**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용 교수님

이름 / 학번 : 배성현/20161595

개발 기간 : 2020/12/19~2020/12/23

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Page fault에 대하여 프로그램이 종료되는 것이 아닌, 적절하게 안정적으로 대처할 수 있는 Pintos 프로그램을 만든다. 이를 위하여 Page fault에 대한 handling을 할 수 있도록 page table과 page fault handler를 각각 구현 및 수정한다. 또한 process에 할당해줄 physical memory가 부족한 경우가 생길 수 있기 때문에 이런 경우에 page를 swap out, swap in 할 수 있도록 Disk Swap을 구현하며, stack access에 해당하는 주소에서 발생하는 page fault를 다루기 위한 stack growth를 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

지금까지의 pintos project에서의 page table에는 기본적인 것들만 포함하고 있기 때문에, page fault를 handling 하기에는 제약사항이 많으며 page fault시에 적절하게 대처하는 것이 아닌 프로그램이 종료되어 버린다. 따라서 이를 위하여 supplemental page table과 page fault handler가 필요하며 구현하게 되면 page fault대하여 프로그램이 종료되는 것이 아닌, 특정 page fault에 대하여 적절하게 수행이 되게 된다.

* 1. Disk Swap

Swapping을 지원하지 않는 경우 process에 할당해 주는 physical memory가 부족한 경우가 생길 수 있다. 따라서 이와 같이 physical memory가 부족한 경우 frame들 중에 하나를 victim으로 선정하여 swap out 해주고, 필요한 page를 swap in 하여 주어야 한다. Disk Swap을 구현하게 되면 특정 page replace algorithm에 따라 victim을 선정하게 되고, 해당 victim frame을 disk로 swap out하고 필요한 page를 swap in 하여 사용할 수 있게 하여 Virtual Memory의 개념인 demand paging이 가능하게 된다.

* 1. Stack Growth

지금까지의 pintos project에서는 virtual address space의 stack의 일부분만을 사용하였다. 따라서 stack이 부족한 경우가 생겨 page fault가 발생하게 되므로 이렇게 page fault가 발생한 경우 user영역의 stack을 늘려줄 필요가 있다. Stack growth를 구현하게 되면 stack이 부족하여 생긴 page fault에 있어서 stack을 늘려주게 되므로 프로그램이 해당 page fault를 잘 dealing할 수 있게 된다.

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

Page fault의 발생은 먼저 invalid한 access인 경우가 있다. 이는 page가 mapping되어 있지 않은 경우, 즉 데이터가 없는 빈 페이지인 경우에 발생하게 되며 또한 read only page에 write을 하기 위해 접근하는 경우에 발생하게 된다. 다음으로는 not in memory(valid reference)가 있다. 이 경우는 해당 page가 physical memory에 올라와 있지 않는 경우 발생하게 된다. 이를 핸들링 할 때에는 page fault가 발생하였을 때, 위의 두 경우 중에 어떤 경우에 해당하는지를 먼저 판단하도록 한다. 이 후 invalid access인 경우에는 exit을 통하여 process가 종료되도록 한다. 만약 not in memory(valid reference)인 경우에는 먼저 free frame을 찾고, free frame이 없다면 victim을 정해 frame을 얻어오도록 한다. 이후 해당 page의 data를 frame으로 가져오고 page table entry가 physical frame을 가리키도록 한다.

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

Disk swap 발생 시에 FIFO Algorithm을 사용하도록 하였다. 해당 replacement algorithm은 victim frame을 찾을 때에 가장 오랫동안 physical memory에 올라와 있었던 frame을 victim frame으로 선정하게 된다. 해당 알고리즘을 사용하게 되면 구현은 쉽다는 장점이 있으나 Belady’s Anomaly가 발생할 수 있다는 단점이 있다.

