**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 이용욱 / 20191626

개발 기간 : 2023.12.01 ~ 2023.12.10

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

이전 프로젝트 1, 2, 3에서 page fault 발생 시 프로그램이 종료된다. 이번 프로젝트 4에서는 page fault 발생 시 처리를 추가한다. 물리 메모리가 가득 찬 경우, swap하도록 구현하고, stack 영역에서 page fault 가 발생하면 stack growth를 구현하여서 더 안정성 있고 적절하게 실행되는 핀토스의 구현을 목표로 한다.

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

기존 page table에 원래의 핀토스에는 없던 기능인 가상 주소를 실제 주소로 변환하는 기능을 추가한다. 이 때, 특정 가상 주소에서 페이지 테이블 내의 physical address를 찾으려 할 때 오류 발생시 page fault가 발생한다. 기존 핀토스는 page fault가 발생하면 프로그램이 exit으로 종료했지만, 적절하게 page fault handler를 구현하여 단순히 shut down시키는 것이 아닌, memory에서의 추가 처리를 통해 프로그램이 제대로 동작하도록 구현한다.

* 1. Disk Swap

swap을 구현한다. process에 할당 해 줄 physical memory가 부족할 때, swap disk에 사용하지 않는 frame을 넣고 new frame을 할당받아 오류가 발생하지 않도록 구현한다. 이 때, 적절한 page replacement algorithm중 하나인 LRU 알고리즘을 사용해서 어떤 frame을 swap disk에 넣을지 선택한다. 그러므로 메모리에 용량이 완전히 차더라도 프로그램을 계속해서 동작시키도록 한다.

* 1. Stack Growth

이전 프로젝트에서는 stack의 크기가 단일 페이지로 4KB로 제한 할당되어 있다. 해당 stack의 크기를 초과하는 영역 접근이 발생하는 경우 추가적인 페이지 할당을 하는 stack growth 동작을 구현한다. 이 때, stack을 확장하는 것에 대한 validity도 검사해야한다. 만약 접근해서느 안되는 세그먼트 영역에 접근을 할 시에는 프로그램을 종료시키는 등의 처리를 구현한다.

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

위에서 설명 했듯이 특정 가상 주소에 접근할 때에, 이에 해당하는 실제 주소가 존재하지 않을 시 page fault가 발생한다. 이를 다시 말하면, 페이지 테이블에서 가상 주소에 해당하는 실제 주소가 틀릴 경우에 page fault가 발생한다는 것이다.

이 때, page fault를 handling한다. 이는 해당하는 virtual address의 내용을 디스크에서 읽어와 memory에 프레임으로 쌓는 과정이다. 이 과정에서 유효성 검사(해당 virtual addr 가 segment의 영역인지 검사)를 해야한다. 만약 handling에 성공했다면, 페이지 테이블 역시 update하고, page table에서 이전에 접근했던 가상 주소에 대응하는 메모리 주소를 mapping시킨다.

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

second chance 알고리즘을 적용한다. 이 알고리즘은 LRU알고리즘과 비슷한 알고리즘이다. 메모리에 로딩된 page의 circular list로 관리한다. 참조 비트는 최초에 1로 설정시킨다. page fault handler에 의해 replace되어야 할 때에, 해당 page의 참조 비트값을 check한다. 그 value가 0이면 replace하고, 1이라면, 해당 참조 비트를 0으로 설정하여 page를 떠난 후 나중에 다시 check하는 방식이다.

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

추가 page가 스택 접근으로 나타나는 경우 페이지를 할당하도록 한다. 따라서 page fault 발생시에 fault\_ addr 가 esp보다 크거나 esp-4 or esp-32일 때, stack 이 확장되어야 한다. esp-4 와 esp-32가 push에 의해 발생한 값인데, 그리고 base\_boundary에서 위치해야 한다. 기본 limit은 8MB이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

23.12.01 ~ 12.03: Page table & Page fault handler

23.12.04 ~ 12.09: Disk swap구현 시도 및 Stack growth 구현

23.12.09 ~ 12.10: 보고서 작성

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

**Page Table & Page Fault Handler**

원래의 핀토스에서 페이지 폴트가 발생하면 userprog/exception.c 중 page\_fault 함수에서 에러 메시지를 출력한 후에 process를 kill하도록 구현되어 있다. 이를 이번 프로젝트에 맞게 수정한다.

