**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 한석기 / 20211606

개발 기간 : 23.11.20 ~ 23.12.10

1. **개발 목표**

기계의 한정된 main memory 크기를 커버할 virtual memory를 구현한다. hardware page table을 보충함으로써 page fault handling을 가능케하고 swap disk를 통해 swap 또한 가 능케한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. Page Table & Page Fault Handler

현재 Pintos는 한정된 물리메모리 공간으로 답답한 면이 있다. 따라서 virtual address를

physical address로 mapping할 수 있는 page table을 구현한다. virtual address를 갖는 page table을 구현하고 page table을 참조해 physical address를 찾을 때, 없으면 page fault를 발생시킨다. 따라서, page fault handler 구현을 통해 page fault 발생 시, 프로그램 이 종료되지 않게 한다.

2. Disk Swap

physical memory 공간이 부족할 때, disk로 swap이 일어나게 구현한다. second chance

algorithm을 이용해 새로운 frame을 할당 받고 swap을 진행한다. 이로 인해 physical memory 공간이 부족하더라도 프로그램이 계속 돌아갈 수 있다.

3. Stack Growth

현재 Stack은 4KB로 고정되어 있는데 4KB를 넘는 주소로 접근했을 때, validate한 지, 확 인하고 valid하다면 Stack을 Growth한다.

* 1. **개발 내용**

1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

` 본래 사용 불가능한 영역에 접근할 때, page fault가 발생했다. 하지만 page fault handing 구현으로 virtual address로 physical address에 접근할 때, 대응되는 physical address가 없을 때, page fault가 발생한다. physical address와의 mapping까지 구현하기 위해 supplementary page table entry struct를 이용했다. supplementary page table entry 를 disk에서 가져와 physical address에 frame형식으로 넣는다. 쉽게 찾기 위해 hash table 을 이용하고 frame list로 mapping을 관리한다.

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

frame list를 관리할 때, second chance algorithm을 이용한다. frame list를 돌며 accessed bit가 1일 경우 accessed bit를 0으로 주어 chance를 한 번 주고 accessed bit가 0일 경우 해당 frame entry가 victim이 된다. 그렇게 swap 과정이 일어난다.

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

현재 stack pointer를 의미하는 esp pointer에서부터 stack grow max size 내에 영역에 접 근할 시, stack을 확장한다. stack grow max size는 최대 8MB이며 비정상적인 접근 시, exit(-1)로 예외처리해 비정상 종료한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

23.11.20 ~ 23.11.27: pintos manual 읽기 및 Prj4 ppt 설명 읽기

23.11.28 ~ 23.12.04: vm folder의 page, frame, swap 구현

23.12.05 ~ 23.12.09: process.c, syscall.c, exception.c, init.c, thread.h 수정

23.12.10: test 및 보고서 작성

* 1. **개발 방법**

1. Page fault를 handling하는 전반적인 과정

vm 폴더에 page.c, page.h를 추가해 supplementary page table entry에 해당되는 struct 를 선언한다. 그리고 hash list를 활용해 빠른 탐색과 접근으로 supplementary page table entry를 관리한다. hash list를 초기화하는 함수, 넣는 함수, 삭제하는 함수 등을 구현한다.

process.c에서 stack을 늘리는 함수와 physical address와 mapping하는 함수를 구현한다.

syscall.c에서는 process.c에서 구현한 함수들을 활용해 address를 검사하고 exception.c 의 page\_fault 함수에서 page fault handling을 진행한다.