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

PUSH, PUSHA Instruction은 각각 4bytes, 32bytes를 한 번에 집어넣게 된다. 또한 stack pointer를 조절하기 전에 access permission을 체크하기 때문에 이 경우 stack pointer의 각각 4bytes, 32bytes 밑에서 page fault가 발생하게 된다. 따라서 fault address와 intr\_frame의 esp(stack pointer)를 비교하여 이 경우(fault\_address>=esp-32)를 체크하여 stack growth(stack 확장 여부)을 판단할 수 있다. 또한 stack을 확장함에 있어서 8MB로 제한되어 있기 때문에 grow하는 stack의 SIZE가 8MB가 넘어가는지의 여부를 체크한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

|  |  |
| --- | --- |
| 일정 | 내용 |
| 12/19 | 숙제 강의 자료 및 Pintos Manual 확인 |
| 12/20 | Page Table & Page Fault Handler구현 |
| 12/21 | Disk Swap 구현 |
| 12/22 | Stack Growth 구현 |
| 12/23 | Document 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

Page table을 구현하기 위하여 먼저 각 thread들이 자기 자신의 page table을 가질 수 있도록 src/threads/thread.h의 thread구조체에 hash자료구조로 page\_table을 만들어 주고 프로세스 시작 시 init 해줄 수 있도록 할 것이다. 이후 src/vm에 page table entry와 frame entry를 나타낼 수 있는 구조체 및 기능들을 표현할 수 있는 헤더파일과 c파일들을 추가하여 줄 것이다(page.h, page.c, frame.h, frame.c). page.h에는 page\_entry라는 구조체를 만들어서 특정 thread의 page\_table에 각 page의 mapping 정보가 저장될 수 있도록 하고 frame.h에도 frame\_entry라는 구조체를 만들어서 이를 list(frame\_table)로 관리하도록 하여 실제로 physical memory에 올라와 있는 frame을 관리할 수 있도록 할 것이다. 이후 page.c에는 page table을 관리할 수 있도록 page를 추가하는 page\_insert(), page table entry를 지우는 page\_delete(), 특정 page의 entry를 반환하여 주는 get\_page\_entry()함수를 구현하여 줄 것이다. frame.c에는 실제 메모리에 올라와 있는 frame들을 list로 관리하는데 있어서 필요한 함수들, 즉 list에 frame entry를 추가하는 frame\_insert()함수, list에서 frame\_entry를 삭제하는 frame\_delete()함수, 특정 frame의 entry를 반환하여 주는 get\_frame\_entry()함수를 추가하여 줄 것이다. 그리고 page fault handling을 위하여 userprog/exception.c의 page\_fault()함수를 수정하여 줄 것이다. 그리고 이 함수안에서 어떠한 이유로 page fault인지를 check하는 코드를 넣어주어 invalid인 경우에는 exit하는 코드를, not in memory인 경우에는 swap\_in해주는 코드를, stack\_growth가 필요한 경우에는 stack\_growth를 해주는 코드를 추가해 줄 것이다.

* 1. Disk Swap

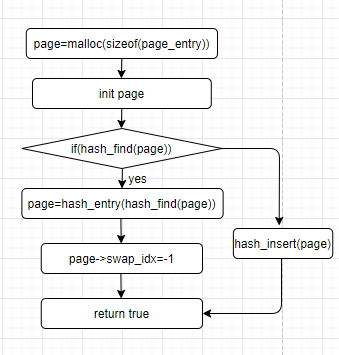
Disk swap을 구현하기 위해서 src/vm에 먼저 swap.h(필요한 함수들의 prototype 정의)와, swap.c(함수 구현)라는 파일을 만들어 줄 것이다. 이후 swap\_disk영역을 나타낼 수 있도록 하는 block pointer형 swap\_disk와 swap\_disk의 각 block이 사용되는지 아닌지를 판단할 수 있도록 하는 swap\_used 배열을 전역변수로써 선언하여 줄 것이다. 그리고 swap\_disk와 swap\_used를 초기화 할 수 있도록 하는 swap\_init()함수와, swap\_disk로부터 실제 physical memory로 page를 올릴 수 있도록 하는 swap\_in()함수, physical memory로부터 swap\_disk로 page를 내릴 수 있도록 하는 swap\_out()함수를 추가하여 줄 것이다. 또한 free frame이 없는 경우 physical memory로부터 frame을 뺐어와야 하므로 vm/frame.c에 fifo replacement algorithm을 적용하여 replace victim을 physical memory에서 해제하여 주는 evict\_frame()라는 함수를 만들어 줄 것이다.

