**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용

이름 / 학번 : 강민석 / 20161559

개발 기간 : 2020.12.19 ~ 2020.12.22

1. **개발 목표**

virtual memory를 구현하고 이에 따라 page table 관리, swapping(in & out), stack growth 등을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Page Table & Page Fault Handler

demand paging을 구현할 수 있다. page fault가 발생했을 때 load해야 할 페이지가 유효한지 검사하고 이에 따라 메모리에 올릴 수 있다.

* 1. Disk Swap

swap영역으로부터 메모리에 가져오고, frame을 다른 process에 할당하기 위하여 swap out 등을 구현한다.

* 1. Stack Growth

기존 pintos는 stack 영역에 4KB 페이지만 할당되어 있다. 스택이 성장됨에 따라 추가적으로 공간을 할당할 수 있도록 기능을 수정한다.

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

아직 메모리에 올라와 있지 않은 가상 주소를 참조하게 될 경우 page fault가 발생한다. 만약 process의 유효한 address일 경우 메모리에 load하고, 유효하지 않은 address일 경우 에러를 출력한다.

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

이번 프로젝트에서는 second chance clock algorithm을 사용하였다. 기본적으로 각 page는 참조될 때마다 1로 reference bit이 세팅되고, victim은 어느 한 page를 가리킨다. 만약 가리키고 있는 page의 reference bit이 1일 경우 0으로 세팅하고 다음 page를 검사하고, 0일 경우 swap out하여 free frame을 반환한다.

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

stack은 PHYS\_BASE(0xC0000000)으로 부터 아래로 자라게 된다. 따라서 연속적인 공간이여야 하므로, (1) 이전 공간이 이미 할당되어 있는지 확인한다. 또한 스택도 무한정 자랄 수 없으므로, 8MB제한을 둠으로써 해당 stack이 0xC0000000 – 8MB를 넘어서 확장되지 않도록 제한한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

2020.12.19 : supplemental page table 구현(vm\_entry)

2020.12.20 : page fault handling 구현

2020.12.21 : swapping 구현

2020.12.22 : stack growth 구현

* 1. **개발 방법**

1. Page Table & Page Fault Handler

우선 supplementary page table을 구현하기 위해 보다 많은 정보를 담는 page를 구현하기 위한 새로운 자료구조 vm\_entry를 정의한다. 해당 page로 부터 각종 정보를 확인할 수 있으며 이를 정의한 내용을 vm 디렉토리에 page.c / page.h를 작성한다. 또한 frame을 관리하기 위해 관련된 함수를 동일 디렉토리의 frame.c / frame.h에 작성한다. Page Fault Handler를 구현하기 위해 userprog/exception.c의 page\_fault 함수를 수정한다.

2. Disk Swap

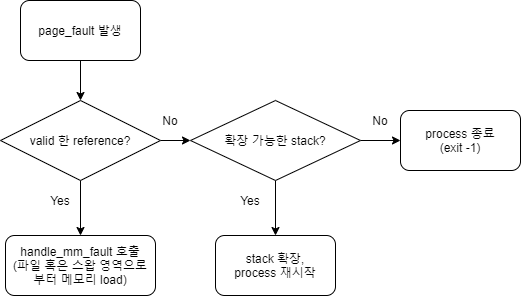
vm 디렉토리에 swapping 관련 기능을 구현한 swap.c / swap.h 파일을 작성한다. 해당 부분에서 swap in / swap out 기능을 구현하는 함수를 구현한다.

3. Stack Growth

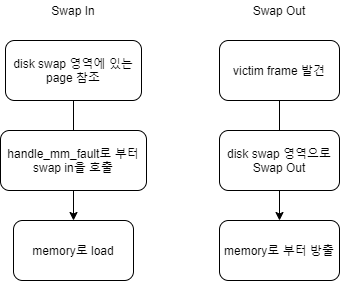
userprog/exception.c의 page\_fault함수가 호출될 때 stack이 확장가능한 상황인지 판별하고 가능하다면 확장하는 함수를 호출하여 stack을 확장한다.

