**Pintos Project 4: Virtual Memory**

담당 교수 : 김 영 재

이름 / 학번 : 이호성 20171680

개발 기간 :11/13~11/21

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

가상 메모리를 구현하여 page fault handling을 구현한다. 물리메모리와 디스크 사이에 swap을 구현하여 더 효율적인 메모리 운용을 가능케 한다. 또한 user program이 legal한 방법으로 stack영역을 더 요구한다면 page fault를 이용하여 페이지를 추가로 할당한다.

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

현재까지는 page fault 가 발생했다면 부적절한 접근이었을 것이다. 하지만 이번 프로젝트에서는 swap space의 개념이 추가되어 page fault 가 일어났더라도 부적절한 접근이 아닌 물리 메모리에 존재하지 않고 디스크에 저장되어 있는 경우 일수 도 있기 때문에 디스크에 해당 페이지가 있는지 확인하고 있다면 로딩을 한다.

* 1. Disk Swap

process에 할당해 줄 물리 메모리가 부족할 때 디스크로 swap-out 이 일어나야 한다. 이때 swap 할 page의 결정은 pseudo-LRU policy 중 하나인 second chance 알고리즘을 활용하였다.

* 1. Stack Growth

만약 stack access 로 보이는 page fault 가 일어난다면 다른 stack page를 할당해 주어 적절한 stack 사이즈를 유지하도록 한다.

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

현재까지는 swap space의 개념이 없어서 page fault는 사용할 수 없는 영역을 접근할 때만 발생하였다. 하지만 swap space의 개념이 추가되었기 때문에 page fault는 앞선 이유 외에도 swap out 되어 page table에 없는 페이지에 접근을 시도한 경우는 디스크에서 해당 페이지를 찾아내 reload한다.

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

frame list를 관리할 때 pseudo-lru policy 인 second chance 알고리즘을 활용하였다. 이 알고리즘은 frame list를 순회하며 accessed bit 가 1일 경우 기회를 한번 더 주어 accessed bit를 0으로 만든다. accessed bit 가 0일 경우 victim으로 선정하여 evict 한다. 만약 dirty bit 가 1이면 로드 된 후 write 가 이루어진 경우이므로 swap\_out 하여 정보를 업데이트 한다.

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

page fault가 일어난 fault-addr에 대하여 validity를 먼저 확인한다음 유효한 주소인데 해당 주소의 vm\_entry를 찾을 수 없다면 stack의 공간이 부족한 상황이므로 stack을 더 할당 할 수 있는 상황이면 stack공간을 할당 해 준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

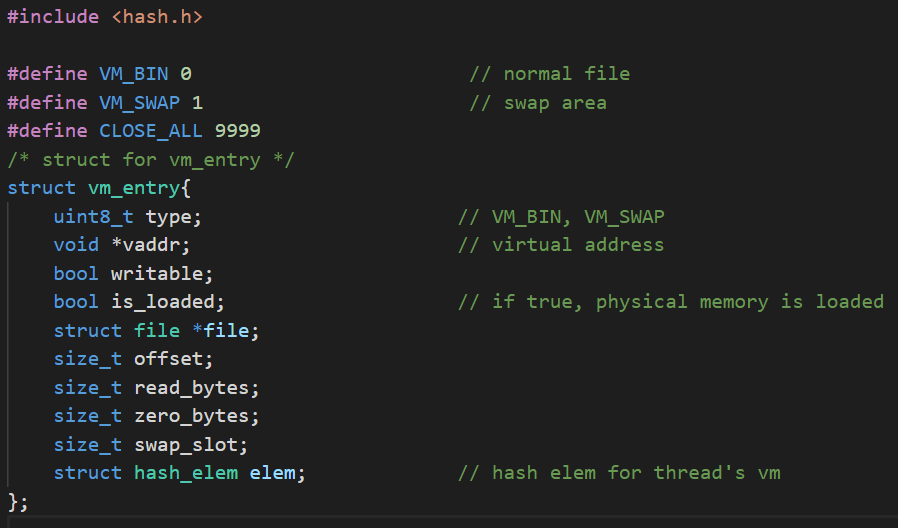
* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

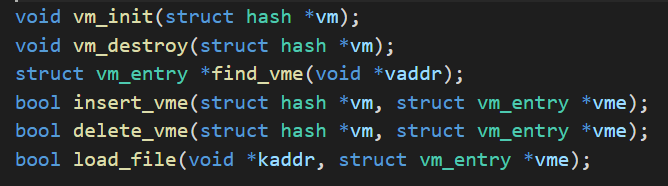
11/15~ 11/17 : Page Table & Page Fault Handler 구현