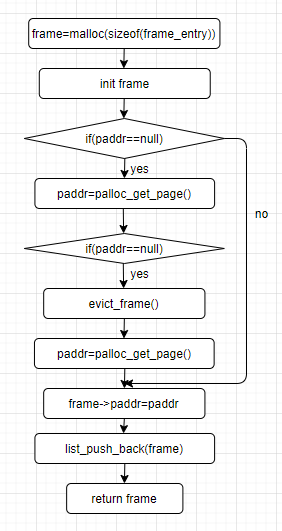
* 1. Stack Growth

Stack\_growth를 구현하기 위하여 /src/userprog/exception.c의 page\_fault함수를 수정하여 줄 것이다. page\_fault함수에서 valid한 reference인 경우에서 또 세부적으로 나누어 fault address>=stack\_pointer-32인 경우이고 grow할 stack의 크기가 8MB보다 작은 경우이면 stack\_growth가 필요한 경우이므로 이 때 stack을 grow해주고 해당하는 page들을 physical memory로 mapping해줄 것이다.

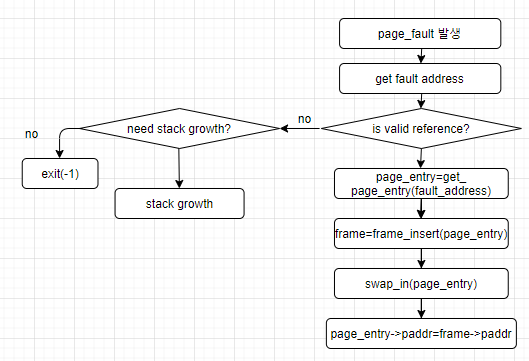
1. **연구 결과**
   1. **Flow Chart**

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

 Page\_insert

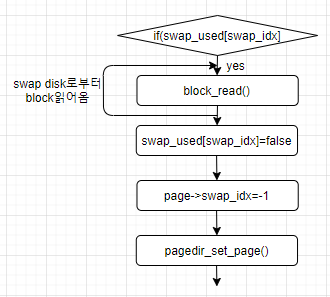
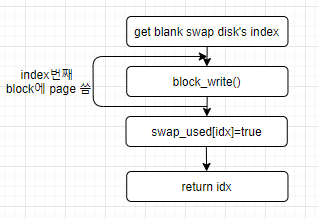


Frame\_insert

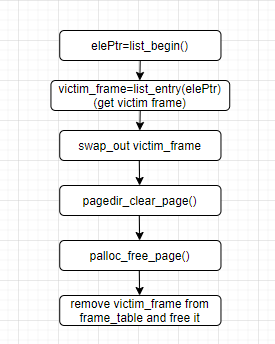


Page fault handling

* 1. Disk Swap

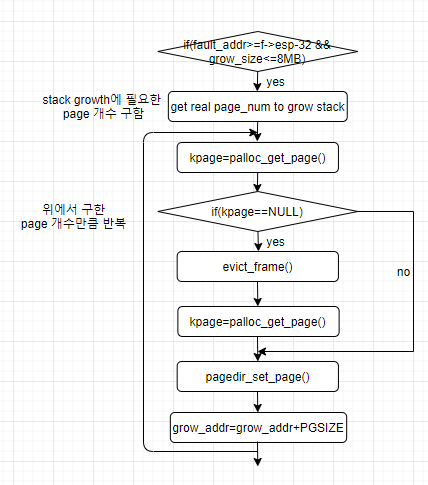
 

SWAP\_IN SWAP\_OUT



Evict\_Frame

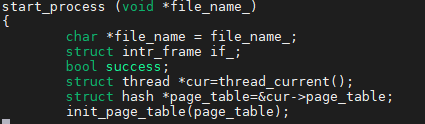
* 1. Stack Growth



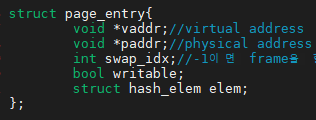
* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명
  1. Page Table & Page Fault Handler



먼저 각 thread들이 자기 자신의 page table을 가질 수 있도록 src/threads/thread.h의 thread구조체에 hash자료구조로 page\_table을 추가해 주었다.

