**OS Project3 Design Report**

letsgo\_seuljongbin

20180038 박형규, 20180480 성창환

1. **Analysis of naïve Pintos.**

Naïve Pintos는 VM을 지원하지 않는다. Naïve 핀토스는 4KB크기의 page를 사용하고 있지만, 이 page들은 우리가 이번 프로젝트에서 구현하고자 하는 demand-paging기법으로 관리되고 있는 것이 아닌, page가 frame에 프로그램 시작시부터 종료시까지 1대1로 대응되어 있는 방식으로 구현되어 있다. 따라서 우리는 그냥 VM에 관한 모든 것을 새로 구현하면 된다. 구현해야 하는 기능들은 크게 demand paging 구현, stack growth 구현, memory mapping 구현으로 나누어 진다.

1. **Solutions for each** **requirements**
   1. Frame Table

page들을 page table을 이용해서 관리하듯, 실제 physical memory를 page와 같은 크기인 4096byte 단위로 나누어 frame을 만들고 이 프레임들에 접근하기 위해서 frame table을 만들어야 한다. 이는 이미 구현되어 있는 pagedir을 참조해서, 수업시간에 배운 방식대로 frame이라는 구조체를 만들고, 별도의 table을 만들어 이를 관리하는 함수들을 구현할 것이다. 기본적인 기능은 이미 pagedir에 구현되어 있으므로 이를 참조하고, 추가적으로 구현해야 하는 부분은 victim의 선정일 것이다. 이를 위해 LRU 알고리즘을 사용할 것이다. LRU알고리즘을 통해서 frame이 부족한 상황에서 적당한 victim을 선정하게 된다. 그리고 그 선정된 frame은 뒤에 구현될 swap 기능을 통해서 여유 공간을 확보하게 된다.

* 1. Lazy Loading

Lazy Loading의 기본적인 컨셉은 실제로 그 메모리를 참조하려고 할 때, load가 되지 않았다면 load 하는 것이다. 그래서 이전의 프로젝트에서 page fault가 나면 무조건 프로그램의 오류라고 간주했던 것과 달리, page fault가 났을 때, 그게 올바른 access라면, 메모리에 load되지 않은 page를 메모리로 load하는 동작을 구현하고, 그 다음 다시 그 instruciton을 실행하도록 page fault handler를 변경할 것이다. 이를 위해 올바른 access 인지 확인하는 is\_valid\_access 함수와 실제로 메모리에 loading을 수행하는 lazy\_loading함수를 만들고 이를 page\_fault 함수 안에서 적절하게 호출해서 lazy loading을 구현할 것이다.

* 1. Supplemental Page Table

기본적으로 naïve pintos에서 구현된 page table은 빈약한 정보만을 가지고 있다. 추가적으로 naïve pintos는 각 프로세스마다 별도의 page directory를 가지고 있게 되어서 VM을 구현하는데 큰 장애물이 된다. 두개 이상의 별도의 page directory에서 같은 physical memory를 참조하는 것 과 같은 오류가 발생할 수 있다. 따라서 이런 문제를 해결하기 위해서 page에 관한 여러 정보들을 포함하고 있는 새로운 table을 구현할 필요가 있다. 그리고 이 s-page들을 관리하고 접근할 때, hash table을 사용한다. 해시 테이블을 이용해서, 주어진 가상주소를 통해 그 주소에 대한 s-page구조체에 접근할 수 있게 된다. 이를 위해 이미 구현되어있는 pagedir과는 별개로, spage구조체를 만들고, sptable을 만들고 이를 관리해줄 함수들을 만들어 supplemental page table을 구현할 것이다.

* 1. Stack Growth

특정 값(ex 8MB)까지 상황에 맞게 추가적으로 스택을 연장해야 한다. 기본적인 구현 아이디어는 현재까지 할당된 페이지를 벗어나는 address로의 접근이 발생했을 때, 앞선 project 2에서 valid한 address인지 확인하는 함수를 통해 유효한 스택 접근인지 확인한다. 그리고 유효한 접근이라면, 추가적인 페이지를 할당해서 주어진 주소가 포함될 수 있도록 한다. 이러한 과정을 진행해주는 grow\_stack 함수를 만들고, 이 함수는 page\_fault함수 안에서 stack growth일 때를 확인해서 호출되도록 구현할 것이다.

* 1. File Memory Mapping

mmap 함수와 munmap함수를 구현해야 한다. 구현 방식은 파일을 메모리 주소에 매핑하는 것이다. 매핑이 되고 나면 read/write syscall이 아닌 load/store로 파일에 접근할 수 있게 된다. 우선 맵핑을 지원하기 위해서 thrad/process에 매핑된 파일들을 관리하기 위한 리스트를 만든다. 이 리스트는 매핑된 파일들을 가지고 있을 것이고, 그 정보는 unmapping시에 혹은 process termination시에 그 정보를 바탕으로 unmap해주게 될 것이다. 그리고 파일이 페이지 하나크기가 아닐 것이기 때문에, mmap함수에서는 파일의 크기에 따라 필요한 만큼 여러 개의 page들을 생성하고, 그것을 하나의 mmaped\_file이라는 구조체로 묶어서 관리할 것이다. 그리고 그 mmapped\_file이 프로세스에 리스트로 저장되어 있도록 구현할 것이다.

* 1. Swap Table

Frame을 확보하기 위해서 없어지기로(버려지기로) 결정된 페이지가를 swap disk에 저장하는 기능을 구현해야 한다. 버려지는(victim) page가 선정되는 것은 앞서 fram에서 구현한 방식을 따른다. pintos에서 제공될 swap partition을 swap slot 단위로 관리하는데, 이때, swap out시에는 슬롯들에서 적절한 위치를 찾아서 victim 페이지를 복사해준다. 그리고 swap in을 통해서 swap disk의 페이지를 다시 불러오는 기능을 구현한다.

* 1. On Process Termination

요구사항에 따라서 process termination시에 리소스들의 낭비 없이 close 할 수 있도록 구현한다. 앞서서 2.1에서 2.6까지의 기능들을 구현함에 있어서, 구조체를 동적으로 할당하게 될 것이다. 그리고 우리는 그 동적으로 할당된 메모리들을 thread\_exit 혹은 process\_exit이 호출되면 그 함수 안에서 모든 리소스를 해제해주는 기능을 구현할 것이다.