**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 김준우 / 20211524

개발 기간 : 12/1 ~ 12/10

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**vm 디렉토리 안에 파일을 생성해 구현한다. page table을 만들어 page fault가 발생하는 상황에 프로세스를 관리할 수 있도록 만든다. 그리고 필요한 부분을 physical memory에 롤린다. 이 메모리가 가득찬 경우에는 swap disk라는 임시 저장 공간에 swap이 가능하도록 구현하고, page fault가 stack에서 발생하면 stack growth를 통해 프로그램이 적절하게 실행되게 구현한다.**

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 핀토스에 있는 page table에 virtual memory에서 physical memory로 옮길 수 있는 기능을 구현해준다. 이때 오류가 발생하면 page fault가 일어나는데, page fault handler를 구현해 프로그램을 종료하는 방식이 아니라 프로그램이 계속 동작할 수 있도록 구현한다.

* 1. Disk Swap

앞에서 virtual memory에서 physical memory로 옯겨주는 작업을 추가해줬는데 physical memory는 한정적이기 때문에 부족한 경우가 말생한다. 이 때 swap disk에 사용하지 않는 frame을 넣어 새로운 frame을 할당 받아 오류를 방지하도록 구현한다. swap disk에는 Second chance 알고리즘을 이용해서 어떤 frame을 swap disk에 넣어 줄 지 골라준다.

* 1. Stack Growth

이름대로 stack의 크기를 늘려주는 작업이다. 기존의 핀토스에는 stack의 크기가 4KB로 고정되어 있는데 이 stack의 크기를 초과하는 영역에 대한 접근이 발생하게 되면 stack을 growth하는 동작을 구현한다. 이 때 segment 영역에 접근을 하는 경우에는 프로그램을 종료 시켜줘야 한다.

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

virtual address가 physical memory에 존재하지 않으면 page fault가 발생한다. 앞서 언급했듯이 기존의 핀토스는 page fault가 발생하면 프로그램을 종료하게 한다. 이번 프로젝트에서 userprog/exception.c의 page\_fault 함수를 수정해서 접근하려는 주소가 page table에 존재하는지 확인해주고, handle\_mm\_fault를 process.c 파일에 추가해 physical memory에 load 되도록 구현한다.

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

second chance 알고리즘을 이용해 swap을 구현해주었다. swap이 발생하면 할당된 page들의 리스트를 돌면서 이미 access된 page는 swap\_out해준다., access되지 않은 page는 마크해두고 다음 page로 넘어간다.

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

page fault를 발생시킨 주소가 stack 영역이어야 stack을 확장해줄 수 있다. 이를 위해 주소를 확인해줘야 하는데 verify\_stack 함수를 구현해 이를 확인 할 수 있도록 한다. 스택의 최대 크기는 8MB이므로 이를 초과하는지 확인하고 이 주소가 user 영역인지도 확인을 해주고 처리한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

23.12.1~22.12.5 : 페이지 테이블 구현 및 기존 페이지 할당 방식 변경

23.12.6~23.12.8 : Stack growth 구현

23.12.9~23.12.10 : Disk swap 구현

* 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

vm 디렉토리에 메뉴얼대로 page.c와 page.h, swap.c와 swap.h, frame.c와 frame.h를 추가한다. virtual page table을 관리하기 위해 hash table을 이용해 구현한다. hash.h에 존재하는 함수들을 이용해 init, 탐색, insert, delete 함수등을 구현해준다. virtual page table에 추가할 entry는 구조체를 이용해 구현하고 physical page도 구조체를 이용해 구현해준다.

swap disk를 위한 page들을 구현해야 한다. 이 때 bitamp을 이용해주고, second chance 알고리즘을 이용해 swap할때 페이지를 고르는 기능을 구현한다.

