**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용

이름 / 학번 : 최준수 / 20151619

개발 기간 : 2020/12/5 ~2020/12/22

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Project3이 완료된 pintos는 여러 user program을 실행할 수 있으며 동기화(synchronization) 기법이 적용되어 있어 공유된 데이터에 대한 동시 접근도 가능하다. 하지만 현재 pintos는 virtual memory에 관한 기능이 기본적인 것들 밖에 구현되어 있지 않다. 특히 stack의 size가 고정되어 있어 실행 가능한 program 수에 대한 제약이 있고, page fault에 대한 처리가 제대로 이루어져 있지 않다. 따라서 이번 project에서 page fault에 대한 적절한 handler와 stack의 size를 키우는 stack growth 기능을 구현함으로써 이러한 문제를 해결할 수 있다. 또한 physical memory가 부족한 경우에 대해 pseudo-LRU기법을 통한 page replacement algorithm을 사용하여 swap 기능을 구현하는 것도 이번 project의 목표이다.

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

각 프로세스가 요구하는 메모리를 페이지 단위로 할당하고, 이를 관리하기 위한 page table을 생성한다. page table은 thread별로 고유하게 가지고 있으며, hash table로 생성하여 사용한다. page table을 사용하여 physical memory 주소로 변환하여 access하게 된다.

* 1. Disk Swap

메모리가 부족할 경우, 별도로 선언된 swap space로 page를 swap out해두었다가, 요청되면 다시 swap in하여 불러온다.

* 1. Stack Growth

초기에는 stakc의 크기가 4KB로 고정되어있으며 이 범위를 벗어나면 segmentation fault가 일어나게 된다. stack growth를 적용하여 확장가능한 스택의 크기를 최대 8MB가 되도록 수정한다.

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

Virtual memory가 처음에 고안된 이유가 프로그램이 실행함에 있어서 프로그램의 모든 부분이 항상 메모리에 탑재될 필요가 없다는 것이었다. 이를 구현하기 위해 demand paging이라는 기법을 적용하였는데, demand paging은 실제로 요구된 page들만 메모리에 load하는 기법이다. 따라서 처음으로 호출되는 page는 필연적으로 page fault가 발생할 수 밖에 없다. page table에 physical page(frame)이 없을 경우 page fault가 발생하기때문에 새로운 page frame를 할당하고 page table을 갱신한다. 그리고 해당 페이지를 디스크에서 page frame에 탑재하면 된다. 만약 page table의 entry에도 없는 가상주소를 access한다면 이는 segmentation fault가 된다.

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

disk swap발생시 사용한 알고리즘은 clock 알고리즘 혹은 second-chance 알고리즘을 사용하였다. 각 page는 reference bit를 가지고 있으며 처음에 load되거나 reference되면 1로 set된다. process는 victim pointer를 유지하고 있으며, page가 replace되어야할 때, victim pointer가 가리키고 있는 page의 reference bit이 1일경우 0으로 바꾸고 다음 entry로 옮긴다. 만약 새로 옮긴 entry의 reference bit이 0일경우 해당 page를 replace한다. 1일 경우 0으로 바꾸고 옮기는 작업을 반복한다. 만약 모든 page의 reference bit이 1이었다면 모든 page를 0으로 바꾸고 초기의 page로 돌아오기 때문에 초기의 page가 replace되게 된다.

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

판단할 수 있는 방법은 크게 2가지가 있는 것 같다. 첫번째는 naive하게 stack의 최대 확장 가능한 크기를 지정해 놓고, 현재 stack의 크기 + 늘리고자 하는 크기가 최대 확장 크기보다 작다면 확장이 가능하다고 판단하는 방법이다. 두번째 방법은 동적으로 판단하는 것인다. demand paging이 적용된 다음에는 page단위로 메모리가 사용되기 때문에 page table에 올라와있는 entry들 중 stack의 마지막 page 다음의 page를 사용하는 entry가 있는지 확인해보고 없다면 할당하는 방법이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

