**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 20161598 손동현

개발 기간 : 2020.12.09~2020.12.23

1. **개발 목표**

기존에 구현했던 프로젝트들에서는 page fault에 대한 처리를 해주지 않아 page fault가 발생하면 프로그램이 종료되었다. 이번 프로젝트에서는 해당 page fault를 다룰 수 있게 구현하여 기존처럼 바로 종료되지 않고, 적절하게 반응할 수 있게 만든다. 또한 page, frame 등을 이용한 virtual memory를 도입하여, 효율적인 virtual memory를 구현한다.

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

* Page fault가 발생하는 경우는 READ ONLY 파일에 write를 시도하는 경우, 유효하지 않은 주소를 참조하는 경우, page에 대응하는 frame이 physical memory에 없는 경우, 매핑 되지 않은 page인 경우 등이 있다. 이에 대한 처리를 해주어 기존처럼 page fault 발생 시 바로 종료하는 것이 아닌, 상황에 따라 맞게 처리하여 준다면, 프로그램을 더욱 효율적으로 만들 수 있다.

2. Disk Swap

- physical memory는 한계가 있으므로, 모든 프로그램을 physical memory에 올릴 수 없다.

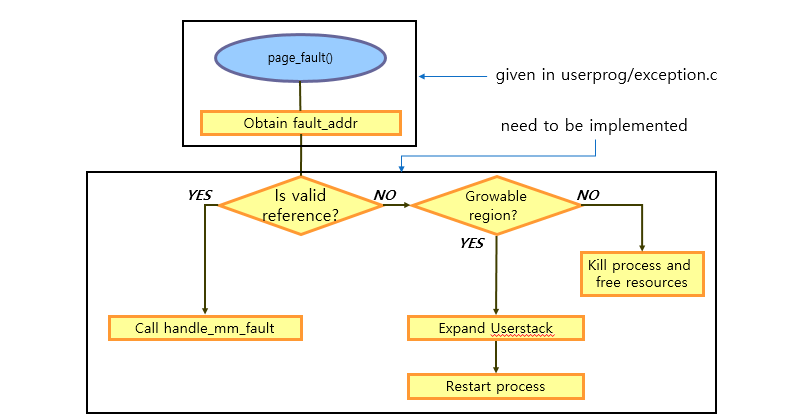
따라서 physical memory가 꽉 찬 경우, 이를 disk에 존재하는 swap slot을 이용하여 frame을 swap in, out 해줄 필요가 있다. 이렇게 한다면, 더욱 더 효율적으로 memory를 관리할 수 있게 된다.

3. Stack Growth

- 이번 프로젝트에서는 여러 개의 page가 존재하므로, user 영역의 stack이 더욱 필요하게 된다. 기존의 page는 4KB, stack은 4MB 였는데, page가 이 크기보다 더 많이 할당된 경우, stack의 크기가 부족해지기 때문에, stack의 크기를 유동적으로 늘릴 수 있게 할 필요가 있다. 이렇게 한 다면, 여러 개의 page를 page fault 없이 다룰 수 있게 된다.

* 1. **개발 내용**
  2. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

- Page fault가 발생하는 경우는 READ ONLY 파일에 write를 시도하는 경우, 유효하지 않은 주소를 참조하는 경우, page에 대응하는 frame이 physical memory에 없는 경우, 매핑 되지 않은 page인 경우 등이 있다. 이를 handling 하기 위해서는 not\_present, write 등의 플래그를 통해 page\_fault 함수 내에서 virtual memory의 kernel 영역이나 data, code 영역을 참조할 경우, 이 접근이 유효한지 판단한다. 전반적인 흐름은 아래 그림과 같다.



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

- disk swap 발생 시, page replace algorithm에는 여러 가지 방법이 있다. 그 중 대표적인 두 가지로 LRU와 FIFO가 있는데, FIFO는 가장 먼저 들어온 page를 계속해서 replace 하는 방법이고, LRU는 가장 오랫동안 사용되지 않은 page를 계속 replace하는 방법이다. 이 두 방식 중, LRU가 평균적으로 효율적이므로 LRU를 사용하는 것이 더욱 좋다.

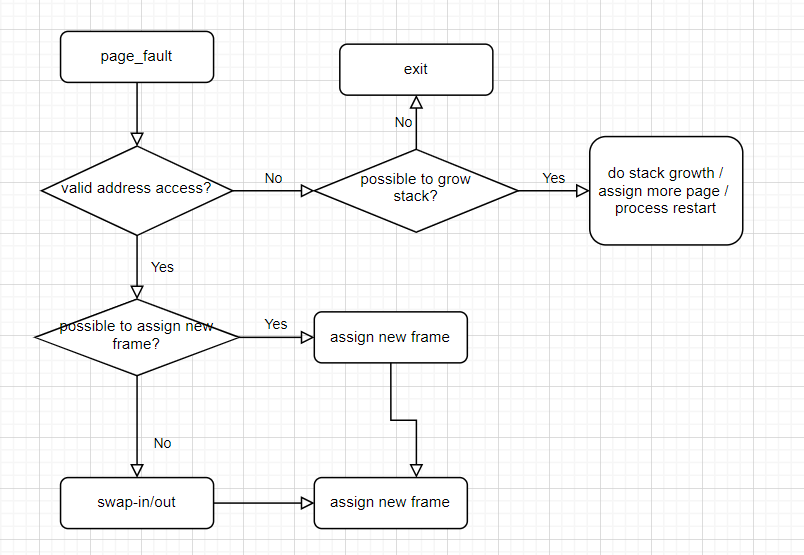
* 1. Stack growth 구현 시 stack 확장 여부를 판단할 수 있는 방법에 대해 서술

