**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용 교수님

이름 / 학번 : 서윤혁 / 20191597

개발 기간 : 3주

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

이번 프로젝트에서는, page fault에 대한 처리를 추가한다. 물리 메모리가 가득 찬 경우, swap을 할 수 있게 구현하고, page fault가 stack 영역에서 발생하면 stack growth를 구현하여 동작하도록 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  + Page Table & Page Fault Handler

기존 page table에 기존 핀토스에는 구현되어 있지 않은 기능인 virtual address를 physical address로 변환하는 기능을 추가한다. Physical address에 해당 주소가 있지 않을 경우 원래는 page fault가 나면서 프로그램이 종료되었지만, page\_fault\_handler를 구현하여 프로그램이 종료되지 않고 적절한 동작을 수행하도록 한다.

* + Disk Swap

process에 할당할 physical memory가 부족할 때, swap disk에 사용하지 않는 frame을 넣고 새로운 frame을 할당받음으로써 오류가 나지 않게 구현한다. 이 때, LRU 알고리즘을 사용하여 어떤 frame을 swap disk에 넣을지 선택한다. 결과적으로 메모리가 가득차도 프로그램이 계속해서 수행하게 된다.

* + Stack Growth

기존 핀토스에서는 stack의 크기가 4KB로 고정 할당되어 있다. 해당 stack의 용량을 초과하는 영역에 대한 접근이 발생하는 경우 stack을 확장시키는 동작을 구현한다. 이때 유효한 접근인지 먼저 판별하고 유효한 접근이면 스택을 확장한다.

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

가상주소에 대응되는 물리주소가 page table에 존재하지 않는다면 page fault가 발생하게 된다. 이때 page fault handler를 통해서 핸들링을 해주게 된다.

우선 page fault가 일어나면 disk로부터 원하는 데이터를 가져와서 물리 메모리의 frame에 넣는다. 이때 hash table을 사용하여 빠르게 원하는 데이터를 가져올 수 있다. 그리고 frame에 넣어주고나면 page table을 업데이트해서 실제 가상 주소와 물리주소를 매핑해준다.

* + Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술

Pge fault가 발생한 경우에, 기존 물리메모리에 있던 frame을 swap out하고, 그 자리에 새로운 frame을 swap in 시켜준다. 어떤 frame을 희생시킬지는 LRU 알고리즘을 이용하여 구현하였다. 이 알고리즘은 가장 오래 사용하지 않은 frame을 희생시킨다.

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

stack pointer인 esp로부터 최대 확장 영역 이내에 포함되는 영역을 접근한 경우, page fault 발생 시에 스택을 확장한다. 최대 확장 영역은 최대 8MB이다. stack의 시작 주소값에서 8MB를 뺀 것이 Maximum limit으로 설정되고, 이 안에 들어오지 않는 접근은 유효하지 않은 접근으로 간주하여 segmentation fault를 발생한다

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 개발 범위를 포함하여 구현 내용에 대한 일정 작성

23.11.29~23.12.06: Page table & Page fault handler 구현

23.12.06~23.12.07: Disk swap 및 Stack growth 구현

23.12.07~23.12.08: 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
* 1. src/userprog/exception.c

handle\_page\_fault() 함수를 구현한다. handle\_page\_fault는 페이지 폴트 발생 시에 핸들링을 위해서 호출되는 함수이다. Page fault 발생 시에 물리페이지를 할당하고 이를 page table에 맵핑해주는 함수이다.

expand\_stack() 함수는 스택을 확장해주는 함수이다.  
stack\_limit\_check() 함수는 스택이 8mb를 넘었는지 주소가 valid한지를 체크해주는 함수이다.

Page\_fault() 함수는 원래 page fault가 발생하면 원래는 kill을 해주는데 이 코드를 지우고 pte를 찾아서 null이면 expand\_stack을 호출하거나 아니면 handle\_page\_fault 함수를 호출한다.

* 2. src/userprog/syscall.c  
  check\_address() 함수는 주소가 유효한지 체크하고 pte를 리턴한다.  
  check\_valid\_string은 syscall이 호출되었을 때 해당 string이 valid한지 체크한다.  
  check\_valid\_read\_write는 read와 write 시스템콜이 불렸을 경우는 좀 특이하게 size가 들어오게 되는데 들어오는 size만큼 모두 valid 체크를 한다.

3. pintos/src/threads/thread.h

pte를 관리하기 위한 vm hash table을 thread 구조체에 선언한다.

* 4. src/vm/page.h & pintos/src/vm/page.c

Pte 파일을 관리하기 위한 함수들을 구현한다. Pte 구조체에 필요한 변수들을 선언해주었고, 탐색의 효율성을 위해서 hash\_table을 이용하여 구현하였다.

Disk Swap

src/vm/swap.h & pintos/src/vm/swap.c & src/vm/frame.h & pintos/src/vm/frame.c

우선 pintos/src/vm/page.h에 phycial page를 표현하는 자료구조인 page 자료구조를 추가한다. 그리고 swap.c와 swap.h에 swap\_init(), swap\_in(), swap\_out() 함수를 구현한다.  
그리고 frame.c에는 실제로 어떤 frame이 스왑되는지를 lru 알고리즘으로 구현하였고, second\_chance도 이용하였다.  
마지막으로 handle\_page\_fault() 함수에서 pte의 type이 VM\_SWAP이면 swap되도록 수정하였다.

