**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 김희진 / 20211530

개발 기간 : 2023.12.01~10

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

이전 project에서는 page fault가 발생하면 system call exit를 이용해 프로그램을 종료했다. 이번 프로젝트에서 우리는 page fault handler를 구현해 이를 개선한다. 또한, 스택 영역에서 page fault가 발생했을 때 스택을 확장할 수 있도록 코드를 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 구현되어 있던 page table에 virtual address와 physical address를 mapping하는 추가적인 기능을 구현한다. OS가 virtual address에 해당하는 physical address를 page table에서 찾지 못하면 즉, page fault가 발생하면 이를 handling하는 page fault handler를 구현한다.

해당 구현을 통해 page fault가 발생하여도 프로그램이 무조건 종료되지 않는다. 그러므로 프로그램이 계속 동작할 수 있게 한다.

* 1. Disk Swap

Physical memory가 가득 찼다면 우리는 LRU 알고리즘을 이용해 가장 최근에 이용되지 않았던 데이터를 disk로 swap시켜, 다른 process가 physical memory를 이용할 수 있게 한다. LRU 알고리즘을 수행하는 방식으로 clock algorithm을 이용한다.

이를 통해, physical memory가 가득 찬 상황에서도 프로그램이 종료되지 않고 계속 동작할 수 있게된다.

* 1. Stack Growth

OS에 할당되어 있는 stack의 영역을 초과하는 access가 발생했을 때, 가능한 영역까지 stack을 확장시키는 것을 구현한다. 이때 가능한 영역이라는 말은 kernel영역을 침범하지 않는 것을 말하며 이를 위해 우리는 주소의 타당성 검사를 해야 한다.

이를 통해, 우리는 프로그램이 더 오래 동작하게 할 수 있다.

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

Page table을 구현함으로써 os는 virtual address를 page table을 통해 physical address로 바꾸어 접근하고자 하는 영역에 access한다. 그런데 이떄, virtual address에 해당하는 physical address가 존재하지 않는다면 page fault가 발생한다.

Page fault handler는 memory reference의 타당성을 검사하고 타당하다면, data의 swap여부에 따라 file system이나 디스크로부터 data를 frame에 fetch시키고 physical address와 연결시킨다. 타당하지 않다면, process의 자원의 연결을 해지하고 process를 종료한다. 이때 reference가 타당한지는 kernel영역에 해당하는 address인지 등을 말한다.

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

Clock algorithm(second chance)을 이용하여 구현했다. 이는 access bit와 dirty bit를 이용하는 alogorithm으로 access bit는 Physical address와 mapping될 때 1로 초기화된다. Disk swap이 일어나야 하는 상황에서, victim pointer가 가리키고 있는 page가 0이라면 해당 페이지의 연결을 끊고 disk swap 시켜준다. 이때 dirty bit가 1이라면 data가 훼손되지 않게 처리해주어야 한다. Victim pointer가 가리키고 있는 page가 1이라면 이 bit를 0으로 바꾸고 다음 page에 대해서 위 과정을 반복한다.

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

Page fault가 일어난 address에 대하여 kernel 영역에 대한 address가 아닌지, stack을 더 할당할 수 있는지 등에 대해서 검사한다. 또한, stack access가 맞는지도 검사하여 모든 상황이 충족되는 경우에만 stack을 확장한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
* 12.01~05 : 매뉴얼, ppt 분석
* 12.06~07 : page table, page fault handler 구현
* 12.08 : disk swap 및 stack growth 구현
* 12.09 : 오류 수정 및 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
  1. Page Table & Page Fault Handler

우선 page fault handler역할을 할 handle\_mm\_fault함수를 userprog/process.c에 구현했다. Handle\_mm\_fault 함수는 userprog/exception.c에서 위에서 설명한 page fault handler가 필요한 상황에서 call된다.

