**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 임정호 / 20171682

개발 기간 : ’21.11.08 ~ ’21.11.25

1. **개발 목표**

* pintos 상에서 Page Table Management와 swap disk로의 paging, stack growth를 구현한다.

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

기존의 pintos는 memory를 process가 생성될 때 할당한다. 실제로 필요하지 메모리도 할당될 수 있다. 따라서 메모리를 실제 프로세스가 사용하기 전에 미리 할당을 해놓는 것이 아니라, 실제 사용할 때, 메모리 할당이 이뤄지게끔 한다.

* 1. Disk Swap

사용되는 메모리가 여러 프로세스들이 공유하면서 사용되기 때문에, 메모리가 부족할 경우가 있다. 이런 경우를 위해, disk 공간을 활용하여 어떠한 알고리즘을 통해 메모리를 evict하고 그 공간을 필요한 프로세스가 사용할 수 있게끔 한다.

* 1. Stack Growth

stack address에 접근하는 주소가 할당 받진 않았지만, growable한 범위의 영역인 경우 스택 메모리를 증가해준다.

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

기존의 pintos는 4KB만큼의 PGSIZE만큼 스택 메모리가 할당된다. 각 프로세스는 0부터 3GB의 메모리 영역을 사용할 수 있게끔 구현되는데, 현재는 실제 할당 받은 메모리는 4 KB만큼밖에 되지 않아 가용한 메모리를 전부 사용하지 못 하는 상황이다. 그렇기 때문에 가용한 영역인 경우, 즉 stack이 growable한 범위의 영역인 경우 추가로 메모리를 할당 받아준다.

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

process가 메모리를 할당 받을 때, user page pool에서 free page를 할당 받게 된다.

이 때, free page가 없는 경우 disk swap이 발생한다. 이 때, disk로 evict되는 page는 LRU 알고리즘을 통해 최근에 접근되지 않은 page부터 disk swap이 일어난다.

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

page\_fault()에서 메모리를 할당받고, 현재 process에 page를 install한다. 이 때, 메모리 할당 받는 것을 실패하거나, page install에 실패한 경우 stack 확장이 되지 않은 것으로 볼 수 있다. 모두 성공한 경우 stack 확장에 성공한 것으로 볼 수 있고, make check로 test파일을 실행해보았을 때, 정답 예시와 실행 결과가 같게 나오면 잘 확장이 되었다고 볼 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

’21.11.08 : 전체 강의 시청 후 pintos manual 정독.

’21.11.09 ~ ’21.11.15 : page.c 구현 시도, 실패

’21.11.16 ~ ’21.11.20 : stack-growth 구현 성공

’21.11.21 ~ ’21.11.25 : page.c 구현 재시도, 실패

* 1. **개발 방법**
  + /userprog/exception.c, /vm/page.c, , frame.c, swap.c

page\_fault\_handler 기능 구현과, supplement page table, page frame, disk-swap기능을 위해 필요한 코드이다.

