경우에는 다시 처음부터 확인하는데, accessed bit가 0인 경우에 그 frame을 evict할 frame으로 선정한다.**

* 1. Stack growth 구현 시 stack 확장 여부를 판단할 수 있는 방법에 대해 서술

**page fault가 발생한 주소가 PHYS\_BASE에 해당하는 주소로부터 0x800000밑의 주소 내에 해당하거나, esp보다 큰 주소를 가진 경우에 stack을 확장할 수 있다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

**11.06 ~ 110.8 page table**

**11.09 ~ 11.10 page fault handler**

**11.11 ~ 11.18 disk swap**

**11.19 ~ 11.26 stack growth**

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드

**page table을 추가하기 위해서, page.c파일을 추가한다. 여기서 supplemental page table entry를 추가하는 부분, 삭제하는 부분과 hash자료구조를 초기화할 때 필요한 함수를 정의한다. page replacement를 위해서 frame.c파일을 추가한다. 여기서는 frame table을 관리하는데, 여기서 stack growth등을 위해서 page를 할당할 때는 frame\_allocate함수를 사용한다. 그리고 swap disk에 swap할 프레임을 선정할 때는 clock\_next함수와 free\_frame함수를 사용한다. 또한 이 파일에서 frame을 초기화하고, 새로운 프레임을 frame자료구조에 넣는 함수, 프레임을 free하는 함수 등을 정의한다. disk swap을 구현하기 위해서 swap.c파일을 추가하는데, 이 파일은 swap의 사용가능한 영역을 판단하기위한 bitmap 자료구조를 포함하고, 이를 초기화하는 함수, destroy하는 함수, 실제로 disk에 swap하는 함수, 다시 memory로 load하는 함수 등이 필요하다.**

* + 수정하거나 추가해야 하는 자료구조

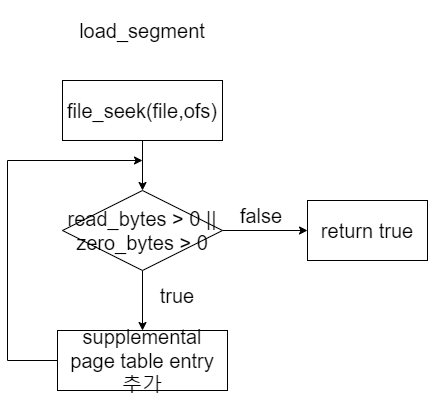
**thread.h 파일의 struct thread자료구조에서 struct hash를 추가해야한다. 이는 supplemental page table을 위한 것이다. 그리고 page.h 파일에서 struct spt\_e라는 구조체를 추가한다. 이는 직접 정의한 구조체로 supplemental page table에 추가되는 entry이다. frame.h 파일에서 struct frame\_e 구조체가 추가되어야한다. 이 또한 직접 정의한 구조체이고, frame table에 추가되는 entry이다. swap.c 파일에서 static struct bitmap 를 추가한다. 이는 swap disk의 빈 공간을 나타내는 bitmap 자료구조이다. frame.c 파일에서 struct list frame\_list를 추가해야한다. 이는 frame entry를 포함하는 list이고, 여기서 할당된 모든 frame들을 관리한다. struct lock frame\_lock자료구조도 필요하다. 이는 frame list에 entry를 추가, 삭제 등을 할 때 동기화하기 위함이다. struct list\_elem\* clock\_ptr 자료구조도 필요한데, 이는 frame을 evict하고 나서 다음에 check할 frame을 기억하기 위함이다.**