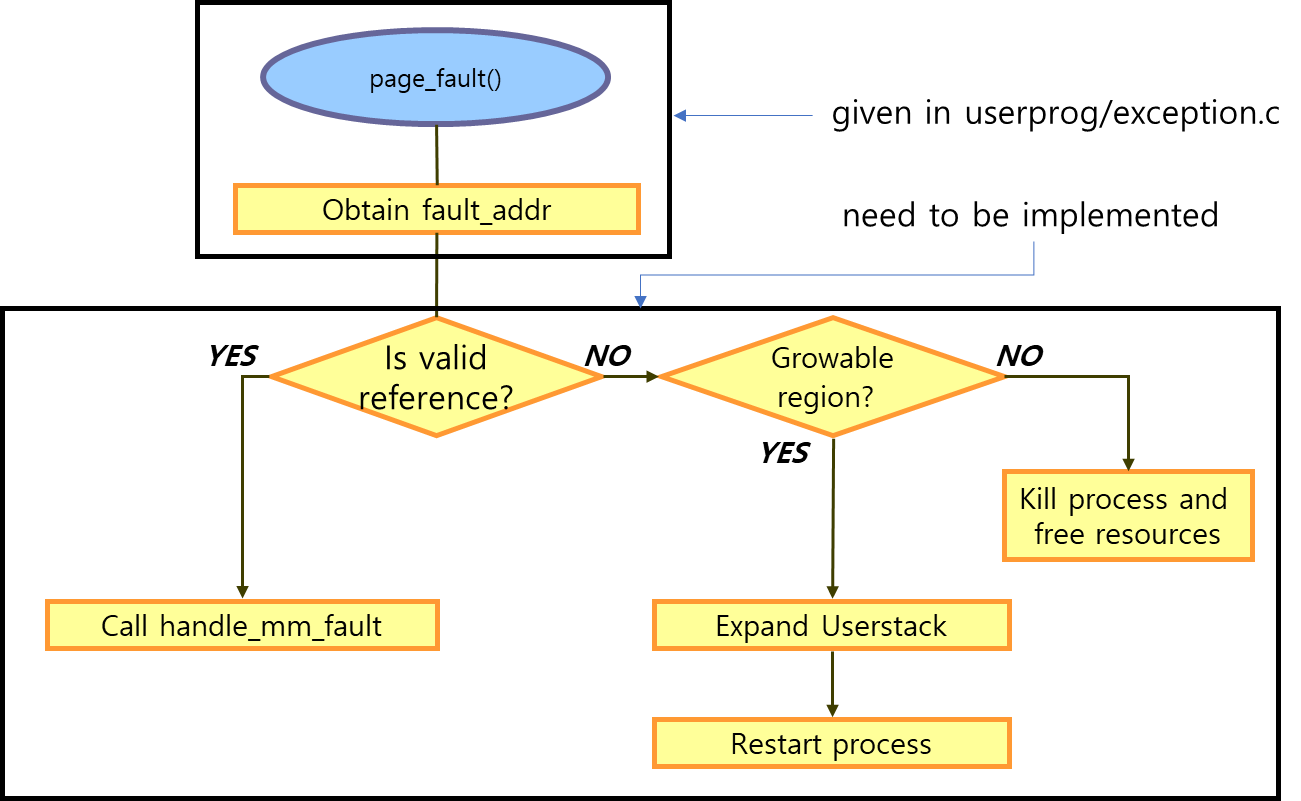
* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2020/12/5 ~ 2020/12/10: 관련 지식 공부

2020/12/11 ~ 2020/12/17: 프로그램 작성

2020/12/18 ~ 2020/12/22: 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 Fault Handler**

****

첫번째로 해당 참조가 유효한지 판단해야한다. 1) Page fault를 발생시킨 주체가 kernel, 2) 현재 메모리에 올라와 있지 않은 page, 3) write 위해 접근하려는 경우, 4) fault address가 NULL인 경우 이 4가지중 하나라도 해당된다면 syscall\_exit(-1)을 통해 해당 process를 종료시켰다

각 thread별로 page table을 구성하기 위해 struct thread에 struct hash vm을 선언하였다.