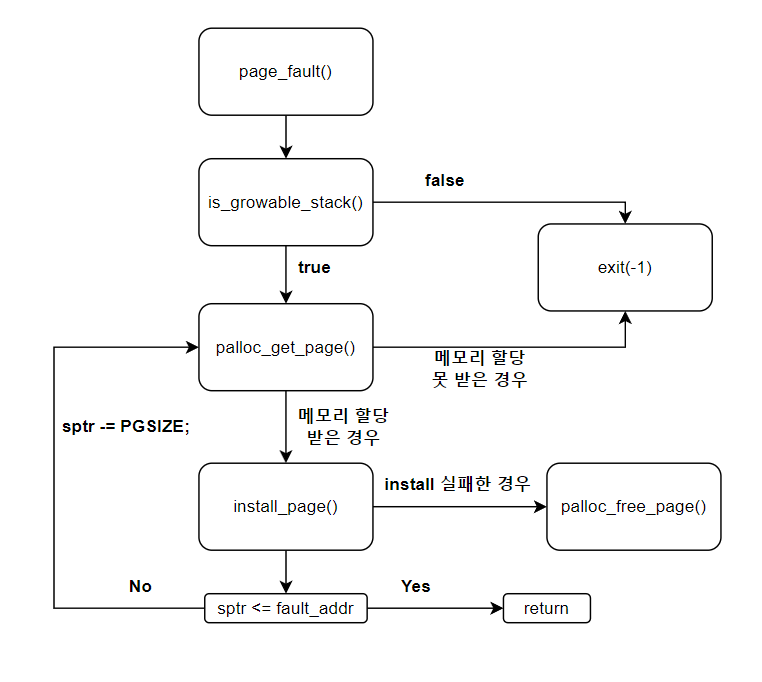
* + page.h, frame.h, swap.h

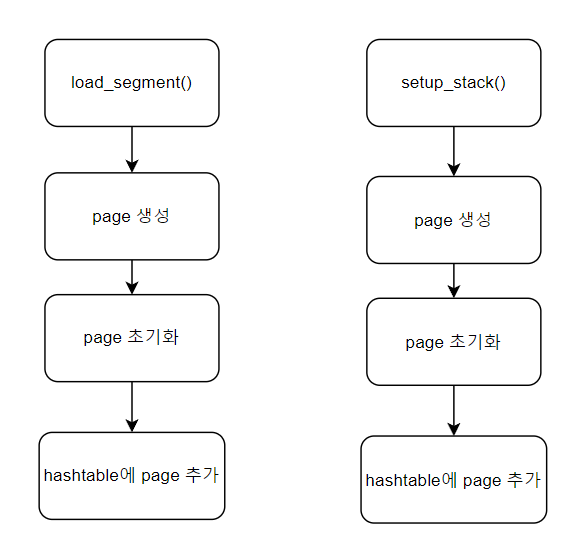
위 코드에서 사용하는 구조체나 함수를 선언하기 위함이다.

* + is\_growable\_stack(), page\_init(), insert\_page(), delete\_page(), find\_page(), page\_destroy\_func(), page\_destroy()

위 코드에서 추가한 함수들이다. supplement page table 구현을 위해 구현한 함수.

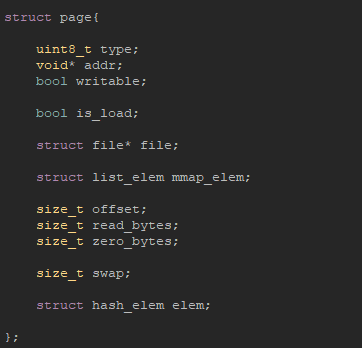
1. **연구 결과**
   1. **Flow Chart**



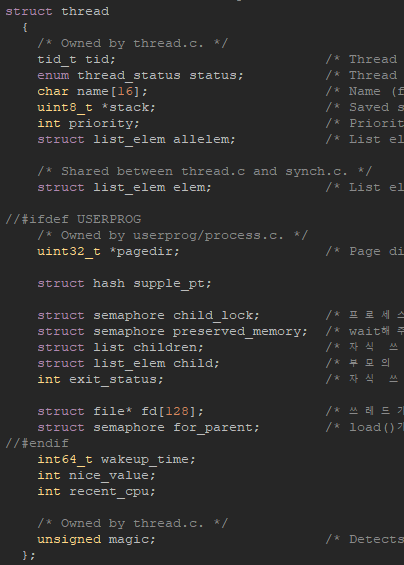


* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

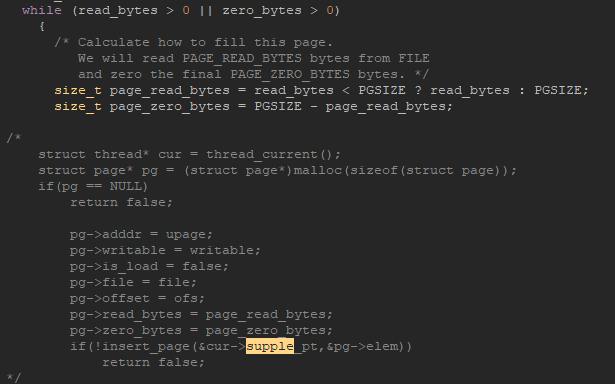
우선 pintos manual과 proj4를 참고하여 vm 디렉토리에 page.c, page.h, frame.c, frame.h, swap.c, swap.h파일을 생성했다. 이 후 기존의 pintos가 사용하는 메모리 할당이 아니라, supplement table page를 사용하는 방식으로 구현하기 위해, page.h에 struct page 구조체를 선언하였다.

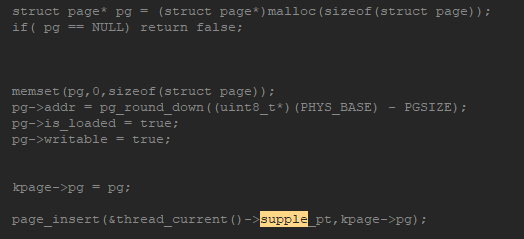


이후 struct thread의 멤버 변수로 struct hash구조체 변수 supple\_pt를 추가하였다.



start\_process()에서 생성되는 thread의 supple\_pt를 초기화하고, load\_segment()와 setup\_stack()에서 현재 thread의 supple\_pt에 생성된 page의 정보를 저장하는 코드를 넣었다.

**(load\_segment)**



**(setup\_stack)**

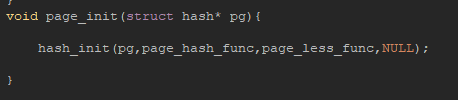
주석 처리한 이유는 결국 kernel panic을 못 잡았기 때문이다.



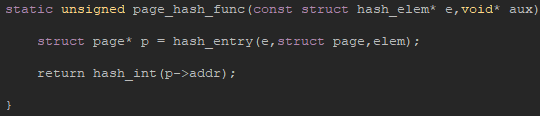
**(process\_exit)**

thread가 사용하던 supple\_pt를 제거하기위해 page\_destroy()를 호출한다.