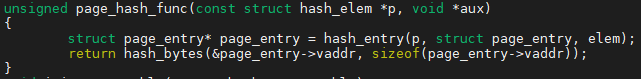


이후 src/userprog/process.c의 start\_process()함수에서 각 thread들이 page\_table을 init하여 줄 수 있는 함수인 init\_page\_table() 함수를 호출하여 주도록 하였다.

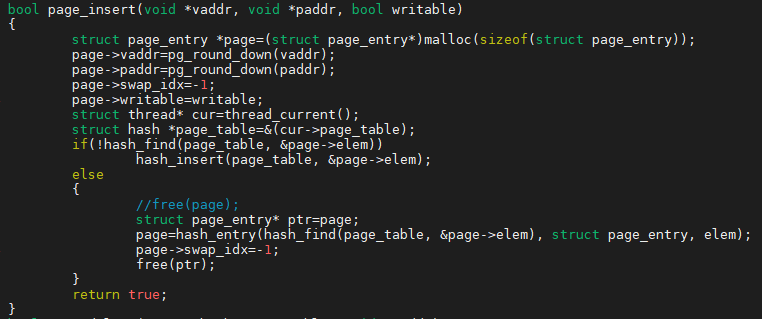


이후 page.h를 src/vm에 만들고 그 안에 page\_entry라는 구조체를 만들어서 각 page의 frame으로의 mapping 정보를 저장할 수 있도록 하였다.

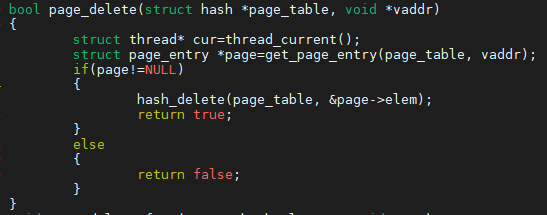




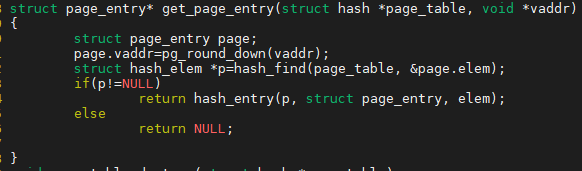
또한 src/vm에 page.c를 만들고 그 안에 page table을 핸들링 하는 여러 함수들을 추가하였다. 위의 init\_page\_table은 초기에 start\_process()함수에서 호출되어 각 thread의 page\_table(hash자료구조)를 초기화 하여 주게 된다.



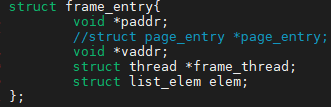
위의 page\_insert함수를 src/vm/page.c에 추가하였고 이 함수에서는 위와 같이 virtual address와 physical address, writable에 대한 정보를 받아 해당 mapping 정보를 entry로써 저장하여 thread의 page table에 저장하게 된다. 이 때 page\_table에 이전에 해당 entry가 있었다면 새로 추가하는 것이 아닌 해당 entry를 update 해주도록 하였다.



위의 page\_delete함수를 src/vm/page.c에 추가하였고 이 함수에서는 위와 같이 page\_table과 virtual address를 받는데, page\_table에서 virtual address에 해당하는 page entry를 찾고 해당 entry를 page\_table에서 제거해 주도록 하였다.



위의 get\_page\_entry함수를 src/vm/page.c에 추가하였고 이 함수에서는 위와 같이 page\_table과 virtual address를 받는데, hash\_find를 통해 page\_table에서 virtual address에 해당하는 page entry를 찾아서 해당하는 entry를 반환하여 준다.