일단 src/vm 디렉토리에 page.c, page.h 파일을 만든다. page.h에 페이지 테이블과 이에 대한 엔트리 구조체를 더한다. 이 때, 페이지 테이블은 해시 테이블로 구현한다. 이를 위해 threads/thread.h에서 thread 구조체에 해시 구조체를 더한다. 또한 hash table을 초기화 함수, 엔트리 추가 함수, 엔트리 제거 함수를 추가한다. 각 함수에 대한 설명은 아래에서 하도록 하겠다.

또한 userprog/process.c 에서 load\_segment 함수를 고친다. 기존 물리 페이지 할당 및 매핑 부분을 주석처리한 후, vm\_entry 생성, 해시테이블에 추가하는 함수로 변경한다.

또한 userprog/process.c.에서 setup\_stack함수를 수정한다. 기존 내용에 덧 붙여서 아래에 vm\_entry를 생성, hash table에 추가하는 함수,필드 초기화를 추가한다.

virtual address의 유효성을 검사하는 내용또한 추가 구현한다. virtual address에 해당하는 vm\_entry의 존재 유무를 검사하는 것이다. 이는 syscall.c에서 구현한다. 각 함수에 대한 설명은 아래에서 계속한다.

마지막으로 페이지 폴트 처리 함수를 추가 및 수정한다. 전에 말했듯이, exception.c 중 page\_fault 함수에서 error message 출력 후 프로세스를 kill 하는 것을 주석 처리하고, fault\_addr 유효성 검사를 하고, page fault handler 함수를 call한다. 위 함수는 userprog/process.c에 구현한다. physical memory에 할당한 후에, 실제 디스크의 파일을 physical page에 load하도록 한다.

이렇게 페이지 폴트를 처리하도록 하는 것이 최종 목표인 것이다.

* 1. **Disk Swap**

swap과 관련된 동작들은 pintos/src/vm 디렉토리의 swap.c와 swap.h 파일에 구현해야 한다.

일단, vm/page.h에 user 가 가지는 phys page 자료구조를 추가한다.

그리고 swap.c, swap.h에 LRU list 초기화 함수, LRU list 끝에 user page 삽입 함수, LRU list에서 user page 제거 함수를 추가한다.

또한 userpage의 allocation/deletion에 대한 함수 또한 구현하도록 한다. 위 함수들은 src/vm 폴더의 frame.c/ frame.h에 구현한다.

위에서 구현한 함수 들을 바탕으로 setup\_stack function을 수정한다. 기존의 palloc\_get\_page와 palloc\_free\_page 또한 다른 함수로 수정한다.

위의 구현을 완료한다면 swapping이 정상적으로 작동하나 시간 의 부족으로 완전히 구현하지 못하였다.