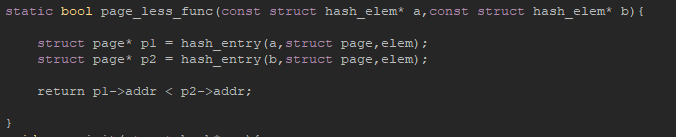
page.c에서 supplement page table 구현을 위해 추가한 함수들은 다음과 같다.



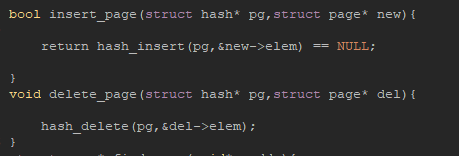
hash\_init()을 이용하여 page를 초기화하는 함수이다.



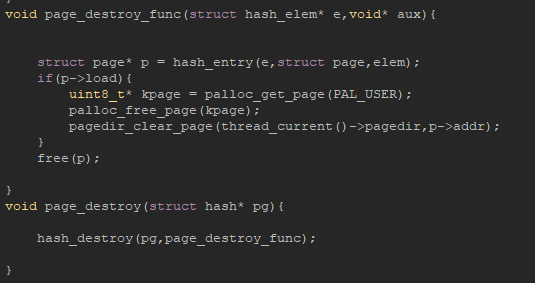
hash\_int()를 이용하여 page의 hash value를 리턴해주는 함수이다.



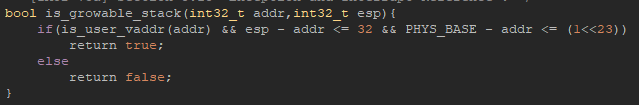
hash\_init()에서 인자로 넣어주는 함수이다. hash table에 어떤 order로 들어갈 지 알려주는 함수이다.



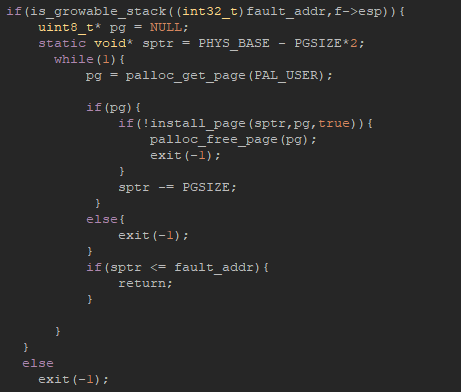
thread의 supple\_pt에 page를 추가하거나, 삭제할 때 사용되는 함수이다.



현재 thread의 supple\_pt를 제거할 때 사용되는 함수로, thread가 exit할 때 supple\_pt가 사용하던 메모리를 반환하기 위해 사용된다.



page\_fault\_handler 구현과 stack growth를 위해 구현한 함수이다. page\_fault()에서 fault\_addr와 struct intr\_frame \* f의 esp값을 사용하여 page fault가 발생한 fault\_addr가 growable한 region의 주소값인지 확인한다. 기존의 pintos의 스택보다 더 자랄 수 있게 하는데, 8MB의 제한이 있기 때문에 검사식에 추가하였다. 이 검사 함수로 pt-grow-bad와 pt-bad-read를 pass할 수 있었다.



page\_fault에 stack growth 기능을 추가한 코드이다. 우선 Growable한 region인지 검사한 후, 아닌 경우 종료한다. Growable한 region인 경우 stack을 확장 시켜준다. 기존의 load\_segment()에서 메모리 할당해주는 부분을 참고하여 구현하였고, palloc\_get\_page()를 이용하여 user pool에서 메모리를 할당받고, 현재 thread의 pagedir에 추가로 받은 메모리를 install\_page()를 이용하여 추가해주었다. 이후 fault\_addr가 가리키는 region만큼 stack 사이즈가 증가한 경우 종료하게끔 구현하였다. 이 부분을 구현하니, 5개를 남기고 모두 pass할 수 있었다.

* 1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부
* 우선 프로젝트 1,2,3은 기존에 있던 코드를 수정하거나 추가하는 작업이 대부분이였지만, 프로젝트 4는 아예 빈 폴더에서 알아서 적절한 코드의 파일을 구현해야했다. 굉장히 막막했고, pintos manualdl나 proj4 ppt자료또한 큰 도움이 되지 않았다. supplement page table을 구현하기 위해 필요한 내용들이 다소 생략되어 있었다. 그래서 타학교 자료를 참고하여 구현하였다. 하지만, 구현을 하면 할 수록, Kernel panic때문에 맞게 구현하고 있는지를 검사할 방법이 없었고, 막막했다. 마음이 조급했고, 순서를 바꿔 stack-growth를 따로 구현하였다. 구현에 성공하고 나서, 다시 supplement page table에 매달렸지만, 아쉽게도 구현하지는 못 하였다.
* 