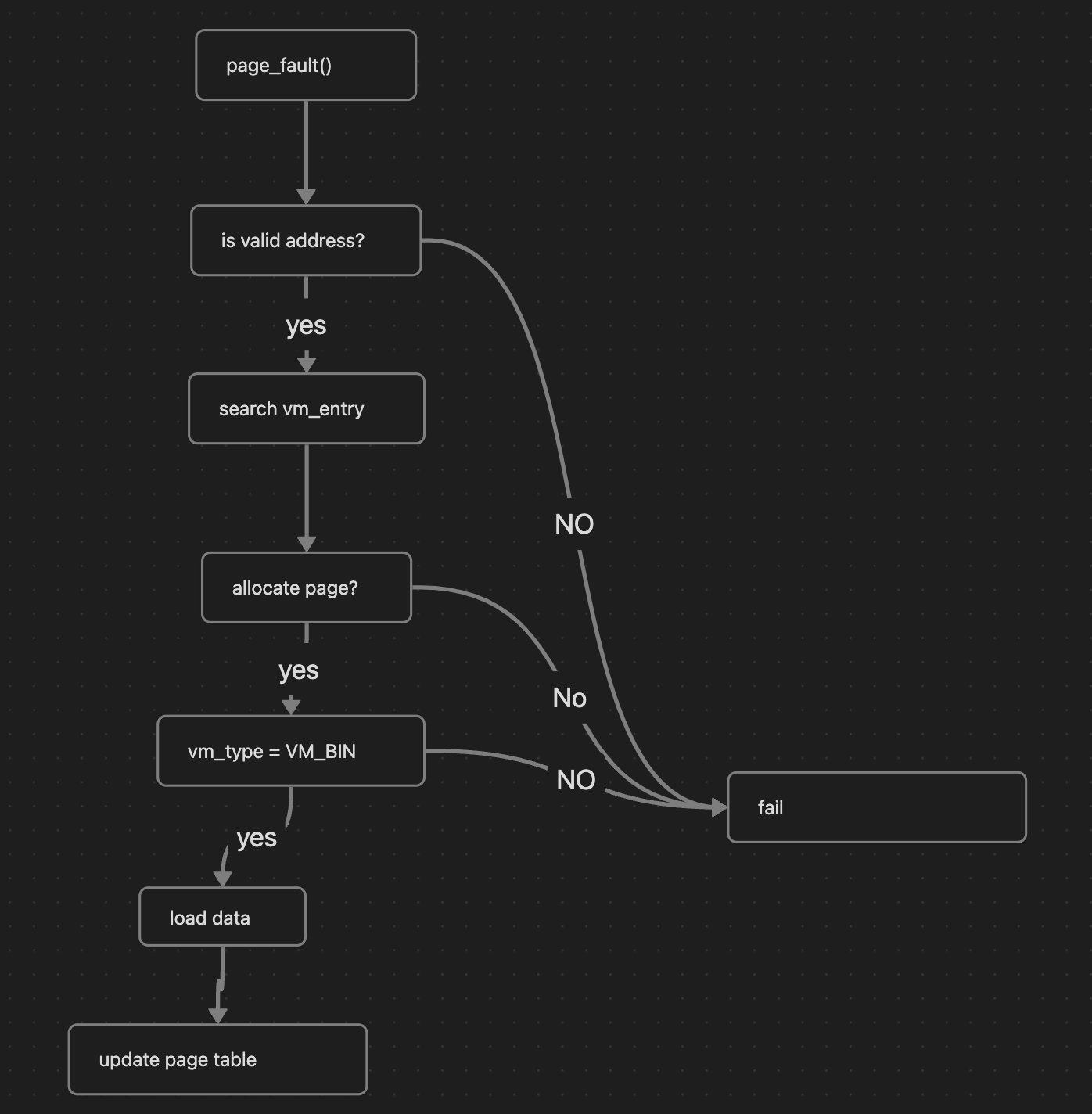
exception,c의 page fault handler 부분을 수정해서 우리가 원하는 대로 종료되지 않고 memory가 load 되도록 구현하고 스택 확장도 구현해준다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성
* 도표, 텍스트, 스크린샷, 평면도이(가) 표시된 사진

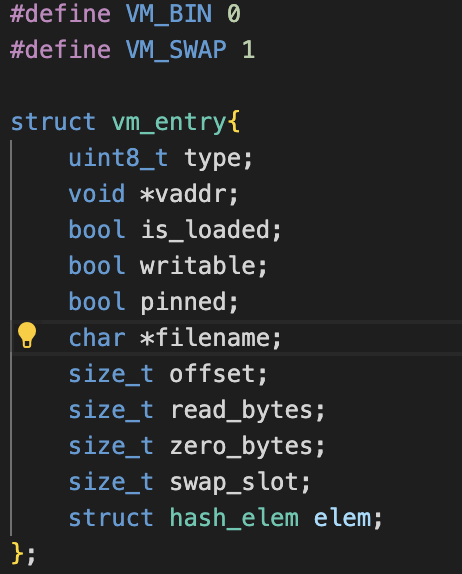
  자동 생성된 설명

도표, 텍스트, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

* 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

vm/fram.c, vm/fram.h, vm/page.c, vm/page.h, vm/swap.c, vm/swap.h 를 생성 후 구현해준다. thread 구조체에 hash table을 위해 struct hash vm을 추가해줬다. page.h에 vm\_entry 구조체를 선언해준다.



type의 종류로 VM\_BIN과 VM\_SWAP이 있고 VM\_BIN 일반적인 바이너리 파일을 로드 한 페이지를 위미하고 VM\_SWAP은 swap 혹은 stack 확장을 나타낸다. load되었는지 여부, write가 가능한지 여부를 나타내는 변수를 그리고 현재 페이지의 offset, read\_bytes, zero\_bytes를 추가하고 swap을 위해 swap\_slot 그리고 hash table을 이용해 관리하기 위해 hash elem인 elem을 추가해줬다. 그리고 page.h에 page 구조체도 추가했다. page 구조체는 physical page를 나타낸다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