2. Disk swap 발생 시 사용한 page replacement algorithm

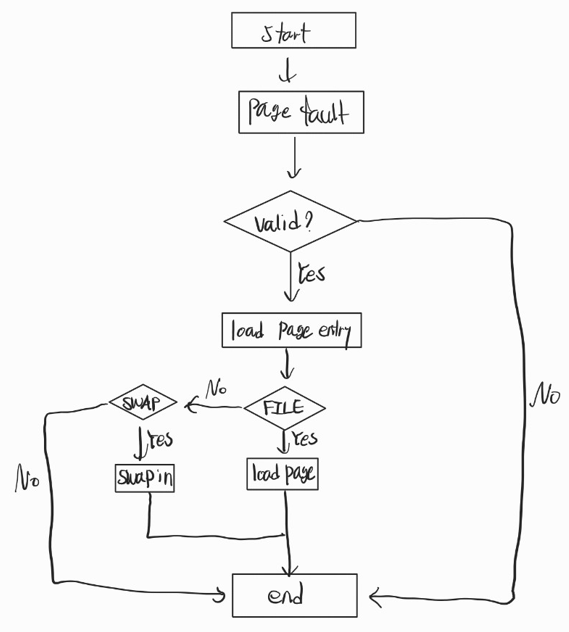
vm 폴더에 frame.c, frame.h를 추가해 frame table entry에 해당되는 struct를 선언한다. 그리고 이 frame table entry를 관리할 lsit를 thread.h에 넣었다. 관리과정에서 사용할 lock 또한 thread.h에 선언했고 frame.c에서 삽입, 삭제, second chance algorithm 등의 함 수를 구현했다. thread.h에 현재 보고 있는 frame table entry를 저장하고 이것을 활용해 frame.c의 함수를 구현했다.

3. Stack growth 구현

process.c에 Stack growth를 구현하고 syscall.c에서 주소가 유효한 지, 검사할 때, stack growth가 필요하다면 호출했다. 그리고 exception.c에서 page fault handling을 진행할 때, stack growth가 필요하다면 호출했다.