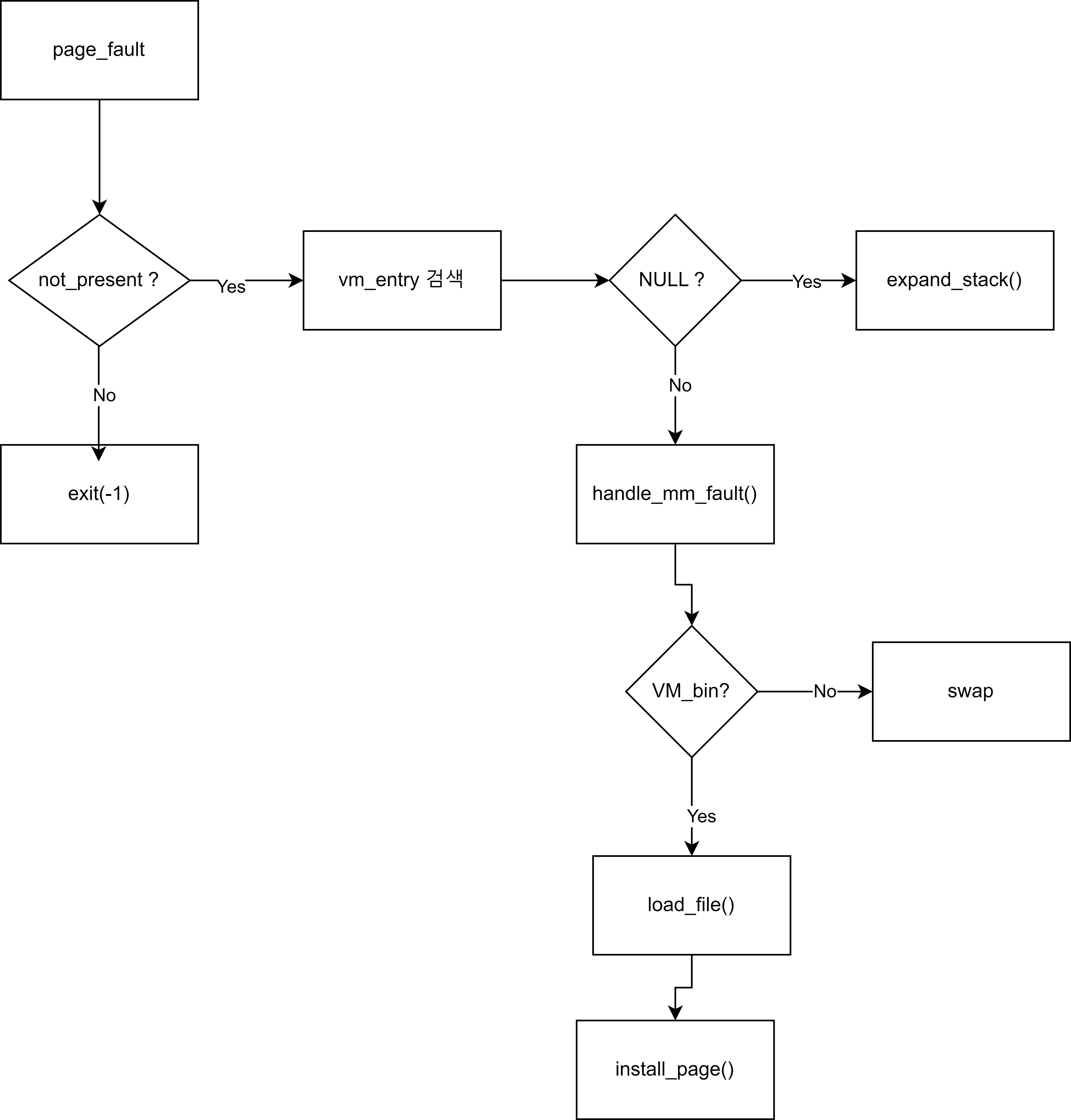
* 1. **Stack Growth**

page.c page.h 에 stack growth 함수를 추가한다. 이 함수는 인자로 받는 addr를 포함하도록 스택을 확장한다. 그 후 alloc\_page함수를 통해 memory 할당을 하고, vm\_entry 할당 과 초기화를 완료한다. 또한 install\_page 함수를 call하여 페이지 테이블을 설정하고, 성공 하면, true boolean을 리턴하도록 한다.