* + Stack Growth

pintos/src/userprog/exception.c에 stack을 확장하는 함수 expand\_stack을 추가한다. 이 함수는 인자로 받는 addr 주소를 포함하도록 stack을 확장한다 이때 page\_fault()함수에서는 스택이 8MB를 넘지 않는지 그리고 올바른 주소를 가지고 있는지를 체크를 해야한다. 이 때 올바른 주소를 체크하는 함수인 stack\_limit\_check 함수를 추가해주었다. 그 후 alloc\_page 함수를 통해 메모리를 할당하고, pte를 할당 및 초기화한다. 그리고 install\_page를 호출하여 page\_table과의 맵핑을 해준다.

1. **연구 결과**
   1. **Flow Chart**

* 개발 내용의 각 항목에 대하여 Flow Chart 작성

<전체적인 Flow Chart>

텍스트, 도표, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* Page Table & Page Fault Handler

텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

* Disk Swap

텍스트, 친필, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

* Stack Growth

텍스트, 도표, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**
* 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 Table & Page Fault Handler

일단 page fault가 일어났을 때 kill이 되지 않도록 해야한다.

이를 위해서 handle\_page\_fault 함수를 정의하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이 함수는 페이지를 할당하고, pte의 type이 VM\_BIN이면 file을 load하고 page\_table에 맵핑해주는 함수이다.  
  
텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
  
그리고 page\_fault가 일어났을 경우 handle\_page\_fault가 불릴 수 있게 exception.c의 코드를 수정해주어야 한다.

이제 가상 메모리를 핸들링하기 위해서 page.h에 pte 구조체를 선언해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Type: pte 타입

Writable: write 가능 여부

Loaded: 물리메모리에 load 되었는지 체크 flag

Is\_second\_chance: second chance를 받았는지 체크하는 flag

Dirty\_bit: 변경된적있는지 체크하는 flag

Uaddr: user addr

Ofs: offset

Read\_bytes: 데이터 크기

Zero\_byte: 0으로 채울 남은 바이트

Swap\_slot: swap 슬롯

File: file

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그리고 이러한 구조체를 관리하기 위해서 thrad.h에 hash 자료구조인 vm을 추가해주었다. 이 vm은 start\_process에서 초기화해준다.  
프로세스가 실행될 때 page fault가 발생하면 pte를 해시 테이블에서 탐색하고, 다 사용한 후에는 제거해준다.

Load\_file에서는 물리메모리 할당 후에 실제 디스크에서 파일을 물리페이지로 load해준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Load\_segment에서는 pte를 생성해주고 해시테이블에 insert하는 과정을 거친다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명  
setup stack에서도 페이지테이블 맵핑에 성공하면 pte를 생성하여 해시테이블에 넣어준다.

Disk Swap

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
frame.c에서 이제 lru알고리즘과 second chance를 이용해서 어떤 frame을 swap할지 결정한다. Page를 할당할 때 공간이 있으면 기존에 할당하는 것처럼 palloc\_get\_page()를 이용하여 할당하고 lru\_list에 추가하고 만약 공간이 없으면 lru 알고리즘을 통해서 사용된 지 가장 오래된 frame을 swap\_out 시키고 새로운 페이지를 swap\_in 시키고 page\_table에 맵핑 시킨다. 이 때 handle\_page\_fault에서 pte의 type이 VM\_SWAP인 경우에 swap 하도록 로직을 추가한다.

그리고 page.h에

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명  
  
페이지 테이블을 위한 page 자료구조를 추가하고, lru알고리즘을 위한 element도 추가해준다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명  
  
lru 알고리즘을 구현한 코드인데, 처음에 lru\_clock은 null인데 begin에서 시작하도록 해주고, 만약 lru\_clock이 끝을 가르키고있으면 다시 처음부터 가르키도록 수정하였다. 나머지 경우에는 next를 가르키도록 되어있다.

그리고 init.c에서 initialize하는 코드를 추가해준다.

스크린샷, 텍스트, 폰트, 시계이(가) 표시된 사진

자동 생성된 설명

Stack Growth  
  
텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Page fault가 발생하였을 때 해당 주소가 stack의 최대범위를 넘어가는지 (8BM)와 유효한 주소인지를 stack\_limit\_check 함수를 통해서 체크하고, 유효하다면 expand\_stack을 이용하여 stack의 영역을 확장하는 코드이다.

**트러블슈팅 기록**

Page-linear.c의 테스트가 통과하지 않아서 printf를 이용하여 디버깅을 진행하였다.

Swap 알고리즘을 진행하는 함수인 retry\_alloc\_page까지는 잘들어가는데 스왑 이후에 계속 커널패닉이 나왔었는데, swap out 이후에 victim의 메모리를 해제할 때 pte만 삭제하였는데 그에 대응되는 physical address의 page까지 같이 삭제해주어야 문제가 발생하지 않는다. 꼭 같이 삭제해주자

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

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

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명