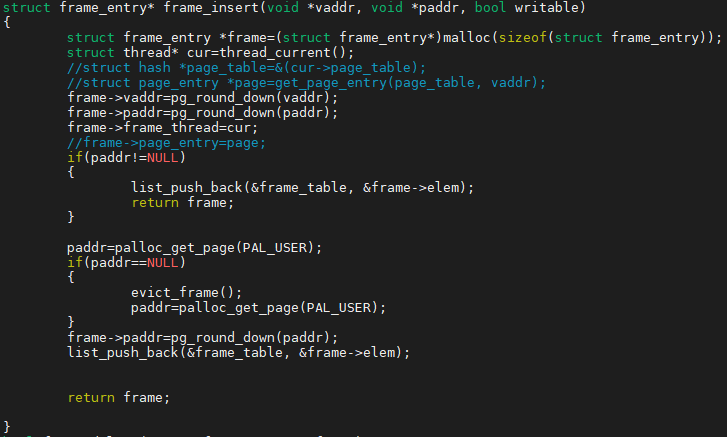


또한 frame.h를 src/vm에 만들고 그 안에 frame\_entry라는 구조체를 만들어서 각 page의 frame으로의 mapping 정보를 저장하고 frame\_table(list 자료구조) 전역 변수를 만들어 실제로 physical memory에 올라와 있는 frame\_entry들을 list로써 관리할 수 있도록 하였다.

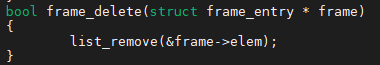




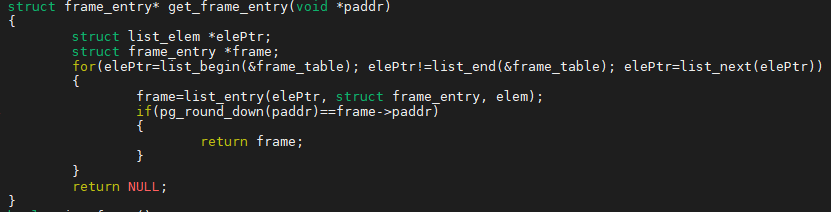
또한 src/vm에 frame.c를 만들고 그 안에 frame\_table을 핸들링 하는 여러 함수들을 추가하였다. 위의 frame\_init은 threads/init.c의 main함수에서 호출하여 주어 pintos 시작시 frame\_table(list)을 초기화 하여 줄 수 있도록 하였다.



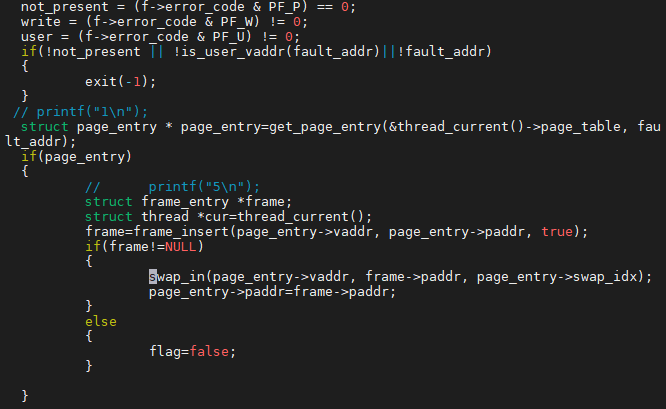
위의 frame\_insert함수를 src/vm/frame.c에 추가하였고 이 함수에서는 위와 같이 virtual address와 physical address, writable에 대한 정보를 받아 해당 mapping 정보를 frame\_entry로 저장하여 frame\_table에 추가하여 줄 수 있도록 하였다. 이 때 physical address가 null인 경우에는 free frame을 할당하여 주어야 하므로, free한 frame이 없는 경우에는 evict\_frame()호출하여 replacement algorithm에 따라 free frame을 만들어줄 수 있도록 하고 이에 따라 virtual address와 physical address를 mapping할 수 있도록 하였다.



위의 frame\_delete함수를 src/vm/frame.c에 추가하였고 이 함수에서는 위와 같이 frame\_entry를 받아 해당 entry를 frame\_table에서 삭제할 수 있도록 하였다.



위의 get\_frame\_entry함수를 src/vm/frame.c에 추가하였고 이 함수에서는 위와 같이 physical address를 받아 이를 가지고 frame\_table을 순회하여 해당하는 entry를 찾아 반환한다.