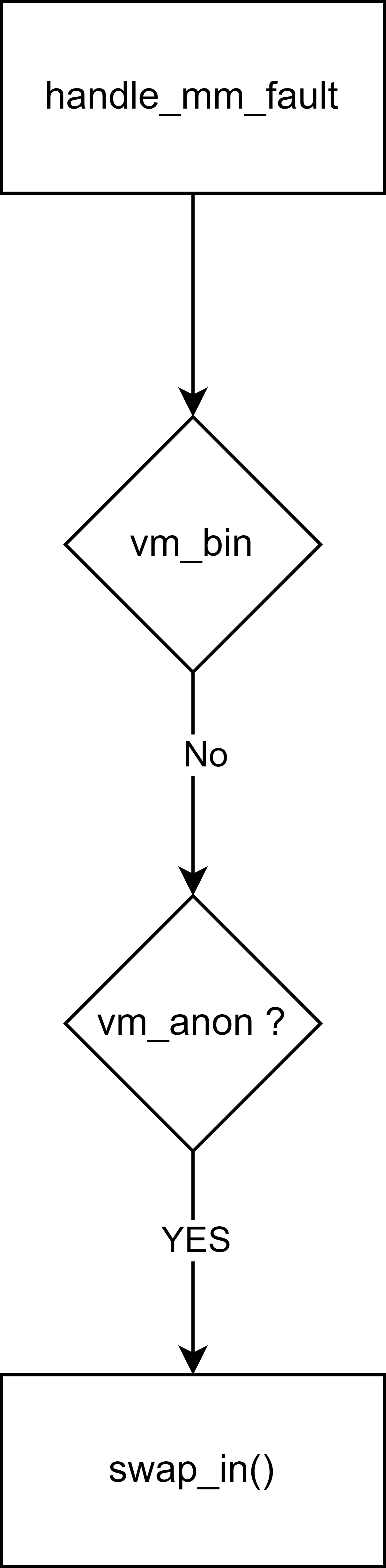
1. **연구 결과**
   1. **Flow Chart**

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

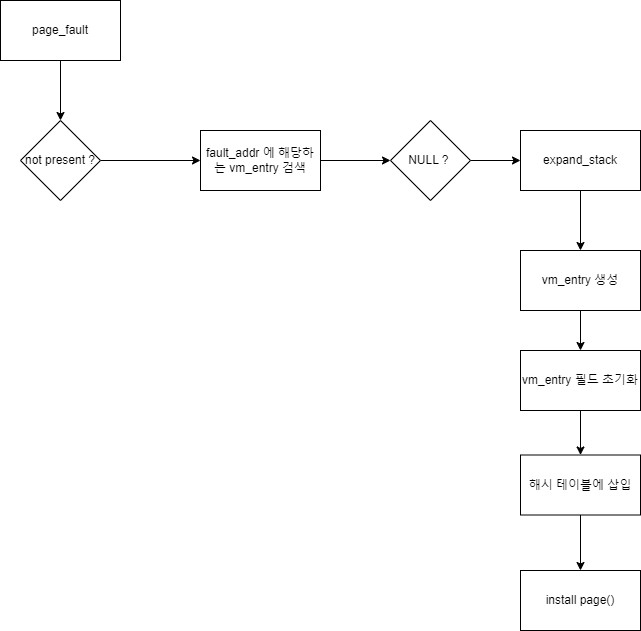
page tagle && page fault handler



Disk Swap



