**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용 교수님

이름 / 학번 :이나연 / 20171662

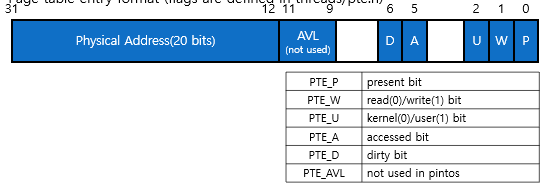
개발 기간 :12/18~12/22

1. **개발 목표**

pintos에선 user program을 동시에 실행될 때의 실행되는 프로세스 개수와 사용되는 메모리가 제한되어 있다. 이번 프로젝트에선 virtual memory를 활용하여 사용가능한 메모리의 범위를 확장하여 page fault가 일어났을 때 프로그램이 더 reliable하게 실행될 수 있도록, 프로그램들도 정상적으로 동작할 수 있도록 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Page Table & Page Fault Handler

Supplemental page table을 구현하고 이를 이용하여 page fault handling을 구현한다. 기존에 pintos에 존재하던 page table에는 제약이 있었기 때문에 supplemental page table을 새로 구현하여 각 page에 대한 추가적인 정보를 저장해야 한다. page table entry의 형식은 아래와 같이 구성한다.



page fault handler의 경우 기존의 page fault는 page fault가 일어나면 어디서 에러가 났는지 출력한 다음 해당 process를 kill하는 방법으로 되어 있었다. 이러한 프로그램 구동 방식을 좀 더 reliable한 방법으로 바꾸어 kill이 바로 되지 않게끔 바꾸어 준다.

* 1. Disk Swap

process에 할당해 줄 physical memory가 부족할 때, 즉 user page pool에 남는 free page가 없을 때 disk로의 swapping out이 필요하다. 여기서 swap out될 page는 page replacement algorithm을 사용하여 결정한다.

* 1. Stack Growth

만약 page fault가 stack에 접근하는 것처럼 보인다면, 다른 page fault와는 다르게 stack page를 할당해주어야 한다. stack의 크기는 최대 8MB로 설정해준다.

* 1. **개발 내용**

1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

page from a file 또는 swap에서 page fault가 일어난다. 기본적으로는 잘못된 영역을 접근했을 때 발생하지만, 그 외에도 디스크에는 존재하지만 memory에 load는 되지 않은 상태의 page에 접근할 때에도 발생하게 된다. 해당 프로그램 내에서는 userprog/exception.c에서의 page\_fault()함수 내에서 세 Boolean 변수 not\_present, write, user를 통해 page fault의 원인을 파악할 수 있게끔 해놓았다.

not present: page fault의 해당 address에 page가 존재하는지 알 수 있음

write: 해당 영역이 writable한지를 알 수 있음

user: page fault가 user space에서 발생한 것인지 알 수 있음

이 중 메모리 공간이 부족하여 page fault가 발생한 상황일 때 이를 handling하기 위해선 stack grow를 해주어야 한다. 잘못된 메모리 공간을 참조한 것이 아니기 때문에 stack의 크기를 확장시켜 현재 fault address를 valid address로 변경하면 된다. 그러한 경우가 아닌 정말 잘못된 address라면, 즉 kernel memory를 참조하였거나 현재 pagedir의 가장 boundary의 주소가 아니라면 exit(-1)을 하게끔 한다.

2. Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술

page를 할당해야 하는 상황에 free page가 존재하지 않는 경우, 즉 해당 프로세스에 물리적으로 메모리가 부족할 경우 존재하는 page들 중에서 disk로의 swap out을 할 page를 골라 수행해야 한다. 이 때 LRU, LFU등의 알고리즘을 사용하여 page를 선택한 다음 swap disk의 slot table을 구축하여 사용된 slot과 비어 있는 slot을 구현하여 관리해야 한다. LRU는 제일 덜 최근에 이용된 page를 swap out하게끔 하는 알고리즘이고, LFU는 제일 덜 (less frequent) 이용된 page를 swap out하게끔 하는 알고리즘이다. 이 때 이러한 기능들은 devices/block.c에서의 block\_read(), block\_write(), block\_get()의 함수들을 사용하여 구현하면 된다.

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

stack 확장 여부를 판단할 수 있는 방법으로는 page fault가 발생한 주소가 stack growth를 통해 8MB 이상이 된다면 접근 불가능한 주소이기 때문에 exit(-1)로 종료시킨다. stack을 grow하는 방법으로는 4 byte 메모리 공간을 가져오는 push 방식과 32byte 메모리 공간을 가져오는 pusha 방식이 존재한다. page를 하나 생성하더라도 메모리 공간이 부족할 수 있기 때문에 현재의 fault\_addr이 valid한 address가 될 때까지 stack의 메모리 공간을 연속적으로 할당한다. 만약 해당 page를 할당하지 못했을 경우 현재 thread를 page fault로 간주하고 종료한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

12/18~12/20: page fault (valid address)

12/21: page fault (stack grow), 보고서

12/22: 보고서 마무리

* 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

userprog/exception.c

page\_fault(): page fault가 발생했을 시 page fault가 발생한 원인을 바탕으로 valid한 reference인지, 맞다면 빈 frame을 가져와 빈 frame이 존재한다면 page swap을 수행하고, 빈 frame이 없다면 process replacement algorithm을 통해 swap out될 page를 고른 다음 page swap을 수행한다. 아니라면 growable region인지 파악하여 stack growth를 수행하든지, 아니라면 kill process를 한다.