1. **연구 결과**
   1. **Flow Chart**

1. Page fault handling



2. Disk swap

스케치, 그림, 도표, 텍스트이(가) 표시된 사진

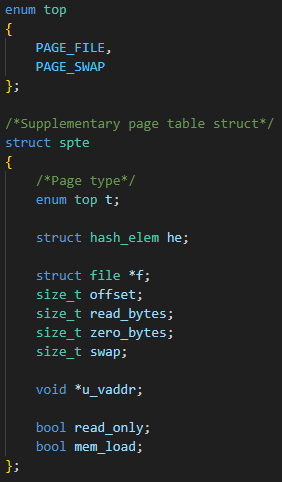
자동 생성된 설명

3. Stack growth

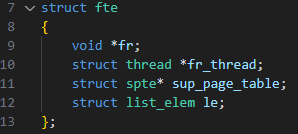
텍스트, 그림, 스케치, 도표이(가) 표시된 사진

자동 생성된 설명

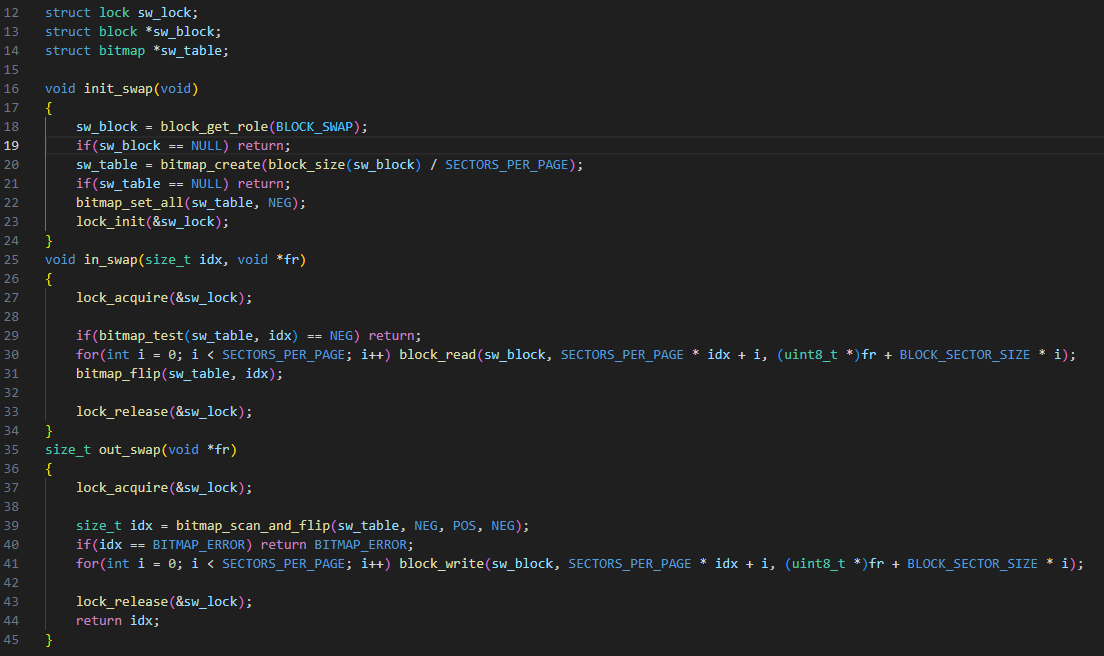
* 1. **제작 내용**



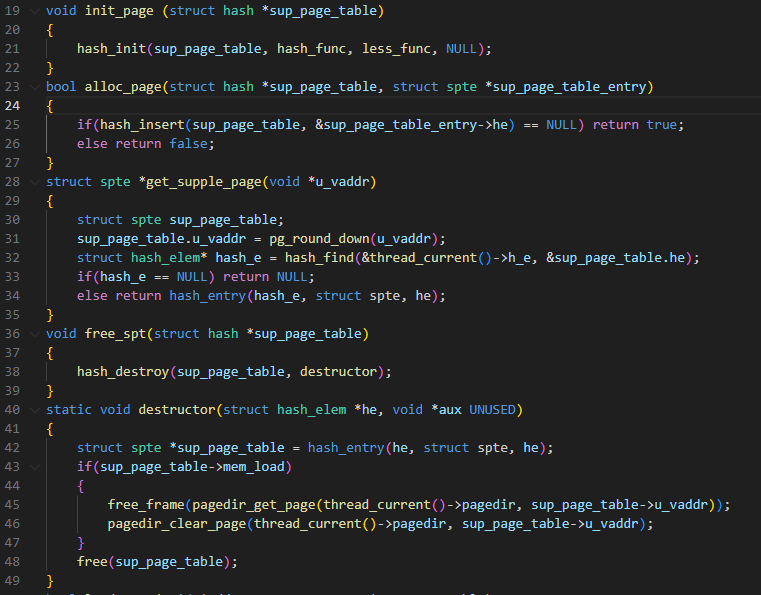
page.h의 supplementary page table entry struct 부분이다. type이 FILE인지 SWAP인지 구 분을 enum으로 진행했고 해당되는 hash element, file, offset, read bytes, zero bytes, swap slot, virtual address, 읽기 전용인지 확인하는 변수, 메모리에 적재되어 있는지 확인하는 변수를 선언했다.



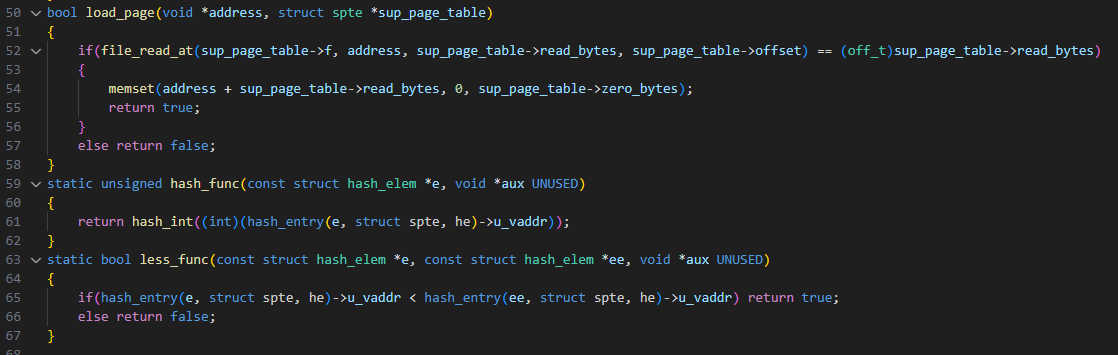
frame.h의 frame table entry struct 부분이다. frame에 해당되는 address, frame의 thread, frame의 supplementary page table entry, frame element를 변수로 선언했다.