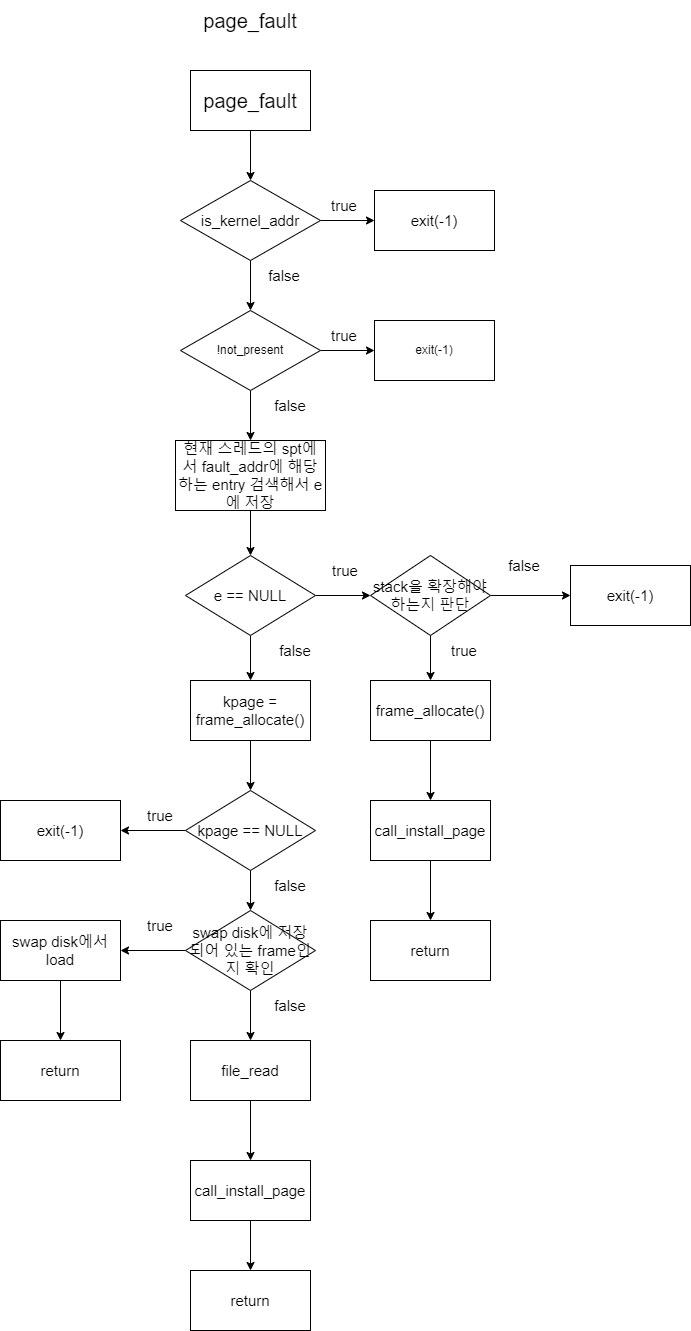
* + 수정하거나 추가해야 하는 함수

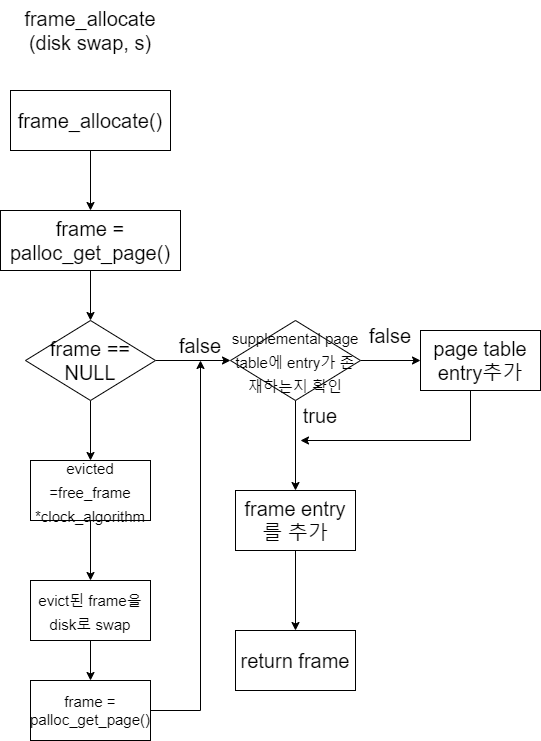
**page fault hanlder를 구현하기 위해서 exception.c의 page\_fault함수를 수정해야한다. 기존에 단순히 kernel영역에서 접근한 경우에 exit을 하는 것에서, 페이지를 할당해야 하는지, stack 영역인지 판단하고 실제로 할당하도록 수정해야한다. process.c의 load\_segment함수에서 미리 모든 파일을 메모리에 로드하지 않고, add\_spte함수를 이용해서 supplemental page table 자료구조에만 entry를 추가해둔다. setup\_stack함수에서도 frame\_allocate함수를 이용해서 page를 할당하도록 수정한다. 또한 process.c파일에서 call\_install\_page함수를 추가한다. 이는 할당한 frame을 유저의 페이지와 연결하는 함수이다.**

