**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용 교수님

이름 / 학번 : 이건영 / 20181662

개발 기간 : 2022.11.15~2022.12.04

1. **개발 목표**

Virtual memory entry를 관리하는 table을 구현하고, page fault 발생 시 page를 할당하고 stack의 확장될 수 있도록 수정한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. Page Table & Page Fault Handler

기존의 페이지 할당 방식은 프로세스 실행 시 물리 메모리에 Stack, Data, Code 부분을 한번에 load후 사용하는 방식이었다. 따라서 page fault가 발생 할 경우 프로세스를 종료하였다. 본 프로젝트에서는 page를 관리하는 page table을 만들고 page에 대한 요구가 발생할 때마다 물리 메모리에 load하는 방식으로 수정한다. 이를 통해 page fault가 발생하여도 추가적으로 page를 할당하는 요구 페이징을 구현할 수 있다.

2. Disk Swap

물리 페이지가 부족할 경우, 수업 시간에 다뤘던 Second chance(clock) 알고리즘을 사용하여 Victim page를 지정하고 이를 swap out하여 새로운 페이지가 물리 메모리에 load될 수 있도록 구현한다.

3. Stack Growth

기존의 4KB로 고정된 크기의 스택 영역에서 벗어나는 주소에 대해 접근이 발생한 경우, 유효성을 검사하여 스택을 확장하는 기능을 구현한다.

* 1. **개발 내용**

1. Page fault가 발생하는 이유와 이를 handling하는 과정

Page fault는 접근하고자 하는 가상 주소가 물리 메모리에 존재하지 않을 경우 발생한다. 이전 프로젝트까지의 pintos에서는 페이지 폴트가 발생한 경우 바로 프로세스가 종료되었으나, 이번 프로젝트에서는 page fault handler인 exception.c/page\_fault() 함수를 수정하여 접근하려는 주소가 페이지 테이블에 존재하는지 확인하고, handle\_mm\_fault() 함수를 추가하여 올바른 행동을 통해 물리 메모리에 로드되도록 구현하였다.

2. Disk Swap 발생 시 사용한 page replacement algorithm

수업 시간에 다루었던 Second chance(Clock) 알고리즘을 사용하였다. 만약 swap이 일어나야 할 상황이 된다면 할당된 페이지들을 리스트로 만든 lru\_list를 순회한다. 만약 이미 access 된 페이지라면, victim으로 선정한다. 만약 아직 access 되지 않았다면, mark해둔 후 다음 페이지를 탐색한다. victim이 된 페이지는 swap out 된다.

3. Stack growth 구현 시 stack 확장 여부를 판단하는 방법

stack 확장이 가능한 지 판단하려면 페이지 폴트를 발생시킨 주소가 stack 영역인지 확인해야 한다. 이를 위해 verify\_stack() 함수를 작성하여 이를 확인하였다. 명세서에서 스택 확장의 최대 크기는 8MB라고 하였으므로 이를 초과하는 지 확인하고, 폴트를 유발한 주소가 user 영역인지 확인하여 처리하였다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

22.11.15~22.11.25 : 페이지 테이블 구현 및 기존 페이지 할당 방식 변경

22.11.26~22.11.28 : Stack growth 구현

22.11.29~22.12.04 : Disk swap 구현

* 1. **개발 방법**

우선 vm 디렉토리에 파일들을 추가하여야 한다. 명세서에 나와있는 대로 page.c와 page.h, swap.c와 swap.h, frame.c와 frame.h를 추가한다.

다음으로는 가상 페이지 테이블을 관리하기 위한 자료구조가 필요하다. 이는 명세서의 추천에 따라 hash table로 구현한다. 또 hash.h의 함수들을 참고하여 초기화, 탐색, 삽입과 삭제 등의 함수를 구현한다.

가상 페이지 테이블에 추가될 entry를 vm\_entry라는 구조체로 구현한다. 또 물리 페이지 또한 page라는 구조체로 구현한다.

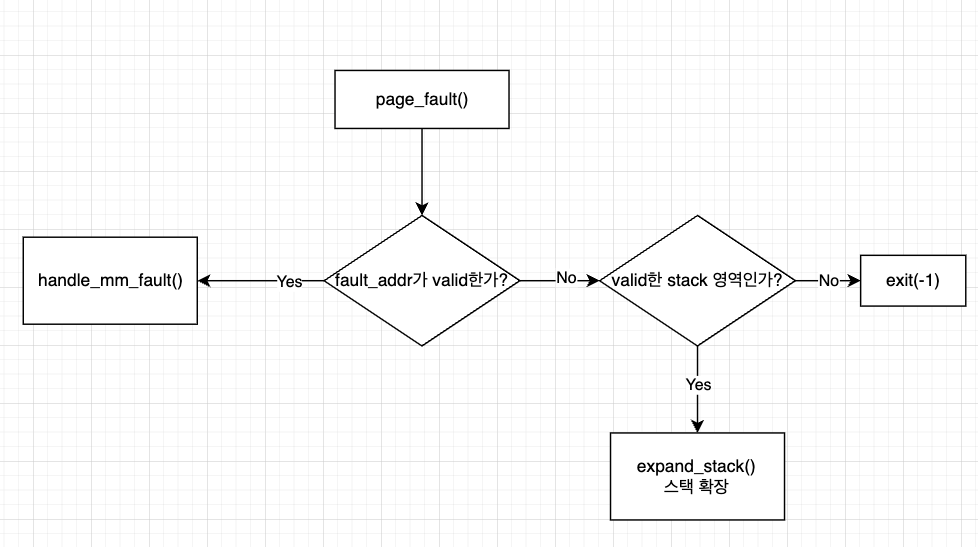
다음으로는 swap disk를 위한 페이지들을 관리하여야 한다. 이는 bitmap을 이용하여 구현한다. 또 clock 알고리즘으로 swap 시 evict 될 페이지를 선정하는 기능을 구현한다.