- stack growth 구현 시, stack 확장 여부는 fault가 발생한 fault\_address를 확인하여, 해당 address가 유효한 지 확인 후, stack의 주소 값을 증가 시키면서 할당된 page를 지나 할당해야 하는 page의 개수를 세어 필요한 만큼 stack을 증가시키는 방법을 사용할 수 있다.

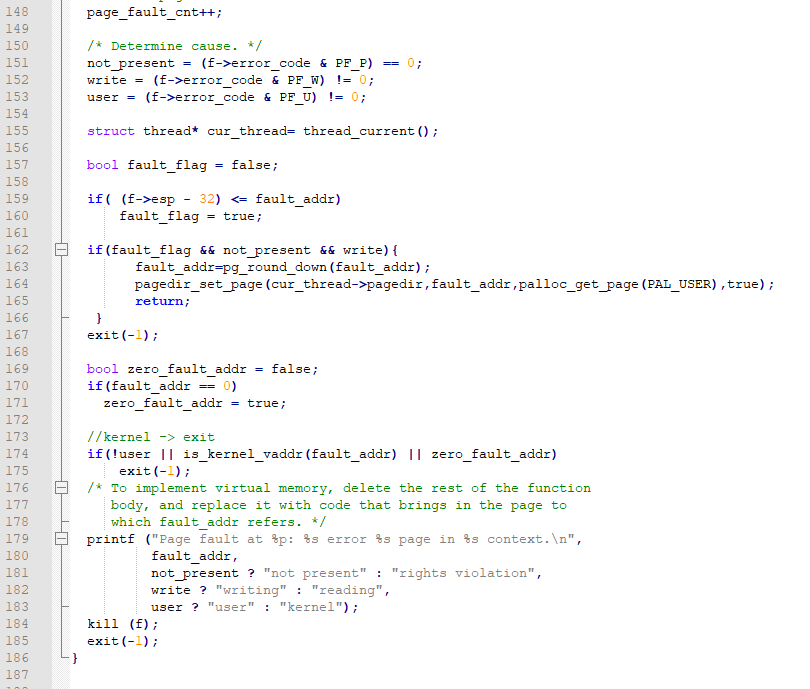
1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 12/19 ~ 12/20 – proj4 동영상 시청 및 명세서 분석   
  12/21 ~ 12/23 – stack growth 구현
  1. **개발 방법**
* userprog/exception.c (page\_fault) : page\_fault가 발생한 경우에 대해 다루는 부분을 추가해야 한다. Stack grow가 가능한지 확인 후, 가능한 경우, thread를 종료시키지 않고, stack을 키울 수 있게 구현한다.

1. **연구 결과**
   1. **Flow Chart**



* 1. **제작 내용**
* userprog / exception.c (page\_fault)



151, 152, 153번째 줄에서 valid한지 체크할 수 있는 flag를 사용한다. not\_present가 참이면 해당 page가 not\_present인 것이고, 만일 false이면, READ\_ONLY 페이지에 write하는 경우이다. Write가 참이면 access가 write인 것이고, false이면, access가 read인 것이다. User가 참이면, user에 의해 access된 것이고, false이면 kernel에 의해 access된 것이다. 기존에는 168~185번까지 있었지만, 여기에 위의 flag들을 사용한 155~167번까지의 줄을 추가한 것이다. 우선 159번 줄로 fault\_address가 범위를 넘어서는지 체크하여, 넘어서면 해당 flag를 참으로 만든다. 그 후, 162번째 줄의 if문은 stack을 grow하기 위한 코드이다. 우선, not\_present이고 write이며, fault\_address가 범위를 넘어서는 경우, ㄴstack을 grow해준다. 우선 vaddr.h의 pg\_round\_down 함수를 사용하여 매개변수로 넘겨준 fault\_address가 가리키는 virtual page의 start를 return 받는다. 그 후, pagedir.h의 pagedir\_set\_page 함수를 사용한다. 이 때, palloc\_get\_page(PAL\_USER)를 통해 user pool에 존재하는 새로운 page를 얻은 후, 해당 virtual page를 현재 실행중인 thread의 pagedir에 mapping하여 준다. 이렇게 하여 stack을 증가시킬 수 있다.

* 1. **시험 및 평가 내용**