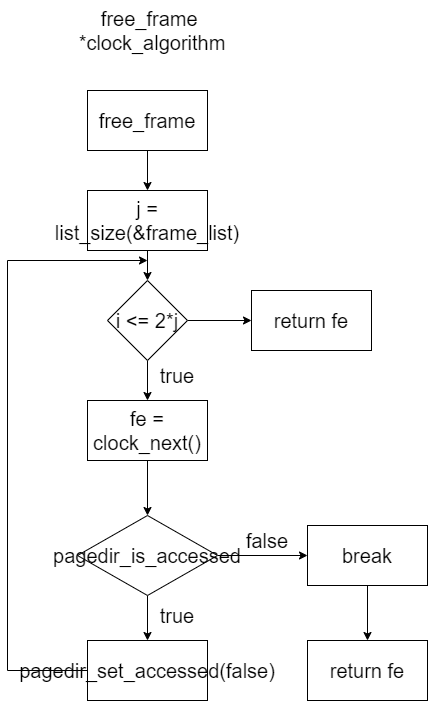
1. **연구 결과**
   1. **Flow Chart**

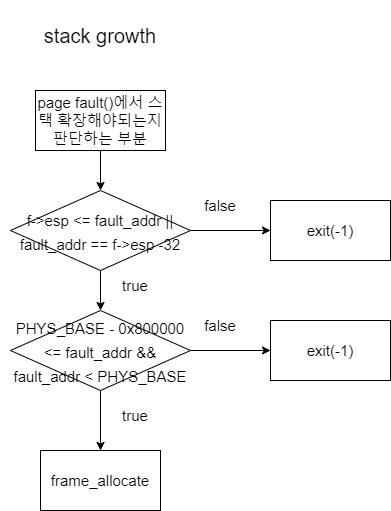
* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성