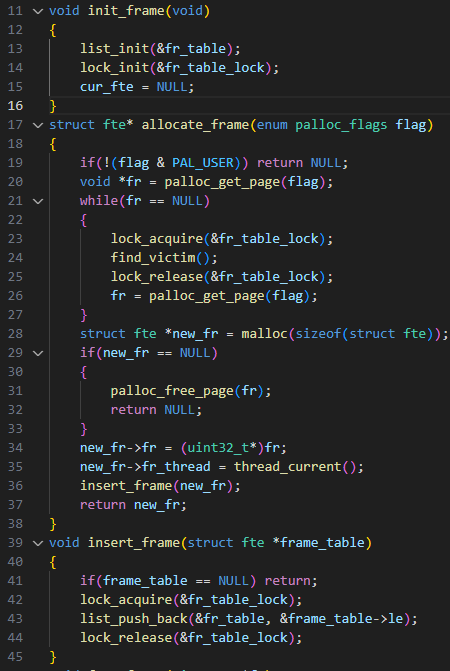
vm/swap.c 부분이다. block과 bitmap 자료구조를 활용했고 각각 초기화하는 함수, swapping을 진행하는 함수들로 구현했다.



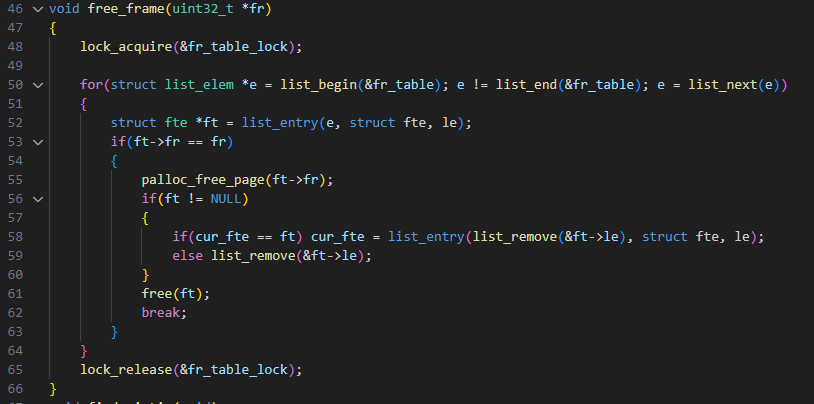
vm/page.c 부분이다. hash list로 관리하며 각각 초기화, 삽입, 현재 hash entry 접근, 삭제 등의 기능을 구현했다.



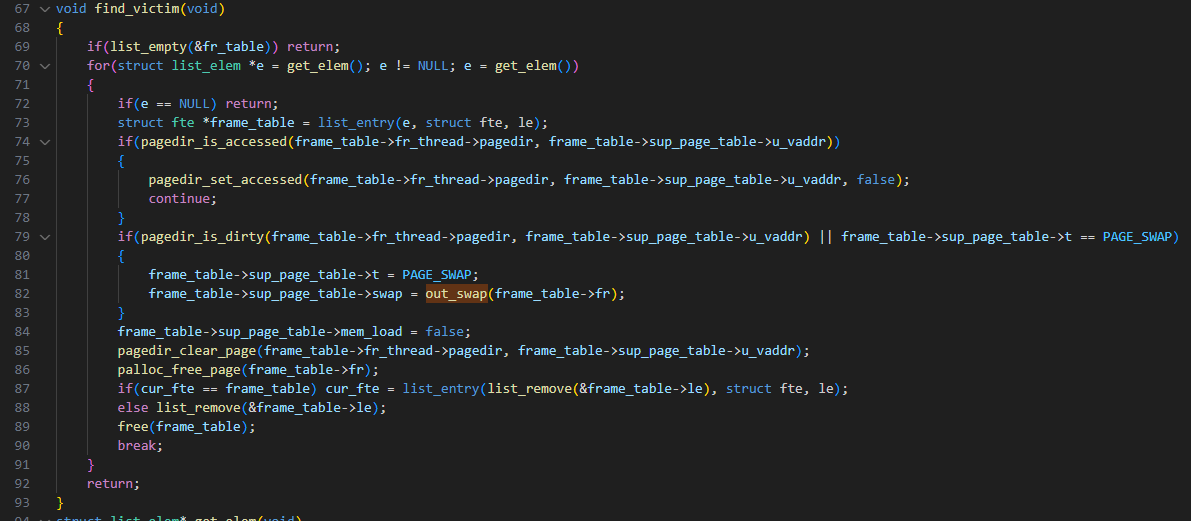
마찬가지로 vm/page.c이다. hash list를 init할 때, 사용되는 hash function과 less function, 그리고 page file을 load하는 함수가 선언된 부분이다.