새로운 자료구조

Supplementary page table이 필요하다. supplementary page entry의 구조는 위의 그림에 따라 구조체 안에 위의 내용들을 다 넣을 수 있도록 한다.

frame table이 필요하다. frame의 address, supplementary page table entry를 가리키는 pointer, 해당 frame을 가지고 있는 thread의 pointer, frame lock이 피요하다. list\_elem형 element도 포함한다.

2. Disk Swap

swap할 size, swap disk를 나타내는 struct disk 포인터, swap disk에 I/O를 할 때에 swap space에 lock을 걸어주는 semaphore, swap slot을 관리해주는 bitmap이 필요하다.

위의 frame table을 사용하여 swap out될 frame을 victim으로 선정하여 swap out시킨다. 선정 방법은 LRU로 한다.

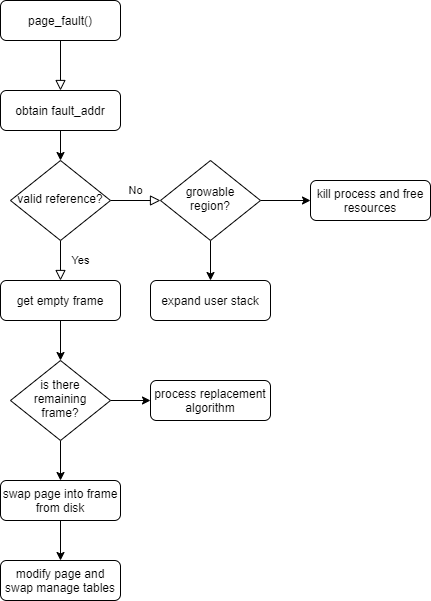
3. Stack Growth

userprog/exception.c

page\_fault()에서 pagedir\_set\_page()를 현재 쓰레드의 pagedir, fault가 일어난 address의 page, 새로 할당된 page를 받아 stack grow가 가능한지 확인한다. stack grow가 가능하다면 page를 할당하고 가능하지 않다면 방금 새로 할당한 page를 할당 해제한다. 해당 과정을 pagedir\_get\_page가 false를 반환할 때 까지 반복한다.

1. **연구 결과**
   1. **Flow Chart**

page fault handler & stack grow

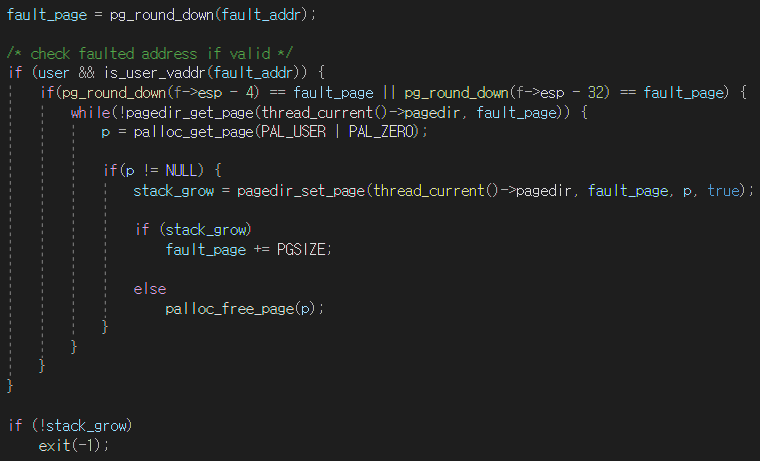


* 1. **제작 내용**

1. Page Table & Page fault Handler & 3. Stack Growth

<userprog/exception.c>

page\_fault():



page fault가 발생했을 때 해당 함수가 발생한다. 앞에선 page fault의 원인을 찾아내고, page fault가 일어난 address를 찾아낸다. 그런 다음 stack growth가 필요한지를 판단하고, stack growth가 필요하지 않는다면 thread를 exit(-1)로 종료한다.

우선 user memory space내에서 일어난 page fault인지, 즉 valid한지를 check하고 pg\_round\_down() 모듈을 사용하여 현재 stack grow할 수 있는 상황인지 파악한다. stack grow가 가능하다면 while문으로 fault address가 valid한 address가 될 때까지 stack을 키운다. 여기서 새로운 page를 할당하기 위해 palloc\_get\_page()함수를 사용하였다.

* 1. **시험 및 평가 내용**
* 