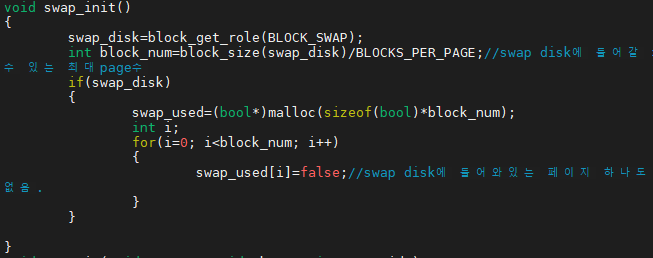


그리고 page fault handling을 위하여 userprog/exception.c의 page\_fault()함수를 수정하여 주었다. 먼저 위와 같이 valid한 access인지를 체크하고 valid하지 않으면 exit을 해서 process를 종료할 수 있도록 하였다. 그리고 not in memory(valid reference)인 경우에는 fault address에 대한 page\_entry가 current thread의 page table에 있는지를 체크하고 있으면 swap out한 page이므로 해당 page를 swap\_in하여 사용할 수 있도록 하였다.

* 1. Disk Swap

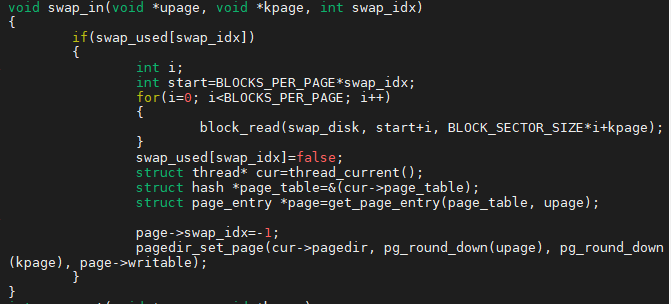


먼저 src/vm에 swap.c파일을 만들고 해당 파일에 swap\_disk영역을 나타낼 수 있도록 하는 block pointer형 swap\_disk와 swap\_disk의 각 block이 사용되는지 아닌지를 판단할 수 있도록 하는 swap\_used 배열을 전역변수로써 선언하여 주었다.

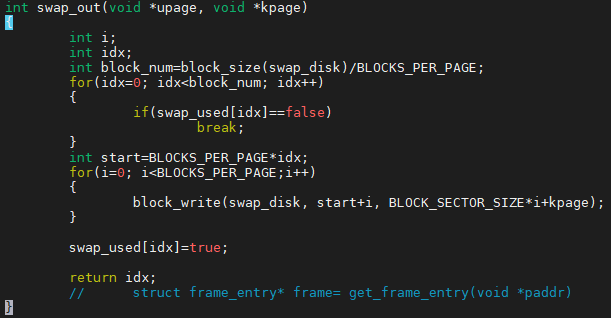




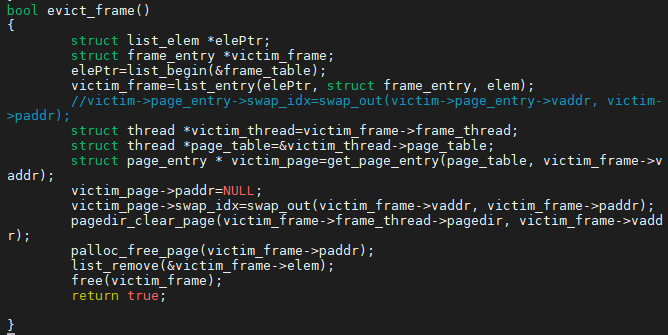
또한 위와 같이 swap\_init()함수를 만들어 이 함수에서 block\_get\_role()을 통해 swap space를 할당 받아 swap\_disk를 초기화 하고, 또 초기에는 swap\_disk의 모든 block이 사용되지 않기 때문에 swap\_used를 모두 false로 초기화 할 수 있도록 하였다. 그리고 이러한 swap\_init을 위와 같이 threads/init.c의 main함수에서 호출하여 주었다.



또한 위와 같이 swap\_disk로부터 실제 physical memory로 page를 올릴 수 있도록 하는 swap\_in()함수를 만들어 주었다. Virtual address와 physical address, swap\_idx를 받아서 swap\_disk의 swap\_idx번째 block에서 해당하는 page를 읽어 올 수 있도록 하였고 또 읽어 왔으므로 swap\_used의 swap\_idx에 해당하는 block을 false로 바꾸어 주고 해당 page\_entry의 swap\_idx를 -1로 바꾸어 주었다. 그리고 pagedir\_set\_page를 통하여 virtual address와 physical address를 mapping하여 주도록 하였다.