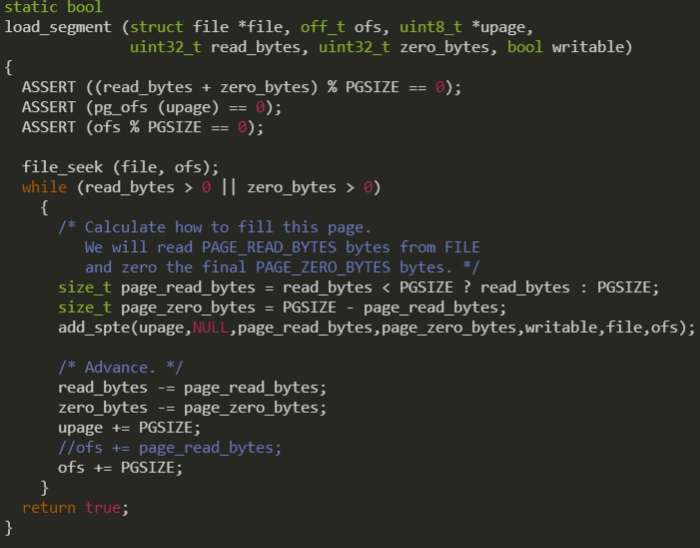




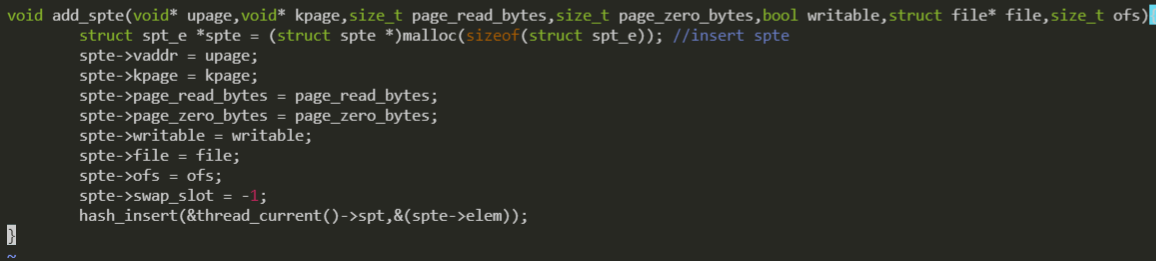


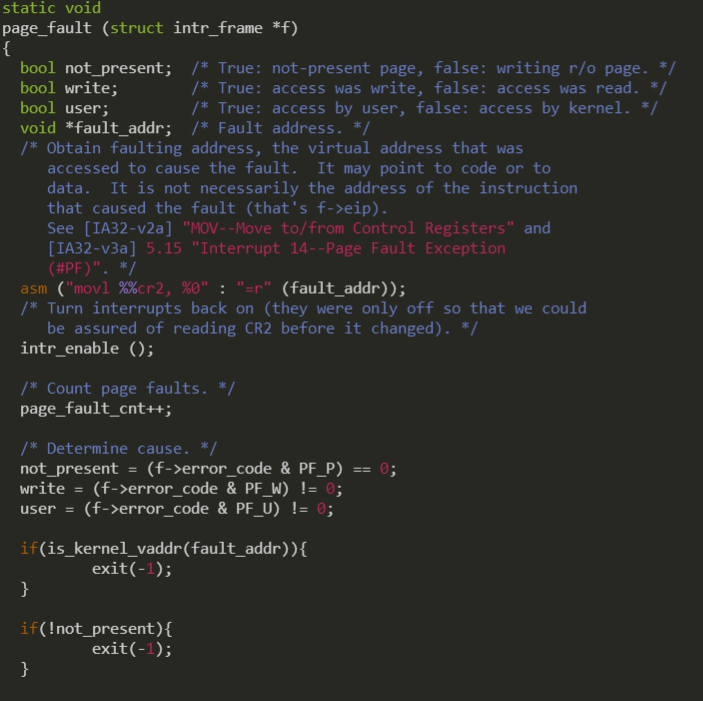


* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

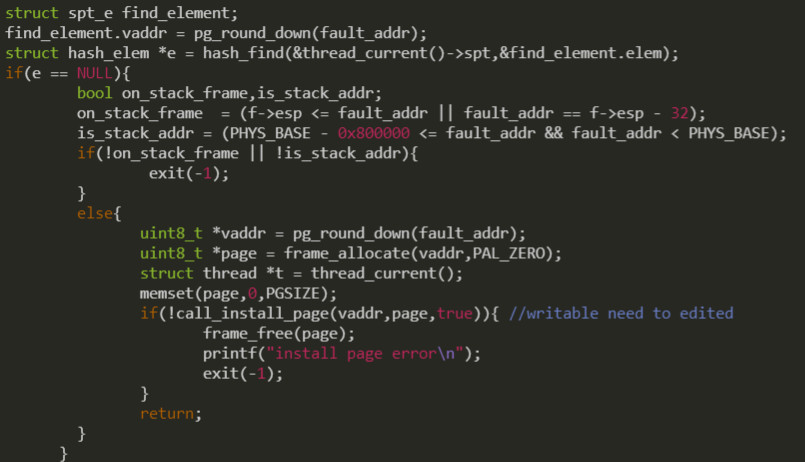
****

**파일을 로드할 때, 미리 모든 파일을 메모리에 올려놓지 않고, add\_spte함수를 이용해서 supplemental page table entry만 추가해둔다. 한 페이지크기를 넘어가는 경우에는 반복문을 통해서 여러 페이지를 할당한다.**

