**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재

이름 / 학번 : 이재영 / 20171669

개발 기간 : 2021.11.10 ~ 2021.11.25

1. **개발 목표**

pintos에서 physical memory에서 벗어난 공간을 사용하는 경우 system call에서 thread를 exit를 하는 것을 통해 프로그램을 kill시켰는데 가상의 메모리 공간인 virtual memory를 새로 구현해서 새로운 page table을 만들고 page fault를 handle를 한다. 메모리 공간이 부족하면 disk에서 swap in, swap out을 하고 또 stack의 범위가 벗어나면 크기를 키우는 작업을 한다.

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

Virtual memory를 관리하는 entry, 즉 page table을 만들어서 이를 통해 physical memory와 virtual memory를 mapping한다. 즉, virtual memory의 크기가 physical memory보다 크면 kill하는 작업을 수행했던 것을 mapping을 통해 가능하게 한다. Handle\_mm\_fault함수를 만들어서 page fault가 발생되는 경우에 대해서 handle하도록 해결한다.

* 1. Disk Swap

Physical memory가 부족할 경우 LRU page replacement 알고리즘을 구현해서 disk에서 swap이 발생하도록 한다.

* 1. Stack Growth

Page fault가 발생할 경우 stack page를 할당 받아서 stack의 size를 8MB까지 늘리게 해준다. 즉, 이는 주소의 범위가 user영역의 address인지를 확인하고 페이지의 entry가 null인 경우 stack의 size를 키워주는 함수를 호출해 8MB까지 늘릴 수 있도록 해준다.

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

Page fault가 발생하는 이유는 page table에 없는 page를 접근하는 경우, read/write 시 발생하는 fault, 그리고 user 영역을 벗어난 곳을 접근하는 경우. page fault가 발생한다. not\_present이 false인 경우, 즉, page가 존재하는 경우는 exit을 수행하고 아니라면 page entry에서 접근이 가능한지를 확인하고, handle\_mm\_fault를 통해 swap in, swap out을 수행하고 접근이 불가능한 영역인 경우, stack growth가 가능한지 여부를 확인해 가능하다면 stack size를 증가시키면서 stack growth를 수행하고 불가능한 경우엔 exit시키는 방법으로 page fault를 handling한다.

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

Disk swap이 발생할 때, LRU algorithm을 사용한다. LRU algorithm은 Least Recently Used Algorithm을 말하며 가장 최근에 사용된 적이 없는 page를 evict시키는 방법의 알고리즘이다.

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

Stack growth는 기존의 stack의 size에서 벗어나는 page가 있는 경우 page fault가 발생하는데 이러한 상황에서 stack의 size를 늘려서 stack page를 할당 받을 수 있도록 해주는 방법이다. Stack의 top에 있는 address가 8MB의 size 범위에 있는지를 확인한다. Page를 할당하면서 주소 값을 확인해서 stack의 확장 여부를 판단할 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

2021.11.10 ~ 2021.11.19 : page table 및 page fault handle 구현

2021.11.19 ~ 2021.11.23 : swap시 사용될 LRU 알고리즘 구현

2021.11.24 ~ 2021.11.25 : stack growth 구현

2021.11.26 : 보고서 작성

* 1. **개발 방법**

**<src/threads/thread.h>**

* 추가한 자료구조

Thread 구조체에 virtual memory를 관리하는데 자료구조 hash를 사용하므로 hash 구조체 vm을 선언한다. 또한 mapping을 일부 구현해야 하므로 mmap\_list, page replacement할 때 사용될 때 lru algorithm을 사용할 것이기 때문에 lru\_list가 필요하고 이들은 모두 list 자료구조를 사용한다.

**<src/threads/init.c>**

* 수정한 함수

main : main함수에서 LRU 알고리즘을 수행하는데 사용되는 list, lock clock을 초기화하고 swap을 초기화하는 코드를 추가한다.