그리고 기존의 page fault handling 부분을 수정한다. 폴트가 발생할 시 메모리가 로드될 수 있도록 구현하고 스택 또한 확장될 수 있도록 구현한다.

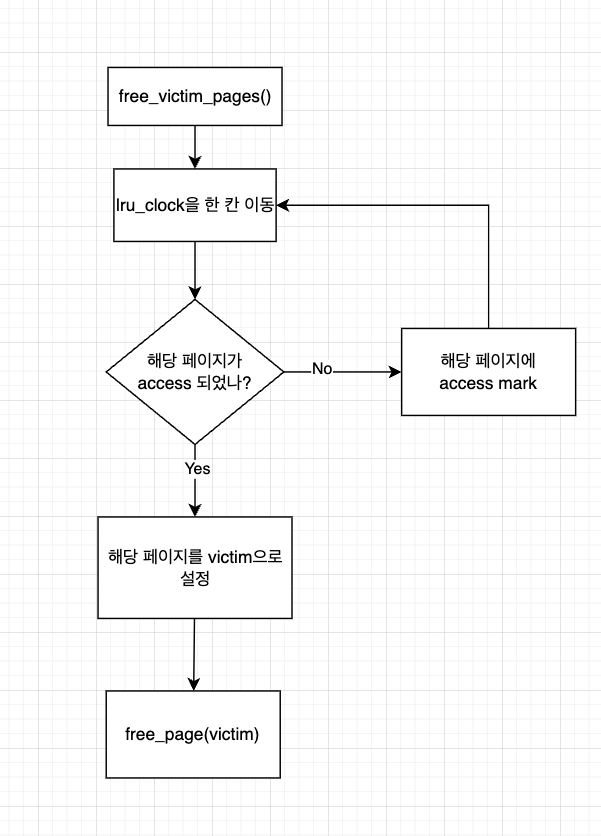
1. **연구 결과**
   1. **Flow Chart**

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

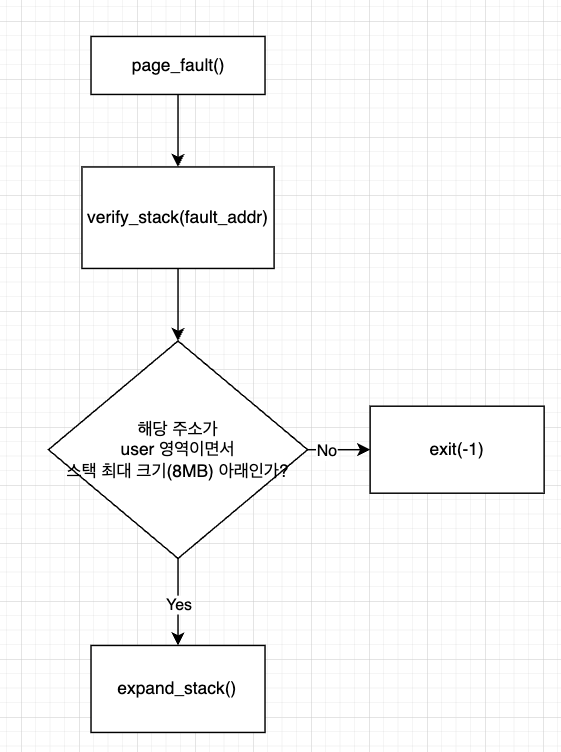
1. page fault handling



2. Disk swap 시 page replacement



3. Stack growth



* 1. **제작 내용**

vm 디렉토리에 추가한 파일은 다음과 같다.

vm/page.c & vm/page.h

vm/frame.c & vm/frame.h

vm/swap.c & vm/swap.h

우선 page table을 구현하기 위한 자료구조를 추가하였다. Pintos 명세서의 페이지 테이블은 Hash table 혹은 bitmap 자료구조로 구현하여 실행속도에서의 이점을 가져가야 한다는 부분을 참고하여 hash table로 구현하였다.

우선 thread 구조체에 hash table을 위한 멤버를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 테이블의 엔트리로 들어갈 구조체 vm\_entry를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

type은 다음과 같은 둘 중 하나이다.