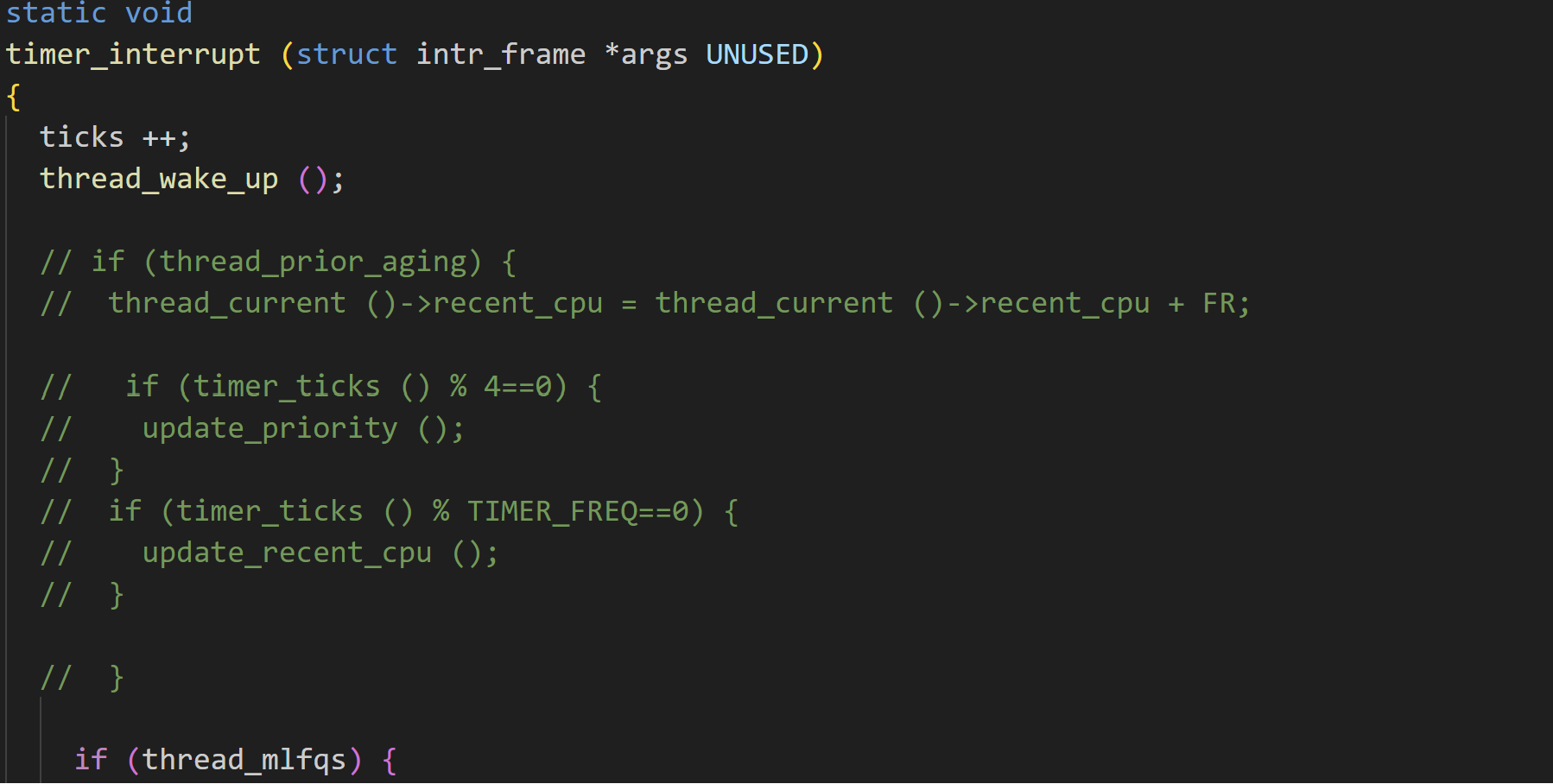
Stack Growth



* 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

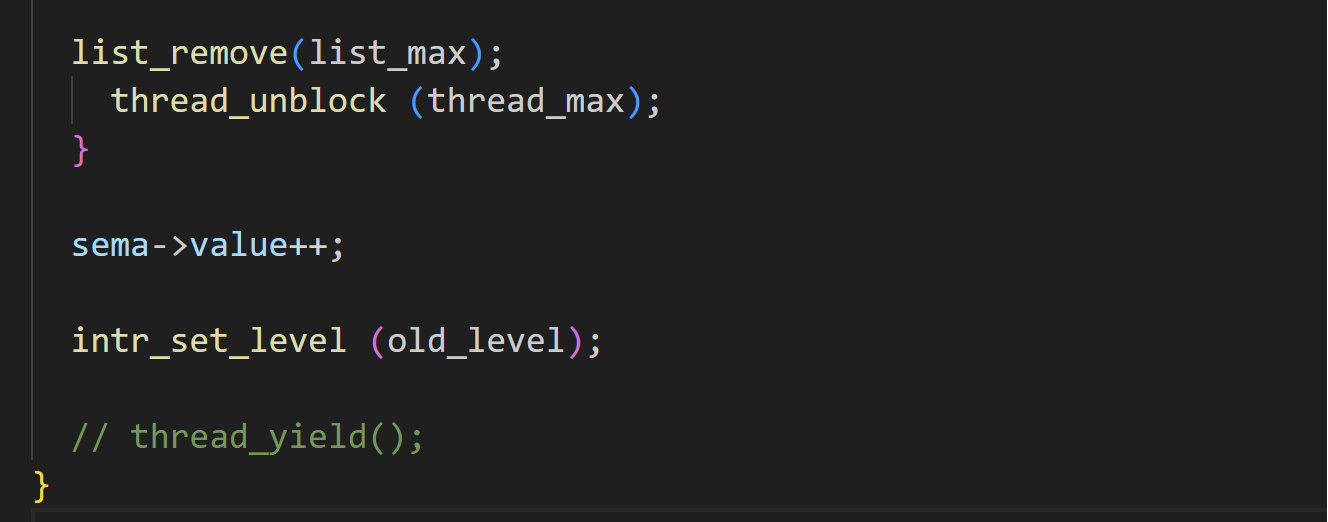
threads/timer.c

timer\_interrupt 함수에서 thread\_prior\_aging변수를 제거한다.