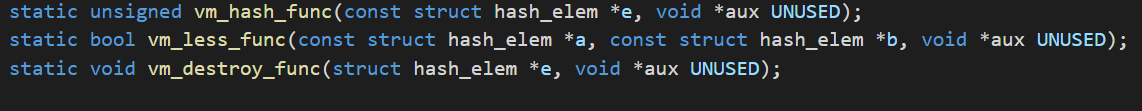
11/18~11/21 : disk swap & stack growth 구현

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  1. Page Table & Page Fault Handler

page.c page.h 를 추가하였다. 그리고 process.c 에 page fault handler 인 handle\_mm\_fault를 작성하여 page fault 가 일어났을 때 무슨 페이지 폴트인지 확인을 하여 적절한 대처를 하는 함수도 작성 하였다.

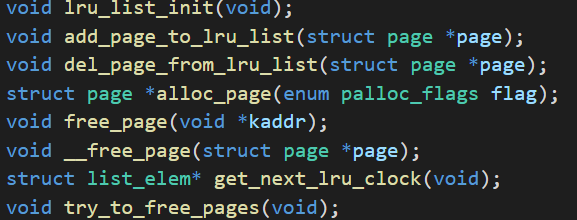




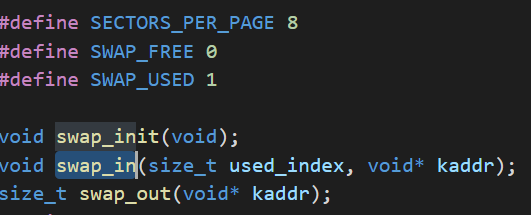


* 1. Disk Swap

swap.c swap.h 파일을 추가하여 swap\_in swap\_out 함수를 구현하였고 frame.c frame.h 파일을 추가하여 pseudo-lru policy 인 second chance algorithm을 이용하여 swap\_out 할 page을 정하였다.



<frame.h>



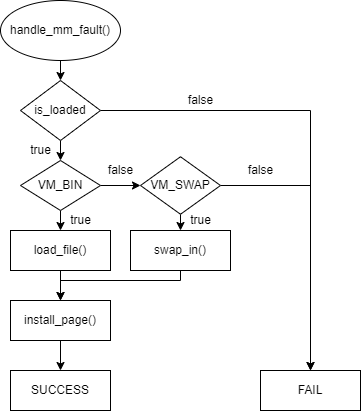
<swap.h>

* 1. Stack Growth

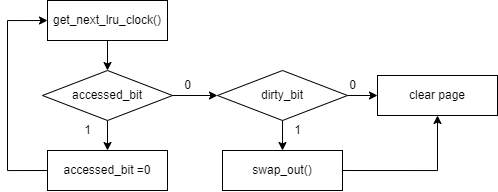
process.c에 expand\_stack(void\* addr)을 추가하여 page\_fault가 발생하였을 때 fault\_addr가 유효한 address인데 vm\_entry를 찾지 못했다면 스택 공간이 부족한 상황이므로 expand\_stack을 해준다.

1. **연구 결과**
   1. **Flow Chart**

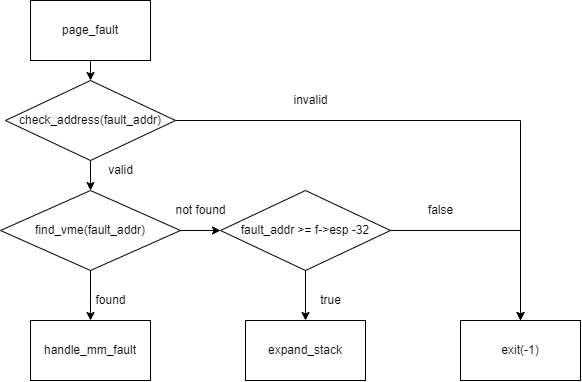
* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성
  1. Page Table & Page Fault Handler



* 1. Disk Swap



* 1. Stack Growth



* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  1. Page Table & Page Fault Handler

