**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 조린 / 20171692

개발 기간 : 2주

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Supplemental page table을 통해 page단위로 메모리에 프로그램을 올릴 수 있도록 한다.

사용할 수 있는 메모리가 없을 때 second chance lru를 통해 페이지를 교체한다.

기본적으로 할당되어있는 스택보다 큰 스택을 사용할 수 있도록 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

메모리에 매핑되어있는 페이지들을 효과적으로 관리 가능하다.

메모리에 매핑되어있지 않은 페이지들을 적절한 알고리즘을 통해 불러 올 수 있게된다.

* 1. Disk Swap

메모리에서 사라지는 페이지를 swap disk에 일시적으로 저장함으로써 변경된 내용을 반영할 수 있도록 한다.

* 1. Stack Growth

pintos상에서 기본 4KB로 제한되어있는 stack segment의 크기를 키울 수 있게되어

User program이 사용할 수 있는 스택의 크기를 유동적으로 할당할 수 있다.

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

program이 메모리상에 존재하지 않는 부분을 접근하려 할 때 발생하게 된다.(범위를 벗어난 접근, 아직 매핑되지 않는 부분으로 접근)

참조하는 virtual address가 범위를 벗어난 경우 종료시키고, 아직 매핑이 되어있지 않다면 page table을 참조해 frame을 메모리에 매핑시킨다.

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

Second chance algorithm을 사용했는데, page table entry의 access bit을 확인해 1로 되어있다면 0으로 바꾸고, 0이라면 해당 page와 매핑되어있는 frame을 disk로 올려보내고 불러올 page를 올려보낸 frame에 매핑한다.

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

os에서 정의한 최대로 사용할 수 있는 stack의 크기내에서 접근을 시도하고 있다면 stack page를 추가로 매핑한다. pintos상에서는 user virtual address의 bottom인 PHYS\_BASE - pg\_round\_down(address) <= STACK\_MAX\_SIZE인 경우에 확장을 가능하게함.

fault가 발생한 주소를 매핑하고 수행중일 때 현재 수행중인 program의 stack pointer가 signal을 push하지 못하는 상황이 발생하면 안된다. pintos상에서 pusha instruction은 32byte이므로 stack을 추가로 할당한 뒤에도 esp가 signal을 push할 수 있도록 esp-32이상의 주소를 사용하는 페이지인지 확인한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11/14 ~ 11/15 - 이번 프로젝트에서 사용할 api를 어떤 상황에서 사용할 수 있는지 분석

11/16 ~ 11/17 - page table의 원소로 사용될 pte, 실제 매핑되는 메모리의 정보를 담은 frame구조체 선언 및 할당 코드 작성

11/18 ~ 11/19 - devices/block을 참고해 swap device를 통한 swap이 가능하도록 하는 함수 작성

11/20 ~ 11/22 - pte와 frame에 접근해 swap에 사용되는 함수들 작성

11/23 ~ 11/24 - page fault handler에 구현해놓은 함수들을 통해 page swap또는 initialize가 가능하도록 구현

11/25 ~ 11/26 - process.c에서 vm에서 구현한 방식대로 메모리에 접근하도록(page table을 사용하도록) 기존의 코드들 수정.

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

Userprog/exception.c의 page\_fault()에 page table을 참조해 page와 frame을 매핑하는 코드 추가.

Page.c와 frame.c에서 전역변수를 통해 evicted frame의 index와 total frame size를 저장하고 이를 통해 다음 evicted frame을 구하는 코드 추가