1. **연구 결과**
   1. **Flow Chart**

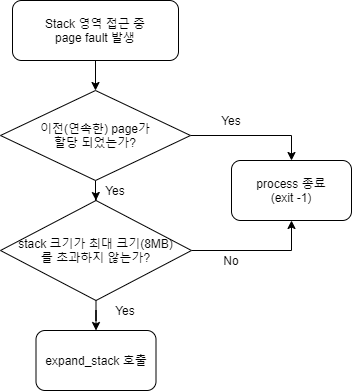
1. Page Table & Page Fault Handler



2. Disk Swap



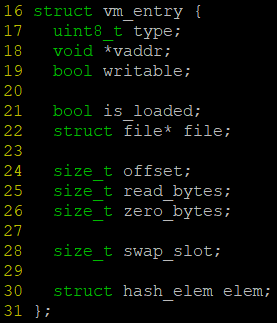
3. Stack Growth



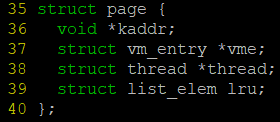
* 1. **제작 내용**

1. Page Table & Page Fault Handler

해당 부분을 구현하기 위해 vm 디렉토리에 page.c/page.h, frame.c/frame.h 파일을 추가하였다. 각각 page, frame을 관리하기 위한 파일이며, 다음과 같이 supplemenatry page를 구현하기 위해 새로운 구조체 vm\_entry와 page 구조체를 정의하였다.



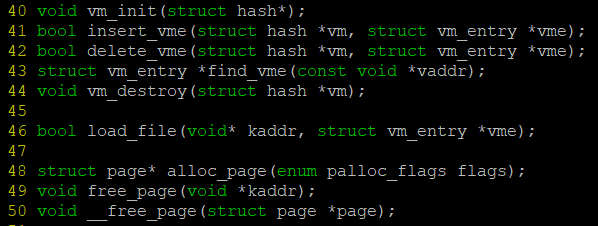
(vm/page.h vm\_entry 구조체)



(vm/page.h page 구조체)

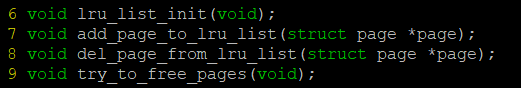
기존의 페이지보다 많은 정보를 포함하기 위해 위와 같은 구조체를 정의하였다. 기본적으로 vm\_entry(virtual memory에서의 page entry)는 hashing 방법을 이용하여 찾으며, 이를 위해 hash\_elem elem 변수를 선언하였다. page 구조체는 vm\_entry를 포함하고 있는 구조체로 이후 LRU 알고리즘을 구현하기 위해 LRU clock list에 삽입하기 위해 정의한 구조체이다.

page.h에 정의되어 있는 함수로는 다음과 같은 것들이 있다.



(vm/page.h 함수)

나머지 함수들은 자료구조를 이용하기 위한 초기화, 해제 등의 기능을 하고, load\_file이라는 함수가 핵심이다. load\_file은 kaddr의 위치에 vme(virtual memory page를 memory에 load하는 함수이다. 이는 process.c의 handle\_mm\_fault함수에서 호출되며, 성공적으로 메모리에 페이지가 load된 경우 true를, 그렇지 않은 경우 false를 return 한다.