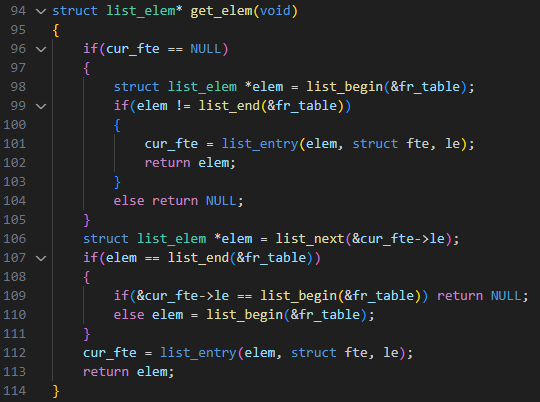
vm/frame.c이다. 각각 frame관련된 것 초기화, frame entry 얻어오기, frame을 list에 삽입하기가 구현된 부분이다.



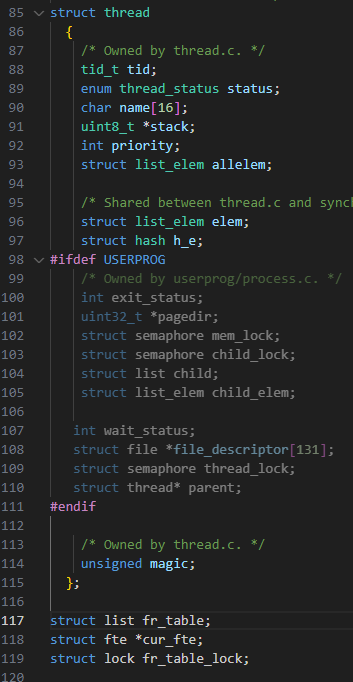
vm/frame.c의 frame list에서 frame을 찾아 free하는 함수다.



vm/frame.c의 second chance algorithm을 활용한 다음 frame victim entry를 찾아 swap하는 부분이다.



vm/frame.c의 frame list에서 현재 가리키는 frame을 list element로 꺼내 오는 부분이다.

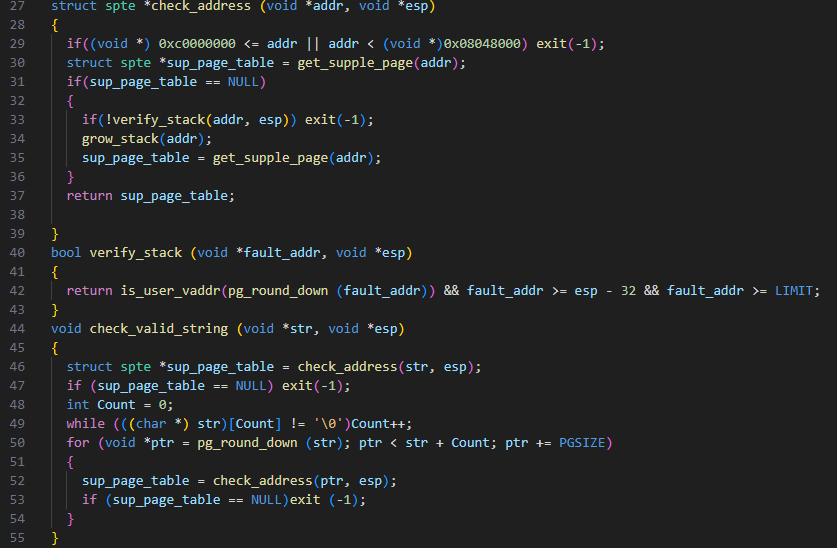


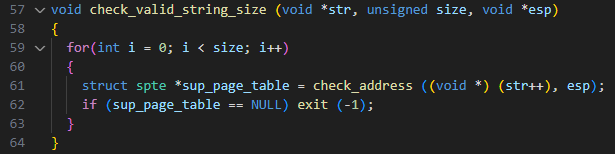
threads/thread.h의 frame table entry list, 현재 frame table entry, frame table entry list를 관리할 때, 사용되는 lock을 전역변수로 선언하고 thread 구조체 내에 hash를 선언한 곳이다.



threads/init.c에서 위에서 구현한 init\_frame, init\_swap 함수를 실행시켜줬다.