우선 vm\_entry를 관리하기 위해 page.h에 vm\_entry 구조체를 추가하였다. 이 vm\_entry 들을 관리하기 위해 hash 자료구조를 활용하야 thread.h에 struct hash vm을 추가하여 스레드가 가진 가상주소공간을 관리하는 해시테이블(supplementary page table)을 만들었다. 이렇게 생성한 vm을 프로세스 생성시 start\_process에서 해시 테이블을 초기화 하였으며 프로세스 실행 중에는 페이지 폴트가 발생 시, vm\_entry을 해시 테이블에서 탐색하였으며 프로세스 종료 시 해시테이블의 버킷 리스트와 vm\_entry들을 제거하였다. 그 다음 가상 주소공간 초기화를 하였는데 load\_segment()에서 가상 메모리 관련 자료구조를 초기화하는 기능을 추가하였다. 이곳에서 프로세스 가상주소공간에 메모리를 탑재하는 부분을 제거한 다음 vm\_entry 구조체 할당, 초기화, 해시테이블에 삽입을 하였다. setup\_stack 에서도 vm\_entry 생성,초기화, 해시테이블에 삽입으로 변경하였다.

manual을 읽었을 때 page를 load 할 때 demand paging을 구현해야 할 것 같았다. 우선 exception.c의 page\_fault()에서 vm\_entry 검색 후 페이지를 할당하도록 코드를 수정하였다. 이때 page fault handler로 handle\_mm\_fault 함수를 추가하였다. 이 함수는 유효한 페이지 폴트 시 물리페이지를 할당한다. 이는 page.c에 load\_file (void\* kaddr, struct vm\_entry \*vme)함수를 추가하여 이용하였다. 이 함수는 vme의 파일과 오프셋으로 한 페이지를 kaddr로 읽어 들인다. 다 읽어 들이지 못한 부분은 0으로 채운다. 그 다음 물리메모리에 적재가 완료되면 process.c 에 있는 static bool install\_page(void \*upage, void \*kpage,bool writable)를 사용하여 가상주소와 물리주소를 페이지테이블로 매핑한다.

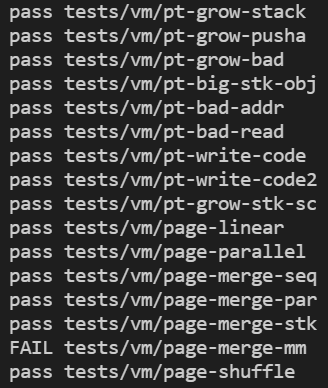
* 1. Disk Swap

우선 page.h에 page 구조체를 만들었다. 이는 물리 페이지 하나를 표현하는 자료구조로이다. 페이지 교체 정책으로는 pseudo-LRU policy 인 second chance algorithm을 사용하였다. LRU list 를 초기화, 삽입, 삭제 하는 함수[void lru\_list\_init (void) / void add\_page\_to\_lru\_list(struct page\* page)/ void del\_page\_from\_lru\_list(struct page\* page)] 그리고 이것을 활용하여 페이지 할당과 해제를 하는 [struct page\* alloc\_page(enum palloc\_flags flags) / void free\_page(void \*kaddr)]를 frame.c에 구현하였다. 그리고는 clock 알고리즘으로 lru list 를 이동하는 [static struct list\_elem\* get\_next\_lru\_clock()]를 구현하였다. swap을 진행하는 함수들을 swap.c에 작성하였다. void swap\_init(size\_t used\_index, void\* kaddr)으로 swap 영역을 초기화 하였으며 [ void swap\_in(size\_t used\_index, void\* kaddr) /void swap\_in(size\_t used\_index, void\* kaddr) ]를 추가하였다. 위 함수들로 실제로 리스트를 순회하며 페이지들을 swapping 하는[void try\_to\_free\_pages()]를 구현하였다. second change algorithm을 이용하여 victim page를 선정하고 방출하여, 여유 메모리 공간을 확보하고 여유 페이지의 커널 가상 주소를 리턴 하도록 구현하였다.

* 1. Stack Growth

현재 스택의 크기를 초과하는 주소에 접근이 발생했을 때, 유효한 스택 접근 인지 판별하여 스택을 확장하도록 구현하였다. page\_fault에 들어온 fault\_addr를check\_address() 함수를 이용하여 유효한 주소인지 확인한다. 유효한 주소라면 해당 주소에 대응하는 vme 가 있는지 확인한다. 대응하는 주소가 있다면 그대로 페이지 폴트 핸들러를 사용하고 그렇지 않다면 현재 스택 포인터로부터 grow limit 이내에 포함되는 접근은 유효한 스택 접근으로 간주하여 스택을 확장한다. 스택의 확장은 process.c 에 [ bool expand\_stack(void\* addr) ]을 추가하였다. 이 함수는 addr 주소를 포함하도록 스택을 확장하는데 alloc\_page()를 통해 vme를 할당하고 초기화한다. 그 다음 install\_page()호출하여 페이지 테이블 설정하고 vme를 해시 테이블에 추가한다.

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



mmap syscall을 구현하지 못해 page-merge-mm은 통과하지 못했다.