pintos에 내장되어 있는 hash 자료구조를 이용해 vm을 thread 구조체   
(threads/thread.h)에 넣어 page table로 사용했다. Vm/page.h에 page구조체와 vm\_entry를 선언하여 page table을 구현하는데 이용했다.   
page구조체는 thread와 이의 physical address, vm\_entry를 저장하는 역할을 한다. Vm\_entry는 offset, read\_bytes등 data의 정보를 저장하는 역할을 한다.   
또한, page.c에서 vm을 초기화하거나 제거하는 함수, 그리고 virtual address에 해당하는 vme를 찾는 함수 등을 구현하여 사용했다.  
vm은 process가 시작될 때 초기화되며 process가 terminate될 때 삭제된다. 그러므로 userprog/process.c에서 start\_process와 process\_exit에서 이를 구현했다. 그리고 기존에 page table에 page를 만들어주고 있던 load\_segment와 setup\_stack함수에서 기존의 코드를 삭제하고 새로 만들어준 page table에 data가 들어가도록 수정했다.

* 1. Disk Swap

Swap과 관련된 동작 즉, swap 영역에서 memory로 fetch하는 함수와 memory에서 swap 영역으로 data를 fetch하는 함수를 vm/swap.c, swap.h에서 구현했다.

Second chance algorithm에 맞게 disk swap이 일어나야 하는데 이와 관련된 함수는 vm/frame.c와 frame.h에 구현했다. 또한, second chance algorithm을 구현하기 위해서는 page를 list에 차례로 넣어주는 과정이 필요하므로 vm/page.h에 선언했던 page struct에 lru list elem을 추가하고 vm/frame.c에 lru list에 추가하고 삭제하는 함수를 구현했다.

또한, 우리는 page table을 추가로 구현했기에 disk로 swap했을 때 혹은 disk에서 다시 memory로 data를 fetch할 때 기존의 palloc함수를 이용하면 문제가 생긴다. 그러므로 page를 할당하고 free해주는 함수도 vm/frame.c에 구현한다.

* 1. Stack Growth

Userprog/process.c에 stack\_growth함수를 추가하여 stack을 확장시킨다. Stack\_gowth는 인자로 받은 address를 포함하게 stack을 확장시킨다. 이는 handle\_mm\_fault와 마찬가지로 userprog/exception.c에서 stack에 관련된 access인지, stack을 확장시킬 수 있는지등을 검사한 후에 call된다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성
  1. Page Table & Page Fault Handler

텍스트, 도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

* 1. Disk Swap

도표, 텍스트, 라인, 종이접기이(가) 표시된 사진

자동 생성된 설명

* 1. Stack Growth

텍스트, 도표, 라인, 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
  1. Page table & page fault handler

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Physical address와 virtual address를 연결해줄 page struct를 vm/page.h에 선언한다. 이때 vm\_entry의 구성요소는 다음과같다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Type – 해당 data가 binary file인지(vm\_bin) disk에 있는 file인지(vm\_anon) 정보를 담는다.  
is\_loaded – 해당 data가 load되었는지의 정보를 담고있다.  
vaddr – 가상 주소를 담는다.

나머지 요소들은 원래 page table에도 있던 요소들이므로 설명을 생략하고 vm\_entry도 page.h에 선언되어 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

각 thread마다 page table을 가지므로 thread.h의 thread 구조체에 이 역학을 해줄 struct hash vm을 선언한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Page.c에 vm을 초기화 해주는 함수, 제거해주는 함수, 인자로 받은 virtual address에 해당하는 vm\_entry를 찾는 함수, vm에 vm\_entry를 넣어주는 함수, vm에서 vm\_entry를 제거하는 함수를 구현한다.