텍스트, 장치, 측정기이(가) 표시된 사진

자동 생성된 설명

VM\_BIN은 바이너리 파일을 로드 한 페이지라는 의미이고, VM\_ANON은 그 외의 경우(swap in / out을 거친 페이지, 스택 확장 등)를 나타낸다.

이 외에 페이지의 쓰기 가능 여부와 load 되었는지 여부, file을 로드하였을 경우 어떤 file인지를 저장하였고, 현재 페이지의 offset과 read\_bytes, zero\_bytes 또한 추가하였다.

또 이후 disk swap을 위한 swap\_slot 멤버도 추가하였고, 마지막으로 hash table로 관리하기 위한 hash element 변수도 추가하였다.

page를 위한 구조체도 추가하였다. 이는 유저에게 할당된 물리 페이지를 나타낸다. vme를 통해 가상 페이지 테이블과 mapping 되어있으며, swap을 위한 lru\_list에서 관리하기 위한 list element 변수도 추가하였다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

페이지 테이블을 관리하기 위한 함수로는 다음과 같은 함수들을 사용하였다. hash.h에 구현된 api를 활용하여 구현하였다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

load\_file() 함수는 다음과 같다.

**텍스트, 스크린샷, 화면, 은색이(가) 표시된 사진

자동 생성된 설명**

우선 offset을 vm entry에 해당하는 offset으로 설정한 후, file\_read() 함수를 사용하여 read\_bytes 만큼 데이터를 물리 페이지에 쓴다. 다음으로 zero\_bytes만큼 남는 부분을 채우고 성공 여부를 반환한다.

다음으로는 swap을 위한 함수들을 구현하였다. swap disk의 경우 마찬가지로 빠른 처리를 위해 bitmap으로 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

swap\_bitmap 전역변수를 통해 남은 swap slot들을 관리하였고 초기화를 위한 swap\_init(), swap\_in()과 swap\_out()으로 swap slot에 페이지를 기록하거나 저장된 데이터를 가져오도록 구현하였다.

다음으로 Second chance 알고리즘을 구현하였다. frame.c와 frame.h에 해당 내용을 추가하였다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

lru\_list로 페이지들을 관리하며 동시성 문제를 방지하기 위해 lock 변수를 사용하였다. lru\_clock는 현재 가르키고 있는 페이지를 나타내는 포인터이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

함수들은 다음과 같이 구현하였다.

초기화를 위한 init, 리스트 추가, 삭제를 위한 add, del 함수는 list.h의 함수를 이용하였다. free\_victim\_pages() 함수는 Second chance 알고리즘에 따라 victim을 선정하여 해당 페이지를 swap\_out하는 함수이다. alloc\_page() 함수는 기존에 사용하던 palloc\_get\_page() 함수에 page 구조체를 할당하여 lru\_list에 추가하는 부분을 구현한 함수이다. find\_and\_free\_page()와 free\_page() 함수는 각각 페이지를 찾아서 할당을 해제하고 lru\_list에서도 삭제하는 함수이다.

init.c의 main 함수에서 swap\_init()과 lru\_list\_init() 함수로 초기화를 진행하고, thread.c의 thread\_create() 함수에서는 vm\_init() 함수를 호출하여 초기화를 진행하였다.

이제 기존 Pintos의 페이지 할당 구조를 바꾸기 위해 기존의 함수들을 수정하였다. process.c/load\_segment() 함수에서는 페이지 테이블 엔트리를 할당하여 구조체의 변수들을 설정해주고 페이지 테이블에 추가하는 부분을 구현하였다.

setup\_stack() 함수에서도 스택에 페이지를 할당하고 이를 페이지 테이블로 관리하도록 수정하였다.

exception.c/page\_fault() 함수에서는 기존에 페이지 폴트가 발생하면 kill() 함수를 호출하던 방식에서 폴트가 발생한 address를 찾아 handle\_mm\_fault() 함수를 호출하도록 하였다.

handle\_mm\_fault() 함수는 vm\_entry를 인자로 받아 해당 엔트리의 type에 따라 처리한다. 만약 VM\_BIN 타입이라면 load\_file 함수를 호출하고, VM\_ANON 타입이라면 swap\_in 함수를 호출한다. 그 후 install\_page 함수를 통해 user address와 kernel address를 매핑하고 엔트리의 is\_loaded를 true로 하여 load됐음을 표시한다.

Stack growth는 expand\_stack()과 verify\_stack() 함수를 사용하였다. 만약 verify\_stack()으로 접근하고자 하는 주소가 유효한 스택 주소임을 확인했다면, expand\_stack() 함수로 해당 주소를 vm\_entry로 만들어 페이지 테이블에 추가하였다.

구현 중 file load와 관련하여 assertion이 자주 발생하였다. 해당 오류의 원인을 찾던 중 기존에 process\_exit() 함수에서 file\_close() 함수를 통해 열었던 파일을 닫아주었기 때문이라는 것을 확인할 수 있었다. 따라서 해당 부분 또한 수정하였다.

* 1. **시험 및 평가 내용**

텍스트이(가) 표시된 사진

자동 생성된 설명

page-merge-\* 테스트 케이스를 제외한 케이스에 pass 하였다.