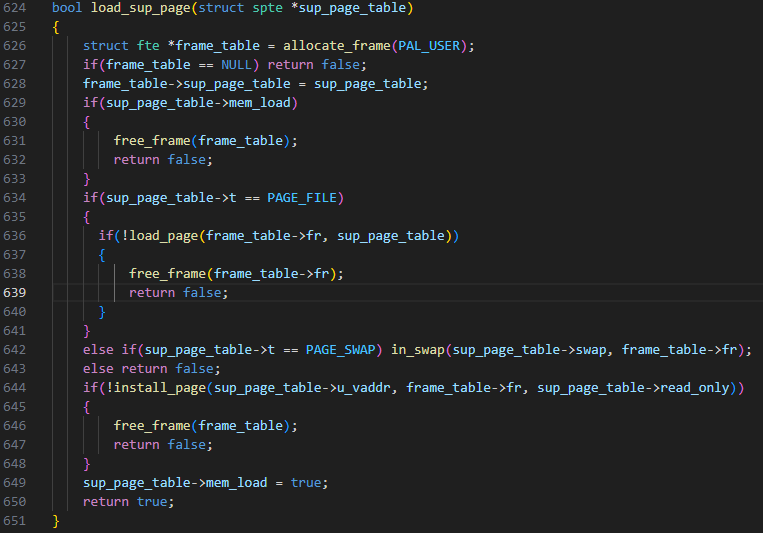




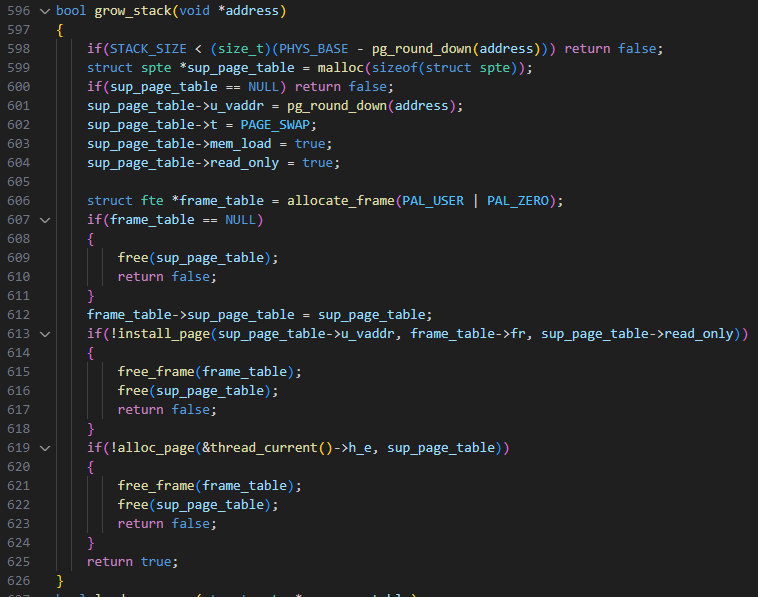


syscall.c에 주소 체크하는 부분을 구현했다. Growth가능한 최대 Size limit는 8MB로 설정했다.

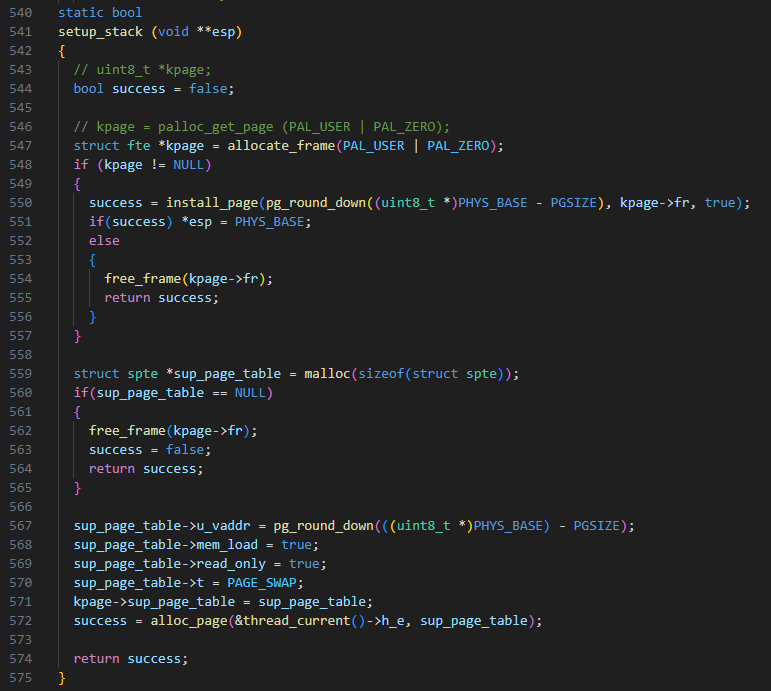
그리고 이 함수들을 이용해 syscall 호출하는 switch entry마다 주소검사가 필요한 부분은 함수 역할에 따라 검사하고 syscall을 호출했다.