**<src/userprog/process.c>**

* 새로 구현한 함수들

bool handle\_mm\_fault : exception.c의 page fault함수에서 호출하는 함수로 빈 frame을 가지고 frame이 없으면 LRU 알고리즘을 수행한 후 page를 disk에 있는 page와 swap하고 page table을 수정한 후 process를 restart하는 함수이다.

bool grow\_stack : 4MB의 stack에서 stack의 size를 8MB까지 grow 시켜줄 수 있는 함수로 함수의 자세한 설명은 4-B의 제작 내용에서 자세히 설명한다.

* 수정한 함수

static bool load\_segment / Bool setup\_stack

기존의 load\_segment는 각 페이지를 물리메모리로 읽어 들인다. 즉, 모두 Data, Code segment를 읽어드린다. Setup\_stack은 stack에 physical page 모두를 할당한다. Segment를 모두 올리는 것은 물리 메모리의 낭비하는 것이기 때문에 모두 할당하는 대신 virtual page마다 일부 정보들만 관리하도록 함수를 수정한다.

static void start\_process / void process\_exit

각각 thread.h의 thread 구조체에 정의한 hash구조의 vm을 src/vm/page.c 에 정의한page\_init(&(thread\_current()->vm)), page\_destroy(&cur->vm)를 호출한다.

**<src/userprog/exception.c>**

* 수정한 함수

static void page\_fault : 기존의 page fault함수의 경우는 page fault가 발생되는 경우들은 system call에서 exit을 수행해서 process를 종료하는 방법으로 구현되어 있었는데 page fault가 발생하면 page entry에 page가 존재하는지 확인하고 virtual address에 해당되는 physical page를 할당한 후 entry의 정보를 확인해서 disk에 있는 내용을 가져오도록 구현한다.

**<src/userprog/syscall.c>**

* 추가한 함수

void check\_address : address와 esp를 받아와서 주소가 0x804800의 MIN\_ADDRESS와 0xc0000000의 MAX\_ADDRESS 사이에 있는 경우 page\_entry에 있는지 찾고 주소의 범위가 벗어나면 system call의 exit을 수행하고 아니라면 vme가 null이라면 stack을 키우는 grow\_stack을 호출해서 stack을 키우는지를 확인한다.

void check\_buf : buffer를 사용하는 read, write system call에서 호출하는 새로운 함수로 기존의 is\_user\_addr을 직접사용한 것 대신에 사용하는 함수로 size만큼 address의 범위를 확인하고 vme에서 find하면서 buffer가 유효한지 확인한다.

void check\_str : string의 유효성을 확인하는 함수로 check\_address를 호출하면서 유효성을 확인한다.

* 수정한 함수

static void syscall\_handler : syscall\_handler에서 기존의 주소의 범위는 is\_user\_addr를 사용해서 벗어나면 바로 exit하는 작업에서 새로 만든 check하는 함수들을 불러주도록 수정한다. Buffer를 사용하는 read, write system call에서는 check\_buf를 추가로 호출해주고 exec, open, create system call에서는 string이 사용되므로 check\_str을 호출해준다.

**<src/vm/page.c>**

* 추가한 함수

void insert\_inform / void update\_inform : page\_entry의 정보를 추가, update하는 함수이다.

Static bool pg\_less\_func : page\_entry의 주소를 비교하는 함수로 page를 초기화할 때 사용되는 함수이다.

void page\_init / void page\_destroy : hash구조의 page\_entry를 각각 초기화, 없애는 함수이다.

bool insert\_page\_entry / bool delete\_page\_entry : page\_entry에 각각 insert/delete 작업을 수행하는 함수이다.

bool load\_file : vme의 정보인 file, offset으로부터 페이지 한 개를 kaddr로 읽는 함수이다.

struct page\_entry\* search\_page\_entry : virtual address에 해당되는 page\_entry를 검색해서 반환하는 함수로 hash\_find함수를 이용해서 page\_entry를 검색한다.