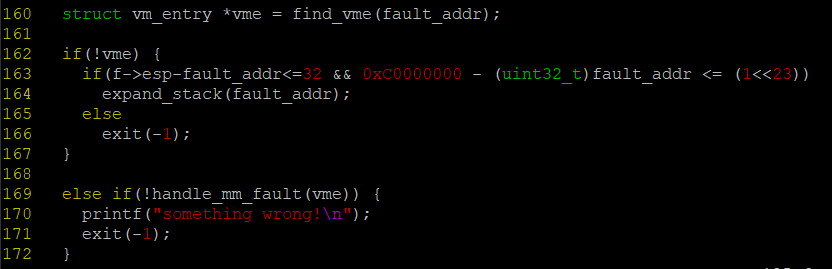
(vm/frame.h 함수)

frame.h 파일에 정의되어 있는 함수는 위와 같다. try\_to\_free\_pages라는 함수가 중요 역할을 하는데 해당 함수는 palloc\_get\_page함수가 호출되어 kernel 혹은 user pool에서 새 frame을 할당받을 때 실패가 일어난 경우 호출되어서 프로세스에 할당되었던 frame을 evict하고 새로운 frame을 할당 받는, 즉 palloc\_get\_page가 실패했을 때 성공하도록 보조해주는 역할을 하는 함수이다. try\_to\_free\_pages의 내부 구조는 다음과 같다.



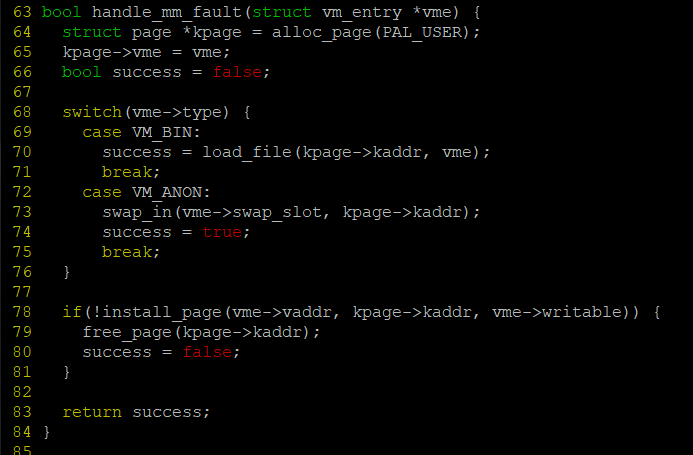
위의 주석된 부분만 보면 된다. 프로젝트에서 second chance clock algorithm을 통해 LRU를 구현하였으므로 62줄에서 만약 해당 page가 access(reference)되었으면 reference bit을 0으로 세팅하고 다음 victim frame을 찾는다. 67줄에서 만약 해당 page가 modified되었으면 evict될 때 consistency를 유지하기 위해서 disk로 swap 해준후, \_\_free\_page 함수 호출을 통해 free frame을 생성해낸다.

page fault를 관리하는 부분은 userprog/exception.c의 page\_fault함수와 userprog/process.c의 handle\_mm\_fault함수가 있는데 우선 page\_fault 함수 중 수정된 부분은 다음과 같다.



(userprog/exception.c page\_fault)

160줄에서 find\_vme함수를 호출하여 fault\_addr에 대응되는 page를 구해낸 후, 169줄의 handle\_mm\_fault함수의 argument로 넘겨준다. handle\_mm\_fault는 다음과 같다.

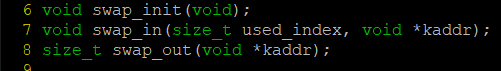


(userprog/process.c handle\_mm\_fault)

vme->type이 VM\_BIN(그냥 바이너리 파일)인 경우, load\_file 함수를 호출하여 파일로부터 메모리에 로드한다. VM\_ANON(swap 영역의 파일)인 경우, swap\_in을 호출하여 swap disk로부터 메모리에 로드한다. 성공여부를 return 값으로 반환한다.

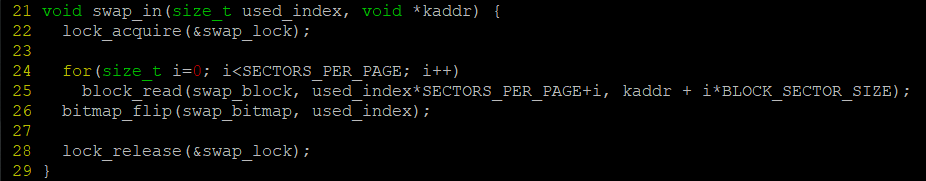
2 Disk Swap

해당 부분을 구현하기 위해 vm 디렉토리에 swap.c/swap.h 파일을 추가하였다. swap 파일에 정의된 함수는 다음과 같다.