그리고 struct hash vm에 들어가게될 vm\_entry는 다음과 같이 구성하였다.

struct vm\_entry{

    uint8\_t type; /\* VM\_BIN, VM\_FILE, VM\_ANON의타입\*/

    void\* vaddr; /\* vm\_entry가관리하는가상페이지번호\*/

    bool writable; /\* True일경우해당주소에write 가능

    False일경우해당주소에write 불가능\*/

    bool is\_loaded; /\* 물리메모리의탑재여부를알려주는플래그\*/

    bool pinned;

    struct file\* file;/\* 가상주소와맵핑된파일\*/

    /\* Memory Mapped File 에서다룰예정\*/

    struct list\_elem mmap\_elem; /\* mmap리스트element \*/

    size\_t offset;/\* 읽어야할파일오프셋\*/

    size\_t read\_bytes;/\* 가상페이지에쓰여져있는데이터크기\*/

    size\_t zero\_bytes;/\* 0으로채울남은페이지의바이트\*/

    /\* Swapping 과제에서다룰예정\*/

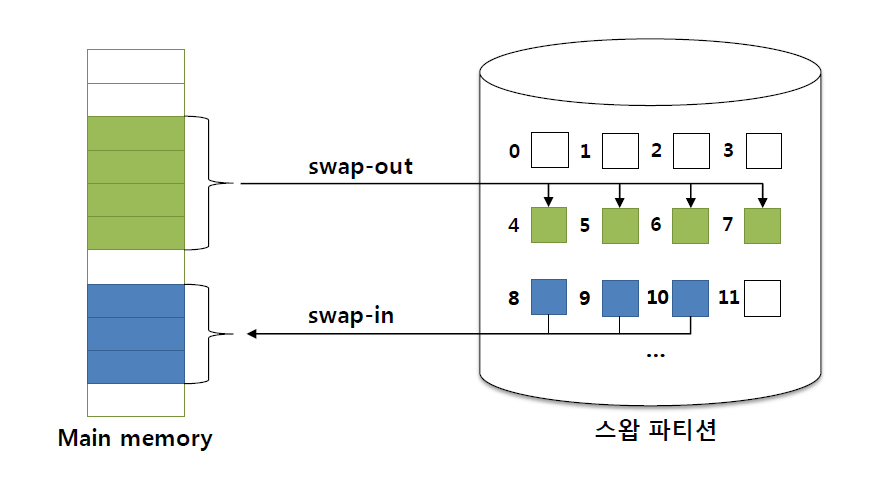
    size\_t swap\_slot; /\* 스왑슬롯\*/

    /\* ‘vm\_entry들을위한자료구조’ 부분에서다룰예정\*/

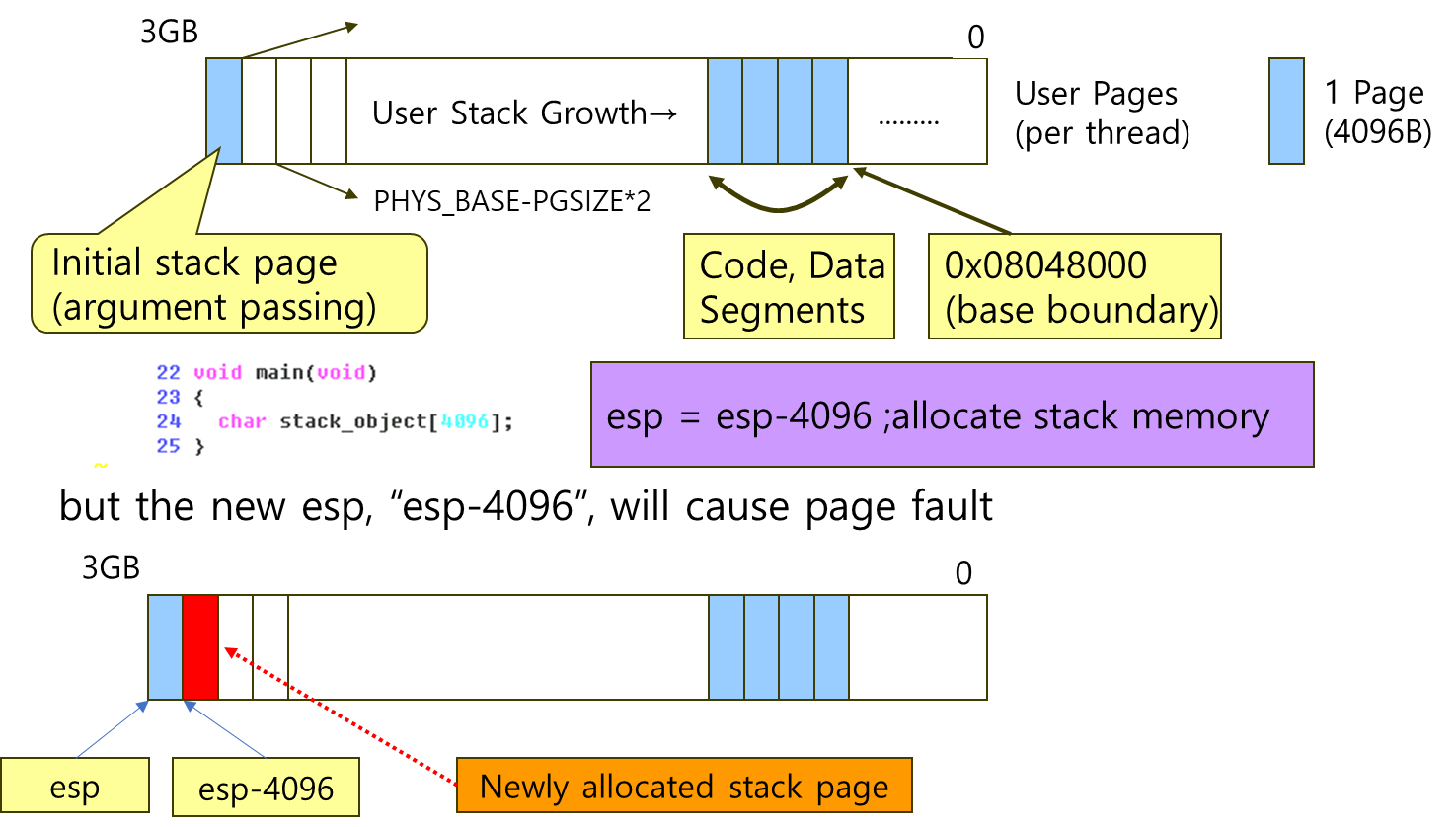
    struct hash\_elem elem; /\*해시테이블Element \*/