Device/block의 api를 통해 pte와 swap disk를 통해 내용을 교환하는 코드 추가

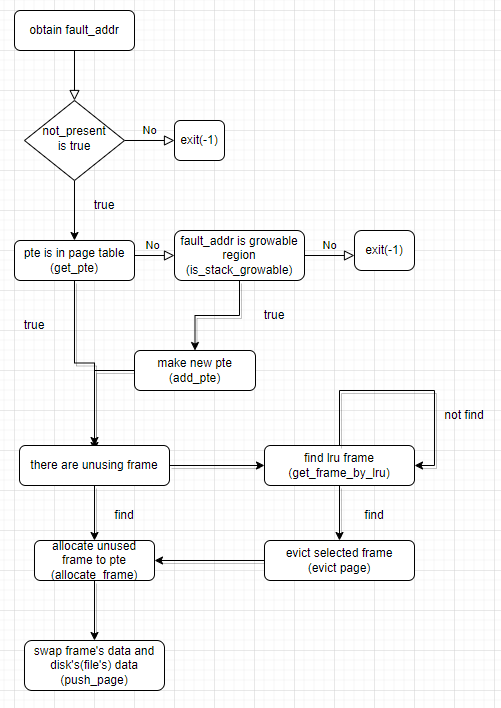
Page.c에 위에서 설명한 조건을 만족하는지를 판단하는 함수 추가.

Thread.c에서 thread에 참조된 pte들을 저장하는 hash 추가.

Userprog/process.c에서 최초의 user stack segment를 할당하는 setup\_stack과 코드부분을 읽어와 data/code segment에 저장하는 load\_segment 함수를 page table에 할당되는 page의 정보를 저장하도록 수정.

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

thread가 자신이 사용하고있는 page를 저장하는 page table인 my\_pages를 hash로 추가한다.

실제 메모리의 물리적인 주소를 page size로 분할해 할당한 frame 구조체에는 physical frame의 address와 객체와 맵핑되어있는 pte를 정보로 가지도록 선언하고, pintos 실행 시 palloc\_get\_page(PAL\_USER)를 통해 user\_pool을 모두 page단위로 할당한다.

swap시 allocate\_frame함수에서 어느 pte로부터도 사용되지 않는 frame이 있는지 확인하고, 그런 frame이 없다면 replace를 성공할 때 까지 수행한다. 결과로 선택된 frame을 반환해 pte와 mapping되도록 한다.

Page table entry의 추가적인 정보를 저장하는 struct pte는 해당 pte의 주소인 user virtual address, disk swap시 저장된 swap disk 의 offset, 맵핑되어있는 frame, data/code segment의 경우 file로부터 data를 읽어오기 때문에 file관련 변수들을 저장하도록 생성한다.

page fault 발생 시 해당 fault\_addr의 pte가 있는지 get\_pte()를 통해 확인하고, error\_code의 not\_present bit을 통해 해당 주소가 메모리 상에 없다는 것을 확인한다.

Not\_present면서 pte도 저장되어있지 않은 경우 새로운 pte를 생성해야 하는데, 주소가 가능한 범위 내인지 확인후에 가능하다면 add\_pte를 통해 주소에 대한 pte를 생성 후 my\_pages에 저장한다. 해당 fault\_addr의 pte를 통해 mapping되어있는 frame을 push\_page를 통해 메모리에 올리게 되는데, 사용되지 않는 frame을 allocate\_frame을 통해 찾아와 정보를 통해 data를 읽어와 load한다.

Process.c의 load\_segment()에서 code/data file의 정보를 pte에 저장한 후, page fault시 load하도록 구현했는데, file이 load되지 않고 해당 page들이 0으로 초기화되는 것을 경험했다. load()의 마지막에 해당 file을 close했기 때문에 load 종료 후에 실행되는 page fault를 통한 file로의 접근이 불가능한 것이 원인이었고, file\_close()부분을 제거하자 문제없이 수행되었다.

Palloc.c에서 bitmap을 사용해 memory pool의 page가 사용 가능한지 확인하는 것을 보고 swap disk도 bitmap을 통해 사용가능한 구간을 확인하도록 했다. page size = 4KB이고, BLOCK\_SECTOR\_SIZE = 512bytes이기 때문에 bitmap의 원소 하나는 8개의 sector를 표현하게 된다. bitmap을 통해 사용가능한 섹터를 찾고, 해당 섹터의 데이터를 읽거나 쓴 뒤, pte의 정보를 수정한다.

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