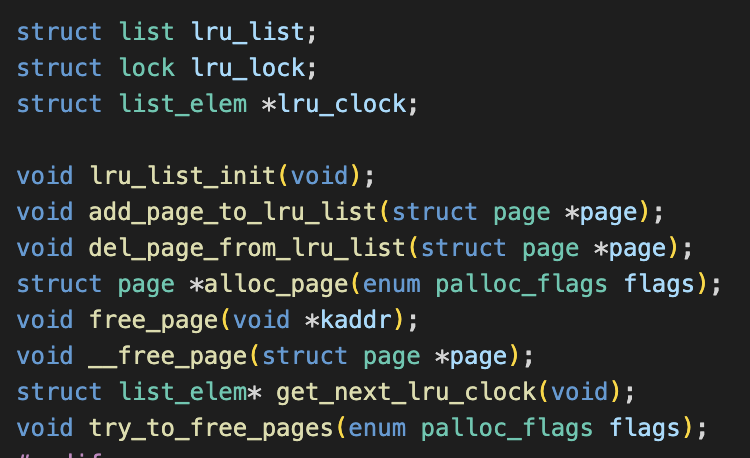
page.c에 구현할 함수들이다. 순서대로 초기화 해주는 함수, hash table의 vm\_entry와 버킷 리스트를 제거하는 함수, vaddr에 알맞은 vm\_entry를 찾아주는 함수, vme를 insert, delete 하는 함수, file\_read함수를 이용해 read\_bytes 만큼 데이터를 physical page에 쓰고 zero\_bytes만큼 남는 부분을 채워주는 함수이다.

여기서 init함수는 thread가 만들어질때 init 되도록 threads/thread.c에 추가해주었고 process.c에서 process가 종료되는 process\_exit에서 vm\_destroy를 추가해서 프로세스 종료시 vm\_entry를 제거하도록 수정했다. load\_segment에서 기존의 physical page를 할당 후 매핑 해주는 부분을 제거하고 page.h에 있는 함수들을 이용해 vm\_entry를 생섷해 필드를 초기화 해서 hash\_table에 insert 해주도록 수정한다. setup\_stack 함수에서는 기존의 페이지 할당, 페이지 테이블 설정, 스택 포인터 설정을 해준 이후에 스택의 vm\_entry를 생성하고 초기화하고 hash\_table에 insert 해주는 과정을 마찬가지로 추가해준다. stack growth를 위해 앞서 말했듯 유저 영역인지 그리고 스택의 최대크기를 벗어나지는 않는지 왁인해줘야 한다. 이 작업은 userprog/syscall.c 에 check\_addr함수를 구현하고 syscall\_handler가 호출된 직후 확인해주도록 함수를 수정해줬다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

vm/swap.h 파일이다. swap slot을 관리하기 위해 전역변수 bitmap을 선언해주고 아래의 함수들을 통해 초기화, 기록, 데이터 가져오기를 구현해줬다.



lru\_list, lru\_clock, lru\_lock을 이용해 페이지들을 관리, 현재 어떤 페이지를 가리키는지, lock으로 동시성 문제 방지를 구현해줬다. init, add\_page, del\_page 함수들은 list.h 함수를 이용해 구현해주었다. alloc\_page는 palloc\_get\_page() 함수에서 page구조체를 이용해 lru\_list에 새로운 page를 할당해 추가하는 함수다. free\_page와 \_\_free\_page는 각각 페이지를 찾아 free하고 lru\_list에서도 삭제해주는 함수다. try\_to\_free\_pages 함수는 second chance 알고리즘을 이용해 이미 access된 page를 찾아서 swap out하도록 만드는 함수다.

폰트, 텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명threads/init.c에 init함수를 호출해서 초기화

userprog/process.c에서 새롭게 handle\_mm\_fault 함수, expand\_stack함수 verify\_stack 함수를 구현해 page fault 핸들링과 stack growth를 구현해줬다. handle\_mm\_fault에서는 vm\_entry를 인자로 받아서 해당 entry의 타입에 따라 VM\_BIN이면 load\_file을 호출하고, 아니라면 swap\_in을 호출하고 install\_page 함수를 통해 user addr와 kernel addr을 매핑하고 엔트리의 is\_loaded를 true로 업데이트 해준다. verify\_stack은 접근하고자 하는 주소가 유효한 주소인지 확인하고 expand\_stack은 주소를 vm\_entry로 만들어서 페이지 테이블에 추가해주는 함수다. 그래서 이를 이용해 stack growth를 만들어 주었다.

make check을 통해 확인해본 결과 page-linear는 write와 read를 못하는 문제가 발생했는데 어떤 방식을 이용해도 해결하지 못해서 결국 pass하지 못했다..

* 1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부
* 텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

  자동 생성된 설명