또한 위와 같이 physical memory로부터 swap\_disk로 page를 내릴 수 있도록 하는 swap\_out()함수를 추가하여 주었다. virtual address와 physical address를 받아주었고 swap\_disk로부터 비어있는 block의 인덱스를 swap\_used를 통하여 찾고 이를 이용하여 swap\_disk의 index번째 block에 해당하는 page를 저장할 수 있도록 하였다. 또한 저장하였으므로 swap\_used의 idx에 해당하는 block을 true로 바꾸어 주고 해당 idx를 반환할 수 있도록 하였다.



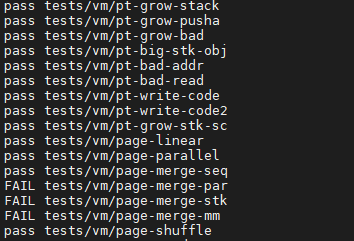
또한 free\_frame이 없는 경우 victim frame을 찾아 주어야 하므로 위와 같이 replace algorithm을 통하여 victim frame을 찾아주는 evict\_frame()함수를 src/vm/frame.c에 추가하여 주었다. FIFO replacement algorithm을 사용할 것이고 실제 메모리에 올라와 있는 page들을 frame\_table(list)로써 관리하고 있기 때문에 frame\_table의 가장 첫 원소가 가장 오랫동안 physical memory에 올라와 있던 frame(victim frame)이다. 따라서 list\_begin()을 통해 가장 첫 원소를 받고 이를 victim frame으로 하여 해당 physical address에 대해 pagedir\_clear\_page와, palloc\_free\_page를 통하여 free frame으로 만들어 주었다.

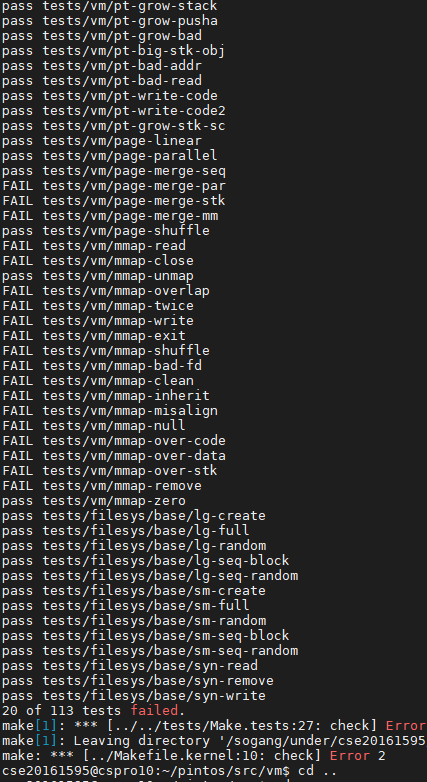
* 1. Stack Growth



Stack\_growth를 구현하기 위하여 위와 같이 /src/userprog/exception.c의 page\_fault함수를 수정하여 주었다. page\_fault함수에서 valid한 reference인 경우에서 또 세부적으로 나누어 fault address>=stack\_pointer-32인 경우를 체크하고 grow할 stack의 크기가 8MB보다 작은 경우이면 stack\_growth가 필요한 경우이므로 이 경우에 stack growth를 수행할 수 있도록 하였다. 이 때 stack의 page중 이미 frame에 mapping된 것이 있을 수 있으므로 해당 부분을 빼주고 stack을 grow해주게 된다. Stack grow의 경우 grow해야하는 page만큼 palloc\_get\_page를 통하여 physical memory의 frame공간을 새로 할당받아 page와 mapping하여 주도록 하였는데 이 때 physical memory가 이미 꽉 차 있어 할당 받을 수 없는 경우가 있을 수 있기 때문에 이 경우에는 evict\_frame()을 통하여 free\_frame()을 만들어 준 뒤 frame을 새로 할당 받을 수 있도록 하였다.

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

 (page-merge-par부터 3개 실패)

 (학번까지 보이도록 캡처)