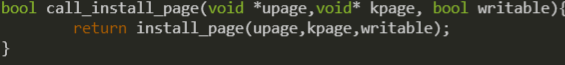
**  
add\_spte함수는 page.c에서 위와 같이 정의하였다. 이는 입력받은 upage, kpage등의 정보를 이용해서 현재 스레드의 supplemetal page table을 나타내는 hash 자료구조에 entry를 선언해서 추가해주는 함수이다.**

****

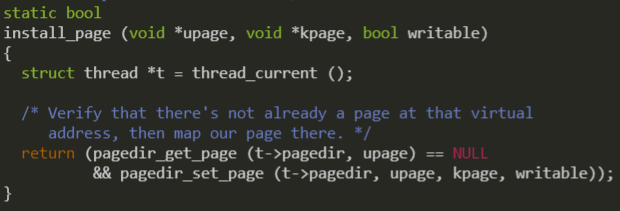
**page fault가 발생했을 때 이를 handling하기 위해서, page fault 발생시 실행되는 page\_fault함수를 수정한다. 위와 같이 접근한 주소가 커널 주소인 경우에는 exit(-1)을, 또한 not\_present error가 아닌 경우에도 exit(-1)을 호출한다.**

****

**그 다음 나머지 경우들에 대해서, fault\_addr의 페이지 시작주소를 알기 위해서 pg\_round\_down함수를 이용한다. 이는 주소를 입력받아서 이 페이지의 시작주소를 반환해주는 함수이다. 이렇게 얻은 페이지의 주소로 supplemental page table에서 검색을 한다. entry가 존재하지 않는 경우에는 stack growth를 해야할지 판단한다. esp보다 fault\_addr이 크거나, esp-32와 같아야한다. 이유는 PUSHA instruction에서 32바이트를 한번에 esp에 넣기 때문이다. 이 조건을 만족해야하고, 또한 스택이 될 수 있는 최대 크기는 0x800000이므로 PHYS\_BASE로부터 0x800000만큼 작은 영역내로 fault\_addr가 포함되는지 확인한다. 이 두 조건을 만족하지 않을 시에는 exit(-1)통해서 종료한다. 만족하는 경우에는 stack growth를 해야한다. frame\_allocate함수로 frame을 할당하고, 이렇게 할당한 frame을 현재스레드의 pagedir에서 연결해야하므로 call\_install\_page를 호출한다. frame\_allocate에 대해서는 밑에서 설명한다. 그리고 call\_install\_page는 다음과 같다.**

****

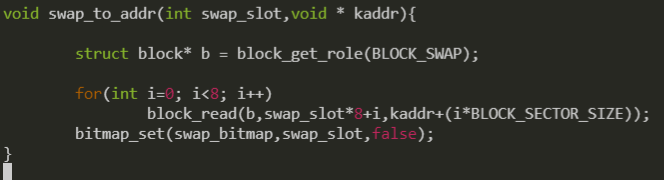
**이 함수는 process.c파일에서 install\_page를 실행한다. install\_page는 다음과 같다.**

****

**이 함수는 현재 스레드의 pagedir에 입력받은 유저가상주소가 없음을 확인하고, frame을 해당 유저주소에 연결한다.**

****

**만약 spt hash에서 찾은 entry가 존재할 경우에는, 위와 같이 frame\_allocate함수로 프레임을 할당한다. 프레임을 할당할 수 없는 경우에는 exit(-1)으로 종료한다. 할당한 경우에는 찾은 supplemental page table entry를 확인하여 disk에 swap되어있는지를 체크한다. swap되어 있는 경우에는 swap\_to\_addr함수를 이용해서 swap disk에서 memory로 로드한다. 그 후 call\_install\_page를 이용해서 새롭게 할당해서 데이터를 가져온 frame을 pagedir에 연결한다.**