Init\_vm과 destroy\_vm은 kernel/lib/hash.h에 있는 hash\_init, hash\_destroy함수를 사용해 구현했으며 함수의 인자로 들어가는 함수는 아래에서 설명한다.  
find\_vme도 hash.h에 있는 hash\_find함수를 사용해 인자로 받은 vaddr에 해당하는 hash element를 찾고 hash\_entry함수를 이용해 entry로 반환한다. 이때 thread/vaddr.h에 선언되어 있는 pg\_round\_down을 이용해 인자로 받은 address를 다뤄주는데 이 함수는 인자로 들어간 address의 starting pointer를 반환해주는 함수이기 때문이다.   
insert\_vme와 delete\_vme는 인자로 받은 vm에서 인자로 받은 vm\_entry를 삽입 혹은 삭제하는데 hash.h에 선언되어 있는 hash\_insert, hash\_delete함수를 사용한다. Delete\_vme에선 vme를 free해주는 과정도 필요하다. 두 함수 모두 성공여부에 대해서 true 혹은 false를 반환한다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

Init\_vm과 destroy\_vm에 사용되는 함수들이며 hash.h에서 hash\_init, hash\_destroy의 인자로 들어가는 함수의 설명에 따라 구현했다.

Vme\_addr는 인자로 받은 hash element의 virtual address를 반환하며 cmp\_vaddr은 address에 대한 정렬함수이다. 또한 destructor에서는 frame.c에서 구현할 free\_page함수와 pagedir.c에 있는 pagedir\_get\_page, pagedir\_clear\_page를 이용해서 physical address와 virtual address를 모두 삭제해준다.

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

Process가 시작할 때 즉, userprog/process.c의 start\_process()에서 vm을 init해준다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Process가 끝날 때, 즉 userprog/process.c의 Process\_exit()에서 vm을 없앤다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

Userprog/process.c의 load\_segment()에서 기존에 page를 할당하던 코드를 지우고 새로 선언한 page table의 vm\_entry가 초기화 되도록 설정한다. 이때 file\_reopen()만 사용했을 때 kernel panic이 발생해 file의 크기보다 +1이 되도록 copy하는 과정을 추가했다.   
vm\_entry를 초기화 한 후에 insert\_vme()를 호출해 vm에 초기화된 vme를 삽입한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Userprog/process.c의 setup\_stack()에서도 위와 같이 vm\_entry를 초기화 하고 insert\_vme를 이용해 vm에 넣어준다. 이때, frame.c에서 새롭게 선언해줄 free\_page()와 alloc\_page를 이용해야 우리가 새롭게 만들어준 struct page에 대한 할당이 가능하다.  
또한 이때의 data type은 binary file이 아니므로 vme의 type은 vm\_anon으로 설정해야한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이는 userprog/exception.c에서 handle\_mm\_fault와 stack\_growth를 call해주는 코드이다. 우선 present page가 존재하지 않는 경우나 fault address가 user 영역이 아닌 경우 exit처리한다.

그 후, find\_vme함수를 통해 fault address의 vm\_entry를 찾고 이가 존재하는 경우에는 handle\_mm\_fault를 아닌 경우에는 주소에 대한 타당성을 검사한 후, stack\_growth를 호출한다. 이때 2^23은 stack의 크기에 해당한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이는 handle\_mm\_fault()에서 vm\_entry의 type이 vm\_bin일 때, file을 읽고 page로 load하는 역할을 하는 함수이다. Filesys.h에 있는 filesys\_open함수를 이용해 파일을 열고 file.h에 있는 file\_read\_at함수를 사용해 인자로 받은 entry에 저장되어 있는 read\_bytes와 일치하는 지를 확인했다. page크기에 맞게 떨어지지 않는 부분은 memset과 zero\_bytes요소를 이용해 0으로 채워주었다.

이는 page.c에 선언되어있다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

Handle\_mm\_fault 함수로 userprog/process.c에 선언되어 있다. 인자로 받은 vm\_entry에 is\_loaded를 확인하여 이미 load된 entry라면 return 시킨다. 아니라면, entry의 type을 검사하여 binary file일 경우 load\_file을 호출해 file system으로부터 memory로 읽어오고 binary file이 아닌 경우 swap\_in을 이용해 disk에서 memory로 읽어온다. 그 후, install\_page를 호출해서 page를 만들어주고 entry의 is\_loaded를 true로 바꿔 load했다는 표시를 한다.