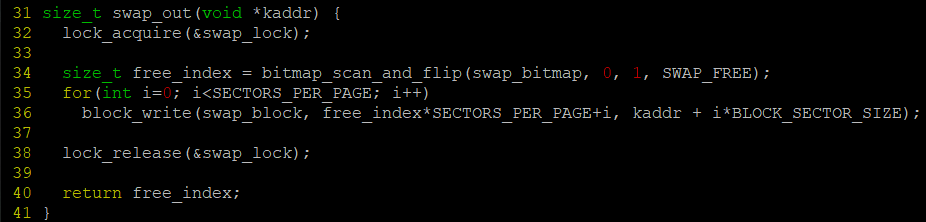
(vm/swap.h)

이 중 swap\_in, swap\_out 함수 구현 부분은 다음과 같다.



(vm/swap.c swap\_in 함수)

하나의 block sector는 512 byte이므로, 8개의 sector가 모여 하나의 page를 이룬다. swapping은 page 단위로 이루어지므로, 반복문으로 block\_read를 호출하여 순차적으로 sector 크기만큼 buffer에 읽어주는 역할을 하였다. 또한 해당 block이 비었음을 알리기 위해 bitmap\_flip을 호출한다.

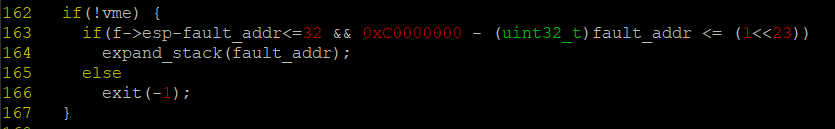


(vm/swap.c swap\_out 함수)

하나의 block sector는 512 byte이므로, 마찬가지로 반복문으로 buffer로부터 swap 영역에 write해준다. 또한 해당 부분을 사용했음을 알리기 위해 block bitmap을 bitmap\_scan\_and\_flip을 호출한다.

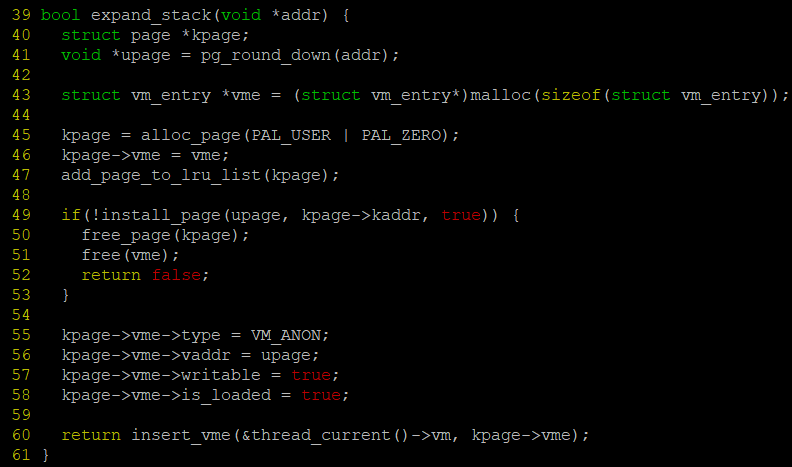
3. Stack Growth

Stack Growth는 exception.c의 page\_fault 함수 내부에서 처리된다.



(userprog/exception.c page\_fault)

163줄에서 스택을 확장할 조건이 되면 expand\_stack함수를 호출하여 스택을 확장한다. expand\_stack은 다음과 같다.



(userprog/process.c)

단순히 위와 같이 새로운 vm\_entry를 생성하여 frame을 할당하고 lru\_list에 추가한 뒤 성공 여부를 반환한다.

* 1. **시험 및 평가 내용**