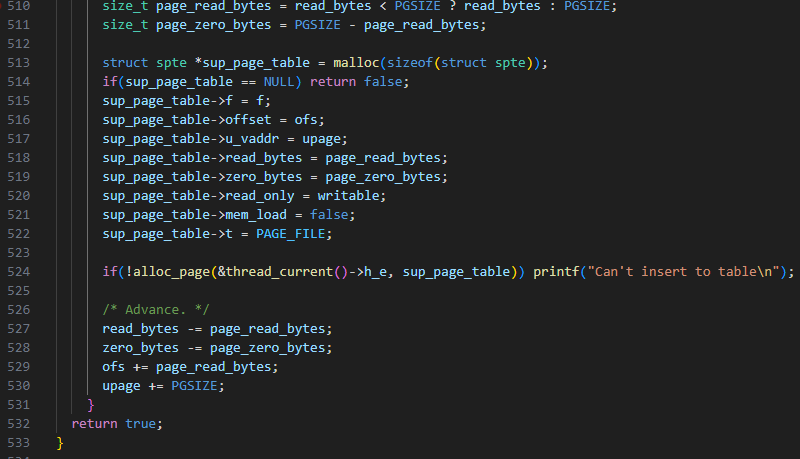
process.c의 supplementary page table entry를 load하는 부분이다. 각 타입에 따라 다르게 load시켰다.



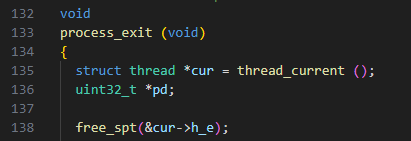
process.c의 stack을 growth하는 함수이다.

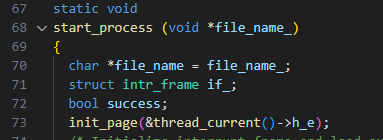


process.c의 setup stack함수를 수정했다.

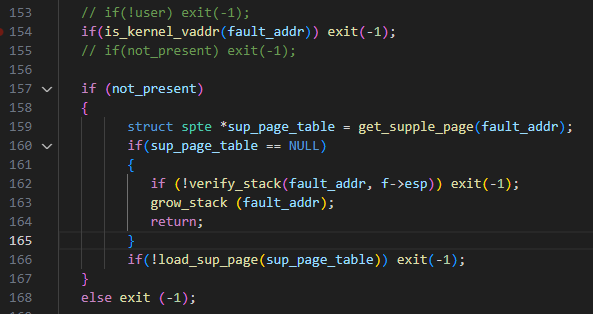


process.c의 load segment함수 또한 위와 같이 수정했다.



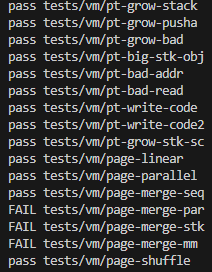


process.c의 process exit 함수에서 supplementary page table hash를 삭제하는 부분을 추가했고 start process 함수에 현재 thread의 supplementary page table hash를 초기화하는 부분을 추가했다.



마지막으로 exception.c 파일의 page fault함수 부분이다. fault된 address를 page entry로 받아오고 없다면 주소 검사 후 stack을 늘린다.

* 1. **시험 및 평가 내용**



mmap, mummap system call 구현을 하지 못했고 16개 중 총 13개를 성공하고 3개를 실 패했다.