};

**Swap**

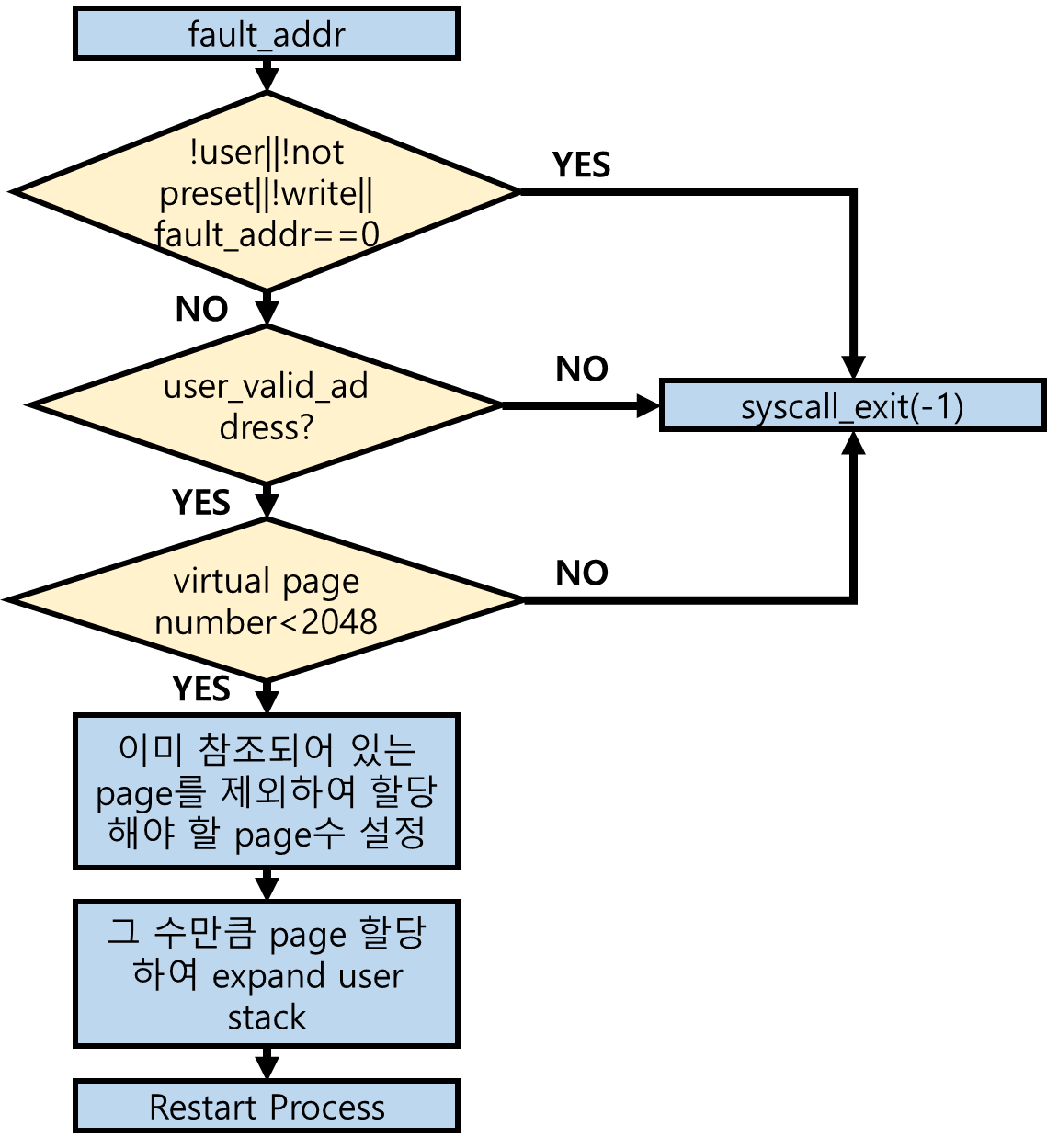


swap은 위와 같은 구조로 이루어진다. second-chance 알고리즘을 구현하기 위해 victim pointer역할을 하게되는 lru\_clock을 선언하여야하고, 실제 메모리에 할당되어있는 메모리를 관리하기 위해 lru\_list라는 list를 하나 선언한다. 만약 page가 swap out 되어야한다면 lru\_list안에서 lru\_clock을 사용하여 replace될 page를 선택하면된다.

* **Stack Growth**  
  참조하려는 virtual address를 통하여 User stack에 총 할당 되어 있어야하는 page 수를 구한다. 이때, 할당되어 있어야 하는 virtual address space는 3GB를 넘지 않도록 해야한다. 이 page수에서 전에 이미 할당되어 있던 page 수를 제외시켜준다면 추가로 늘려야하는 page 수를 구할 수 있다. 따라서 유저 스택을 검사하면서 page 수를 할당하여 user stack을 늘려준다**.  
  **

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

**Page Fault Handler**

Page fault가 발생하면 interrupt frame을 통해 발생 주체가 누구인지, 메모리에 올라와 있는지, read인지 write인지, fault address가 무엇인지 알 수 있다. 이에 대한 정보는 변수 bool user, not\_present, write, void\* fault\_addr에 저장되어있다. not\_present가 flase이거나 fault\_addr을 가상주소로 가지는 page table entry가 없을 경우에는 syscall\_exit(-1)을 호출한다.