****

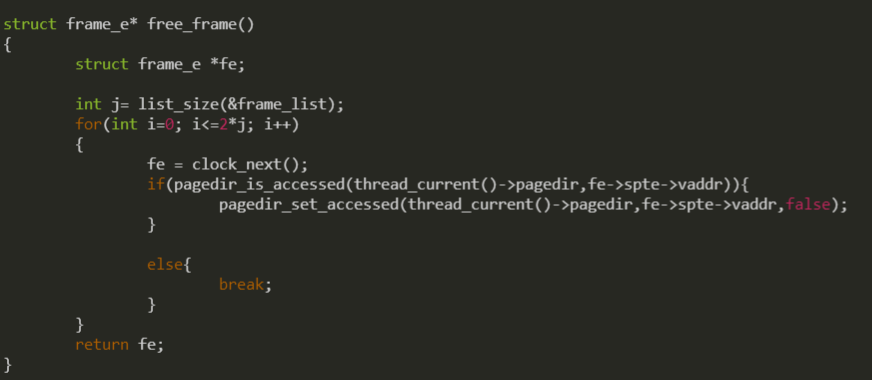
**swap\_to\_addr함수는 swap.c에서 정의하였고, 위와 같다. 하나의 sector는 512bytes이므로 하나의 페이지는 8개의 sector에 저장되어있다. 이를 위해서 block\_read를 8번 호출하여 데이터를 페이지로 읽어들인다. 그 후 swap disk의 해당 slot이 비어있음을 나타내기위해서 bitmap을 false로 set해준다.**

**swap disk에 저장되어있지 않은 경우에는 파일을 새롭게 읽어야한다. 이를 위해서 file\_read함수를 이용한다. file\_read통해서 kpage에 데이터를 읽은 후에, 마찬가지로 call\_install\_page를 호출해서 현재 thread의 pagedir에 frame을 연결한다.**

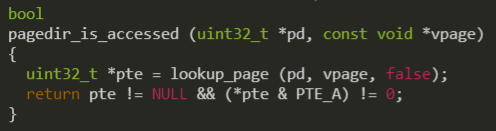
**frame\_allocate함수는 다음과 같다.**

****

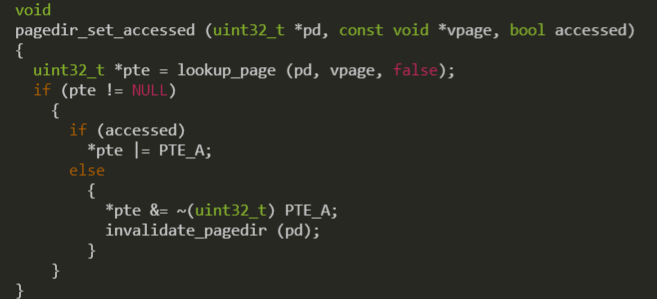
**frame\_allocate 함수는 frame.c파일에 정의되어있다. 이 함수는 우선 palloc\_get\_page함수를 이용해서 프레임을 할당한다. frame이 할당되지 않을 경우에는 palloc\_get\_page함수는 NULL을 반환하는데, 이 경우에는 free\_frame함수로 clock\_algorithm을 이용해서 frame을 하나 evict해줘야한다.**

****

**free\_frame함수는 위와같다. 이는 frame.c에서 정의되어있다. 이 함수는 frame list에 연결된 순서대로 접근을해서 해당 frame의 accessed bit를 체크한다. 이는 pagedir\_is\_accessed함수를 이용한다. set된 경우에는 이를 false로 바꿔주고 다음 프레임을 확인한다. 이렇게 확인을 하다가 accessed bit가 false인 프레임을 찾으면 이를 반환한다. 이때 clock\_ptr는 해당 프레임을 가리키고 있게된다.**