**<src/vm/frame.c>**

* 추가한 함수

struct page\* alloc\_page : palloc\_get\_page를 통해서 page를 할당하면서 page 구조체를 할당하고 초기화하는 함수이다. lru\_add\_page로 lru list에 page 구조체를 넣어주고 page 구조체의 주소를 반환하는 함수이다.

void free\_page : physical address인 kaddr에 해당되는 page 구조체를 lru list에서 찾아서 찾는다면 lru list를 remove하고 page 구조체에 해당되는 메모리를 free해주는 함수이다.

static struct list\_elem\* get\_next\_lru\_clock : lru 알고리즘을 통해 lru list를 이동하는 작업을 할 시 사용되는 clock 알고리즘을 사용하는 함수이다.

void lru\_add\_page / void lru\_del\_page: lru\_list에 page를 추가 / 제거하는 함수이다. 제거할 시에는 clock이 page일 시에는 clock에 list\_entry를 넣고 제거한다.

void check\_free\_pages : physical page가 부족할 때 clock 알고리즘을 통해 메모리의 공간을 확보하는 함수이다.

**<src/vm/swap.c>**

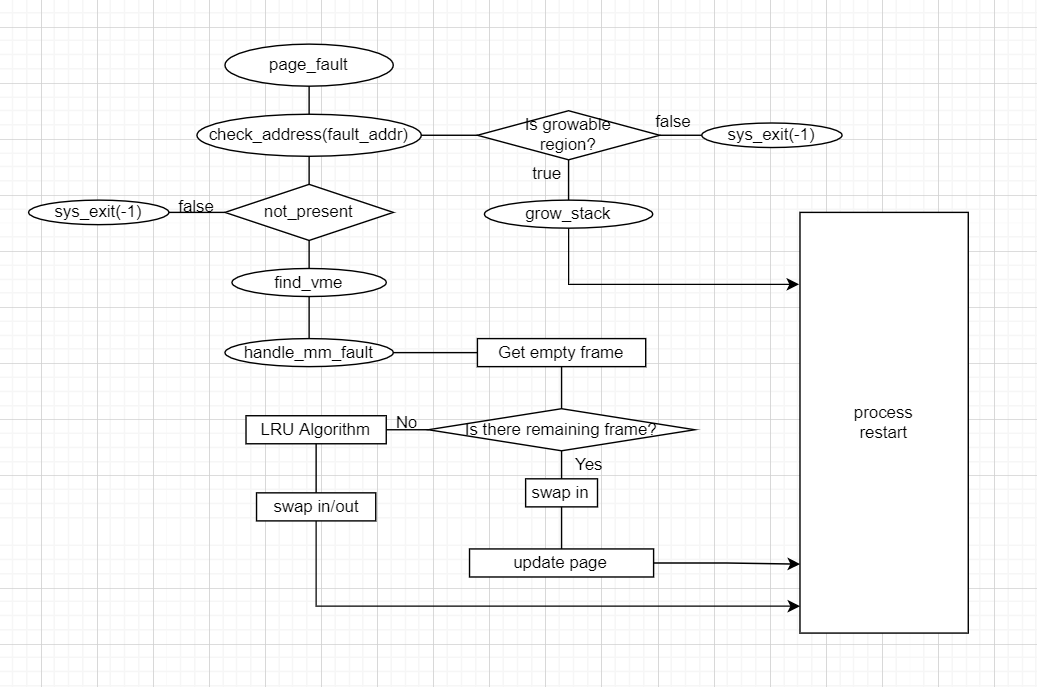
* 추가한 함수

void swap\_init : swap 영역을 초기화하는 함수로 swap\_lock을 초기화하고 bitmap 구조체인 swap\_bitmap에 bitmap\_create를 한다.

void swap\_in : disk에 page를 swap in하는 함수로 swap 영역에 저장된 데이터의 주소를 복사하는 작업을 수행한다. Block\_read 함수를 참조한다.

size\_t swap\_out : Disk에서 swap out하는 함수로 kaddr 주소가 가리키는 page를 swap 공간에 기록하고 그 영역의 번호를 반환하는 작업을 한다. Block\_write 함수를 참조한다.

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

<src/threads/init.c>

텍스트이(가) 표시된 사진

자동 생성된 설명

lru\_list, lru\_list\_lock을 초기화하고 lru\_clock을 null로 초기화한다. 그리고 swap 영역을 8\*1024의 size로 초기화한다.

**<src/userprog/process.c>**

* start\_process/process\_exit





thread구조체에서 새로 선언한 hash 구조의 vm을 각각 start\_process에서는 초기화, process\_exit에서는 destroy를 하는 함수를 호출한다.