각 thread가 page table을 가지고 있으며, load될 때 실제로 page를 할당하지 않고 vm\_entry를 생성하고 insert\_vme()를 통해 해당 thread의 page table에 삽입한다. 그리고 page fault가 발생하면 exceptions.c의 static void page\_fault()에서 find\_vme()를 통해 page fault가 발생한 가상메모리 주소가 page table에 있는 지 확인을 한다. 만약있다면 이때 실제로 disk에서 메모리로 읽어와서 탑재한다. 만약 없다면 segmentation fault이므로 syscall\_exit(-1)을 호출한다.

**Swap**

구현 X

* **Stack Growth**User stack에 총 할당되어 있어야 하는 page 수를 구하기 위하여, 참조하려는 virtual page number에서 PHYS\_BASE에 해당하는 page number을 뺀다. 만약 이 virtual address space가 3GB를 넘는다면 syscall\_exit(-1)을 호출한다. 추가로 할당할 page 수를 구하기 위해 이미 할당되어 있는 page수를 빼야 한다. 그러므로 PHYS\_BASE부터 PGSIZE씩 내려오면서 pagedir\_get\_page를 호출하여 반환 값이 NULL이 아닐 때, 이미 할당되어 있다고 판단할 수 있다. 이를 통하여 추가로 할당해야하는 page수(page\_num)을 구하고 fault address가 속한 page부터 올라가면서 pagedir\_set\_page()를 통해 할당한다.
  1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부
* 