threads/synch.c

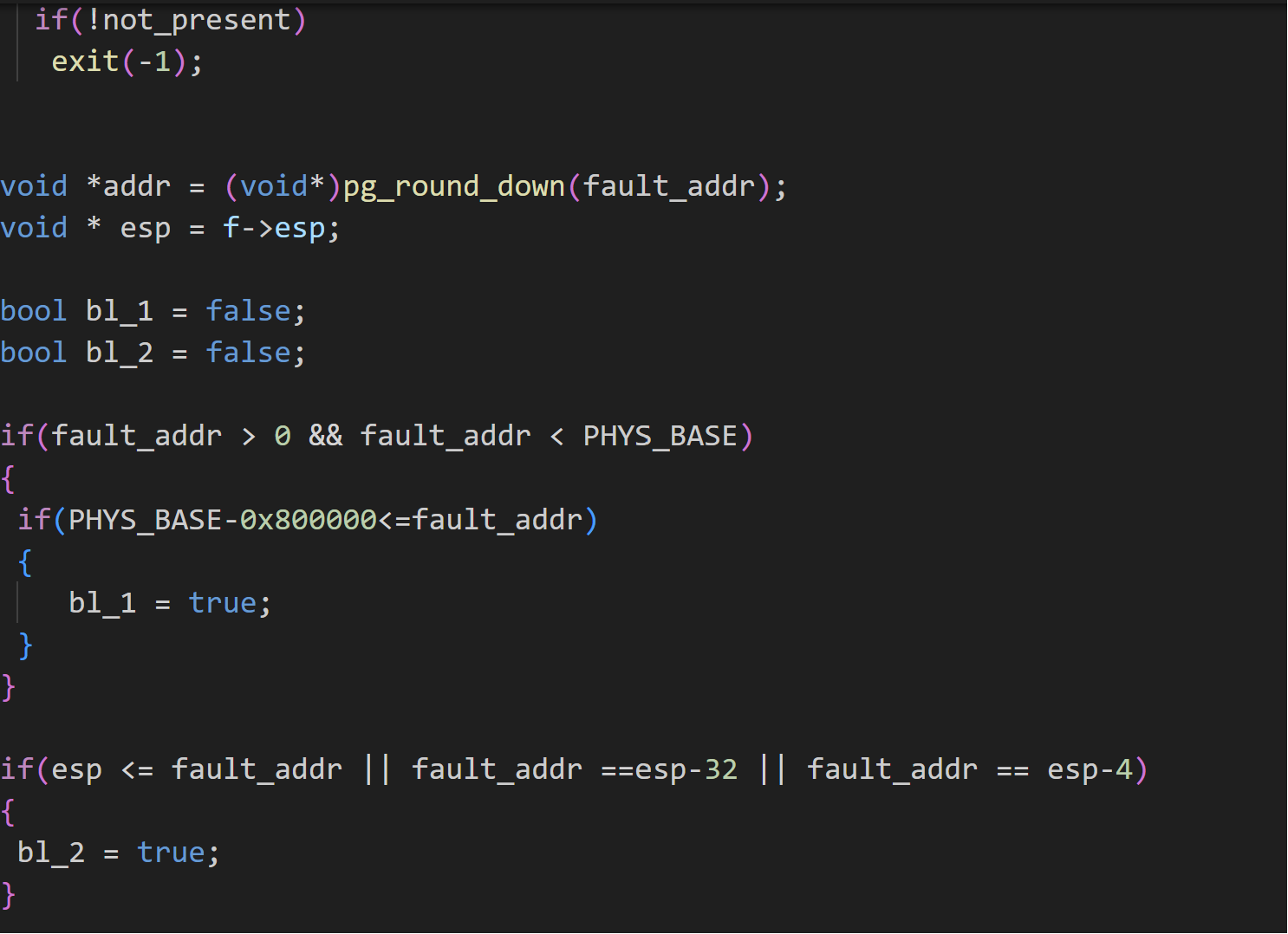
sema\_up에서 thread\_yield의 호출시 에러가 발생하므로 주석처리한다.

****

userprog/exception.c

page\_fault 함수에서 exit(-1)로 프로세스를 종료하는 코드를 주석처리 한 후에,

새로 핸들링 하는 코드를 추가한다.

****

****

우선, not\_present가 false 라면 writing r/o page,

true일시 not\_present\_page라는 뜻이다. 따라서 false => exit(-1)함수로 바로 종료 한다. 그리고, pg\_round\_down함수를 호출하면 fault\_addr 근처에의 page boundary down한것을 addr에 저장하여 현재 frame의 esp는 esp에 저장하도록 한다.

fault\_addr의 연산에 관해서는 (fault\_addr > 0 && fault\_addr < PHYS\_BASE) 그리고 마지막으로 base boundary보다 작은지 확인한다. 그리고 esp가 fault\_addr과 비교하여 작거나 fault\_addr이 esp와 비교해서 32또는 4가 작은지 check 한다. 위의 두 개의 boolean 조건이 모두 true라면 stack\_growth\_func 함수를 call하여 스택을 확장시키고 아닐시에는 exit함수를 call해서 종료시킨다.

vm/page.c 와 page.h

****

현재 thread와 fault\_addr, boolean을 매개변수로 받는다. pg\_round\_down 함수를 이용하여서 fault\_addr nearby page boundary 내림을 us\_page에 저장, palloc\_get\_page 함수를 call하여 새로운 단일 free page를 가져와 kernel virtual address를 ke\_page에 저장한다. 다음 pagedir\_get\_page, pagedir\_set\_page 함수를 차례로 call해서 user 가상 주소에 일치하는 physical address를 찾는다. 마지막으로 page directory에 매핑을 추가해서 stack을 확장한다.

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