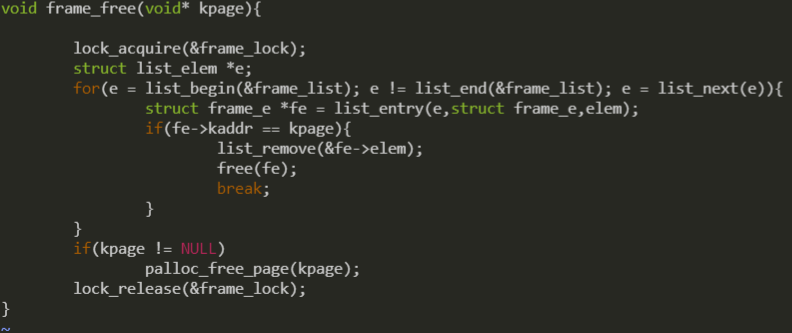
****

**pagedir\_is\_accessed함수는 page directory에서 page table entry를 찾은 후에 그것의 accessed bit를 확인하는 함수이다.**

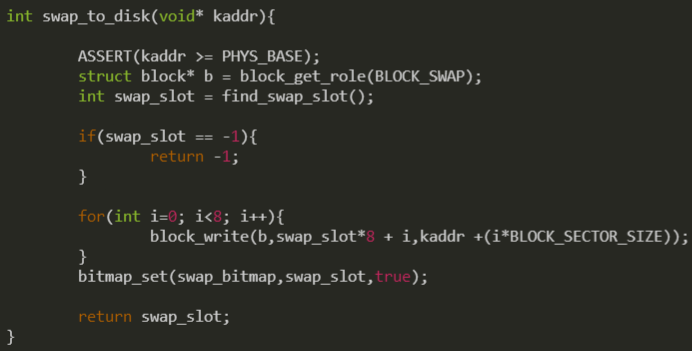
****

**pagedir\_set\_accessed함수는 위와 같이 page directory에서 page table entry를 찾아서 그것의 accessed bit를 입력받은 값으로 설정해주는 함수이다.**

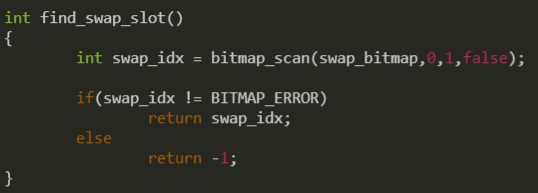
**이렇게 free\_frame을 통해서 evict할 frame을 선택하면 frame\_allocate함수에서는 이 frame을 swap\_to\_disk함수를 통해서 swap disk에 저장한다. 그리고 pagedir의 해당 user address를 clear해주고, 프레임을 free시킨다. 이때 frame\_free함수를 이용한다.**

****

**frame\_free함수는 위와 같다. frame\_list를 처음부터 탐색해서 원하는 kernel address를 저장하고 있는 frame entryf를 찾은후에 frame\_list에서 remove하고 free해주는 역할을 한다.**

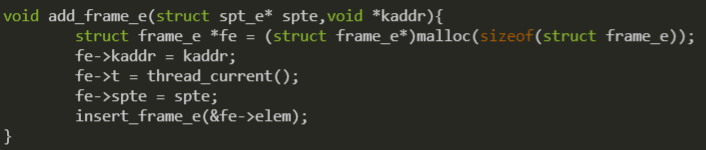
****

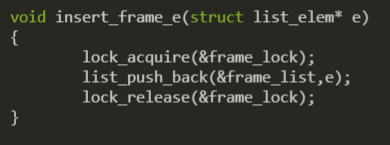
**swap\_to\_disk함수는 위와 같다. find\_swap\_slot을 통해서 swap disk의 빈 slot을 확인하고 이 slot에 data를 write한다. 하나의 페이지는 8개의 sector에 저장되어야한다. 하나의 sector크기의 8배이기 때문이다.**

****

**find\_swap\_slot함수는 bitmap에서 false로 설정된 bit를 찾아서 반환한다.**

**frame\_allocate함수에서는 이렇게 frame\_free함수를 완료하고나서, 적어도 하나의 frame이 할당가능하게 되므로 palloc\_get\_page를 이용해서 프레임을 할당한다. 이렇게 할당한 프레임을 supplemental page table와 frame table에 추가해주고 이 프레임을 반환한다. 프레임을 프레임 list에 추가할 때는 add\_frame\_e라는 함수를 이용하는데 이는 다음과 같이 정의된다.**

****

****

**add\_frame\_e함수는 새로운 frame\_e 구조체를 할당해서 kernel address와 현재 스레드 그리고 frame과 연결된 supplemental page table 정보를 저장한 후에 insert\_frame\_e함수를 실행한다. insert\_frame\_e함수는 frame\_list에 생성한 frame entry를 집어넣는 함수이다.**

* 1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부