* load\_segment

텍스트이(가) 표시된 사진

자동 생성된 설명

기존의 load\_segment는 각 page를 physical memory로 읽어 들인다. 즉, 모든 Data, Code segment를 읽어드린다. Segment를 사용하지도 않는데 모두 올리는 것은 physical memory를 낭비하는 것으로 lazily하게 함수를 수정한다. Page\_entry를 할당받아서 page entry의 정보들 file, offset, vaddr, read\_bytes, zero\_bytes, writable, is\_loaded, type, pinned를 update한 후 page\_entry에 insert하는 함수를 호출한다.

* setup\_stack

텍스트이(가) 표시된 사진

자동 생성된 설명

기존의 Setup\_stack은 stack에 physical page 모두를 할당하는데 이것 또한 physical memory를 낭비하는 것이기 때문에 모두 할당하는 대신 virtual page마다 일부 정보들만 관리하도록 physical memory와 page\_entry를 mapping해주도록 한다. 기존의 palloc\_page로 하던 것을 frame.c에서 정의한 alloc\_page로 할당받는다. 할당받은 kpage가 NULL이 아니라면 install\_page를 실행을 수행한다. 그 후, page\_entry를 새로 할당해 최소의 stack을 만들어서 page\_entry에 insert하고 insert가 성공하면 success아니라면 fail을 return한다.

**<src/userprog/exception.c>**

* page\_fault

텍스트이(가) 표시된 사진

자동 생성된 설명

기존의 page\_fault 함수에서는 invalid 영역에 접근하면 바로 process를 종료시키는 방법과는 다르게 fault\_addr의 address영역을 check하고 not\_present 변수를 확인해서 exit 수행여부를 판단한다. present하지 않으면 page\_entry에서 page를 찾아서 NULL이 아니라면 handle\_mm\_fault를 수행해서 page\_entry의 is\_loaded가 참이면 page entry의 정보인 pinned를 false로 바꾸고 load를 true로 한다.

<src/vm/page.c>

* pg\_less\_func

텍스트이(가) 표시된 사진

자동 생성된 설명

두 page\_entry의 vaddr을 비교하는 함수로 page\_init에서 hash\_init 함수를 호출할 때 사용된다.

* pg\_hash\_func / pg\_destroy\_func

텍스트이(가) 표시된 사진

자동 생성된 설명

pg\_hash\_func은 hash entry를 통해서 page\_entry의 구조체를 검색해서 page\_entry의 vaddr에 대한 hash 값을 구해서 반환하는 함수이다. pg\_destroy\_func은 hash\_entry를 통해 page\_entry를 구하고 virtual address가 physical memory에 load된 경우에는 physical address를 가지고 page를 free하고 page table을 clear한다. 그 후에 page\_entry를 free하는 함수이다.

* page\_init / page\_destroy

텍스트이(가) 표시된 사진

자동 생성된 설명

각각 page를 초기화, destroy해주는 함수로 hash\_init과 hash\_destroy함수를 참조하고 parameter로 pg\_hash\_func, pg\_destroy\_func를 사용한다.

* search\_page\_entry

텍스트이(가) 표시된 사진

자동 생성된 설명

virtual address인 vaddr에 해당되는 page\_entry를 검색해서 반환하는 함수로 page 번호는 pg\_round\_down함수를 참조하고 hash\_find함수를 이용해서 page\_entry를 검색한다.

* insert\_page\_entry / delete\_page\_entry

텍스트이(가) 표시된 사진

자동 생성된 설명

hash인 page\_entry에 element를 insert, delete해주는 함수로 hash\_insert와 hash\_delete를 참조한다. delete시에는 같이 들어온 page\_entry인 vme을 free해주는 작업 또한 수행한다.

* load\_file

텍스트이(가) 표시된 사진

자동 생성된 설명

file를 읽어서 읽은 vm\_entry의 file을 physical memory에 load하는 함수이다. File\_read\_at함수를 참조하며 read\_bytes를 읽어서 memset함수를 사용하면서 file의 load를 수행한다.