* 1. Disk Swap

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명disk swap을 구현하기 위해 vm/swap.c에 구현한 함수들이다. Disk는 공용자원 이기에 swap\_lock을 선언해 critical section을 보호해주었다.

매뉴얼에 나온대로 Bitmap과 block을 이용하여 관련된 함수들을 구현했다. 또한 bitmap.h에 있는 bitmap\_flip과 bitmap\_scan\_and\_flip을 이용해서 swap영역에 있는 값들을 읽고 flip시켜 주었으며 devices/block.h에 있는 block\_read와 block\_write함수를 이용해 page를 block단위로 읽어주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Disk swap을 구현하기 위해서 즉, second chance algorithm을 구현하기 위해서는 lrulist가 필요하고 위는 이와 관련된 함수들이다. 이는 vm/frame.c에 구현되어 있다. 마찬가지로 lock을 사용하여 critical section을 보호해 주었고 LRU를 구현하기 위해 add\_to\_lrulist는 list의 뒤에 page를 삽입하게한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위는 vm/frame.c에 선언된 page를 할당해주는 함수이다. palloc.h의 palloc\_get\_page를 이용해 physical page를 만들어준 후, 해당 page가 비었다면 second chance algorithm을 실행시킨다. 아니라면 malloc을 이용해 page를 만들고 page의 physical address로 palloc\_get\_page를 이용해 만들어준 physical page를 연결한다. 또한 page의 thread요소에도 현재 thread를 연결한 뒤, add\_to\_lrulist를 호출해 lru list에 page를 삽입한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위는 vm/frame.c에 선언된 page를 free하는 함수이다. Lru list를 돌며 인자로 받은 address와 일치하는 page를 Palloc.h의 palloc\_free\_page()를 이용해 삭제한다. 또한, delete\_from\_lrulist()를 이용해 lru list에서도 없애고 page는 free시켜준다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Frame.c에 선언된 second chance algorithm을 실행시킬 함수이다. Accessed bit가 0인 page를 while안에서 찾고 write되었다면 swap\_out 시킨다. 그 후, 해당 page를 delete\_from\_lrulist()를 이용해 lru list에서 지우고 pagedir\_clar\_page(), palloc\_free\_page를 이용해 physical address와의 연결도 끊어준다. 해당 영역도 critical section임으로 lock으로 보호한다.

코드를 좀 더 자세히 들여다 보면, while문은 access bit가 1이면 해당 bit를 0으로 바꾸고 next\_lru\_clock()을 call함으로써 다음 page에 대해 탐색을 진행한다. 또한 while문 밖에서는 dirty bit가 set되었거나 data의 type이 vm\_anon이라면 write되었다는 의미임으로 swap\_out을 시켜주는 것이다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이는 clock\_algorithm()에서 사용된 다음 page로 넘어가게 해주는 함수이다.

List가 비었거나 요소가 하나라면 null값을 반환하고 현재 가리키고 있는 victim pointer가 없는 경우나 list의 끝에 있는 경우는 list의 첫번째 값을 반환한다. 그 외의 경우에는 다음 page를 가리키는 값을 반환한다.

* 1. Stack Growth

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Stack\_growth를 call하는 과정은 앞 서 설명했으므로 생략한다. Stack\_growth는 userprog/process.c에 선언되어 있다.

Alloc\_page()를 이용해 page를 할당해 준 후, page가 할당 되었다면 vm\_entry의 요소들을 세팅해준 후에 insert\_vme(), install\_page()를 이용해 만들어준 vm\_entry가 vm에 들어가게 하고, page와 연결되게 한다. 이 과정이 실패한 경우나 alloc\_page()를 이용해 page할당이 실패한 경우 관련 자원을 free시키고 false를 반환한다.

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

  자동 생성된 설명