**<src/vm/frame.c>**

* alloc\_page

텍스트이(가) 표시된 사진

자동 생성된 설명

lru\_list의 삽입과 삭제를 위해 page를 allocate해주는 함수로 palloc\_get\_page함수를 참조해 physical memory를 할당한다. 할당이 실패한 경우에는 physical page가 부족한 것이므로 clock 알고리즘을 통해 메모리의 공간을 확보하고 다시 memory를 할당한다. Page를 할당해서 page를 초기화하고 lru\_add\_page를 사용해서 lru\_list에 추가한다.

* free\_page

텍스트이(가) 표시된 사진

자동 생성된 설명

page를 free해주는 함수로 lru\_list\_lock을 걸어서 다른 process에서 사용하지 못하도록 한다. Page를 돌면서 page를 찾으면 palloc\_free\_page로 찾은 page의 kaddr을 free하고 lru\_del\_page를 사용해서 lru\_list에서 삭제하고 그 lru\_page를 free해주는 함수이다.

* lru\_add\_page / lru\_del\_page

텍스트이(가) 표시된 사진

자동 생성된 설명

lru\_list에 page를 추가/제거하는 함수로 page를 추가할 때는 page가 null이 아닌 경우에 lru\_list\_lock을 걸고 page->lru를 lru\_list에 psuh\_back을 수행한다. 제거할 때는 page가 null이 아닌 경우 lru\_clock이 page인 경우는 lru\_clock에 list에 제거하면서 list\_entry를 넣고 page가 아닌 경우는 list에서 바로 제거한다.

* get\_next\_lru\_clock

텍스트이(가) 표시된 사진

자동 생성된 설명

clock 알고리즘을 사용해서 lru list를 이동하며 lru\_clock을 설정하는 작업을 수행하는 함수로 lru\_clock이 NULL인 경우에는 lru\_list가 empty list가 아니라면 list의 첫번째를 반환한다. 만약에 lru\_clock page가 lru\_list의 마지막 page인 경우에는 lru\_list가 하나의 page를 가진 경우 NULL을 반환하고 하나보다 많은 page인 경우에는 lru\_clock을 list의 시작 page로 point하게 한다.

* check\_free\_pages

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

physical page가 부족한 경우 clock 알고리즘을 이용해서 memory를 확보하는 함수이다. 이 경우에도 lru\_list\_lock으로 critical section을 보호한다. get\_next\_lru\_clock함수를 참조해서 next element들을 확인한다. page address가 accessed된 경우 accessed bit를 false으로 set하고 반복문을 진행한다. access되지 않은 경우는 evict할 page로 page의 dirty bit을 확인한다. Page\_entry가 mmap\_file이 아닌 경우는 vme의 type을 VM\_ANON으로 바꿔주고 swap\_out을 진행한다. 후에 page를 free한다.

**<src/vm/swap.c>**

* swap\_init

텍스트이(가) 표시된 사진

자동 생성된 설명

swap영역을 초기화하는 함수로 swap시 lock으로 보호할 swap\_lock을 초기화하고 bitmap구조체인 swap\_bitmap에 size만큼의 bitmap\_create를 수행한다.

* swap\_in

텍스트이(가) 표시된 사진

자동 생성된 설명

Swap out이 수행된 page를 memory에 올리는 작업을 수행하는 함수로 Lock을 사용해서 다른 process가 간섭하지 못하게 한 후 block 구조체에 block\_get\_role함수를 참조해 받아온다. Block\_read를 swap\_block과 used\_index, kaddr+BLOCK\_SECTOR\_SIZE \* i 만큼 수행하면서 swap in을 수행한다.

* swap\_out

텍스트이(가) 표시된 사진

자동 생성된 설명

Memory의 내용을 Disk의 swap 부분으로 내보내는 작업을 하는 함수이다. block 구조체에 block\_get\_role함수를 참조해 받아온다. Swap\_out함수 또한 Lock을 사용해서 다른 process가 간섭하지 못하게 한 후 Bitmap\_scan\_and\_flip 함수를 참조해서 Swap\_index에 저장한 후 block\_write함수를 사용해서 swap\_out을 수행한다.

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

텍스트이(가) 표시된 사진

자동 생성된 설명