# Pintos Project 4 - VM

**(설계 프로젝트 수행 결과)**


과목명 : [CSE4070] 운영체제

담당 교수 : 소정민

조 / 조원 : 1조 / 이규연, 허남규


개발 기간 : 2018/12/21 - 2018/12/26

**프로젝트 제목:** Pintos Project 4 - VM

**제출일:** 2018/12/26

**참여 조원:** 20141552 이규연, 20161662 허남규

## Table of Contents

## I. 개발 목표

이번 과제는 기존 Pintos 소스에서 virtual memory (VM)를 추가적으로 구현하는 것을 목표로 합니다. 이전 소스에서는 VM이 구현되어 있지 않았기 때문에, 전체 프로세스가 사용하는 메모리의 크기가 physical memory (PM)를 넘어가면 새로운 프로세스를 더 이상 로딩하지 못했습니다.

하지만 이번 과제에서는 VM을 구현하여 PM이 부족할 때, 즉 page fault가 발생했을 때, PM 상에서 페이지를 저장하는 단위인 프레임을 일시적으로 디스크 내의 swap 영역에 저장함으로써 새로운 데이터를 PM에 올릴 수 있도록 합니다. 즉, PM 용량에 구애받지 않고 프로세스를 불러올 수 있도록 하는 것입니다.

## II. 개발 범위 및 내용

아쉽게도 이번 과제에서는 시간적인 제약으로 모든 구현 사항을 완수하지 못했습니다. <개발 범위>에서 모든 구현 범위를 설명하고, <개발 내용>에서는 실제 제출한 소스에서 구현된 부분을 구체적으로 설명합니다.

### A. 개발 범위

이번 Pintos 과제에서 구현할 사항은 크게 다음과 같이 세가지로 나뉩니다.

a) Virtual 페이지의 할당

VM상에서 메모리 영역, 즉 페이지를 할당받는 과정입니다. 크게 다음과 같은 세 가지 경우가 있습니다.

- 프로세스 호출 시 프로세스 자원을 보관하기 위한 페이지 할당

- 프로세스의 함수 호출 스택이 커질 경우 추가적인 페이지 할당
- Memory-mapped file (mmap) 사용 시 필요한 페이지 할당

이는 VM 상에서 메모리를 할당하는 과정이며, 모든 경우에 실제 PM상의 프레임으로 데이터가 복사되지는 않습니다. 단, 데이터를 실제로 참조할 때 PM으로 가져올 수 있도록 데이터의 원래 위치와 현재 상태 등이 page table (PT)와 supplemental page table (SPT)에 기록됩니다.

b) (참조 시) Physical 프레임 할당 및 반환

실제 어떠한 데이터를 사용하려면 그것이 PM상에 존재해야 합니다. 즉, 해당 페이지는 PM상의 어떠한 프레임에 들어가 있어야 합니다. 만약에 데이터를 참조했는데 PM상에 존재하지 않는다면 page fault가 발생합니다. 이번 과제의 핵심 목표는 이러한 page fault를 적절하게 대처하는 것입니다. 이는 다음과 같은 과정을 통해 이루어집니다.

1. SPT를 통해 page fault 여부 확인, 프레임 위치 확인
2. Page fault 시, frame table (FT)을 참조하여 빈 프레임 가져오기
3. 사용 가능한 프레임이 없을 경우, eviction policy와 FT의 정보를 통해 기존 프레임에 있는 데이터 초기화 (데이터 종류, 쓰기 가능 여부, dirty 상태 등에 따라 swap 영역으로 복사되거나, 매핑된 파일로 복사).
4. 필요한 데이터를 프레임으로 복사 (데이터 종류에 따라 디스크에서 가져오거나 다른 메모리 영역에서 가져오거나 0으로 초기화).

이때 PT, SPT, FT, swap table (ST)이 모두 관여합니다.

c) 메모리 해제

프로세스가 종료되거나 메모리 매핑을 해제 (munmap)할 경우, 프레임과 페이지 모두 해제되어야 합니다. 단, 이때 쓰기 가능 여부 및 dirty 상태에 따라 PM 내에서 쓰여진 내용을 다시 디스크로 복사해야 하는 경우도 있습니다.

### B. 개발 내용

a) Page Fault 시 빈 프레임 할당

이번 과제에서는 전체 개발 범위의 일부만 구현할 수 있었습니다. 구체적으로, page fault가 일어날 때, FT를 참조하여 필요한 프레임을 할당받은 후, 접근하고자 하는 페이지를 해당 프레임에 로딩하도록 했습니다. 이번 개발 단위에서는 이용 가능한 프레임이 없을 경우 오류를 발생시키도록 했습니다. 이와 관련된 구현은 다음 모듈에서 이루어졌습니다.

src/userprog/exeption.c: page_fault()

b) 프로세스 종료 시 파일 닫기 및 리소스 반환

프로세스가 종료될 때, 프로세스와 연관된 모든 파일을 close 처리하고, 자식 프로세스도 정상적으로 종료될 수 있도록 관련 semaphore를 해제하도록 했습니다. 이와 관련된 구현은 다음 모듈에서 이루어졌습니다.

src/userprog/exeption.c: process_exit()

## III. 추진 일정 및 개발 방법

### A. 추진 일정

- 2018.12.21 ~ 2018.12.22 Pintos Manual 학습 및 구현 사항 확인
- 2018.12.22 ~ 2018.12.23 소스 코드 수정
- 2018.12.23 ~ 2018.12.24 디버깅
- 2018.12.24 ~ 2018.12.26 보고서 작성

### B. 개발 방법

a) 알림 script 및 tmux 활용

채점 도중에 소스를 디버깅할 수 있도록 위의 채점 스크립트를 background 에서 실행하기 위해 tmux 커맨드를 활용했습니다. 또한, 채점이 완료되기를 “busy-wait” 하지 않고, 완료되었다는 알림을 받을 수 있도록 slack web

hook을 이용한 알림 스크립트를 작성하고 PATH에 추가하여 알림 명령어로 활용했습니다. 이 또한 scripts 디렉토리 안에 구현 및 설명되어 있습니다.

b) 프로세스와 연관된 파일 close (list)

프로세스 해제 시, 그와 연관된 모든 파일에 대해 file_close 처리하기 위해 Pintos에서 제공하는 기본 리스트 (linked-list) 관련 함수를 이용하여 구현했습니다.

c) 상세 구현

상세 구현 사항은 C 기본 문법을 사용하여 logic에 맞는 코드를 작성하고, 상술한 방법과 같이 채점을 하고, 디버깅을 하는 방식으로 구현을 해나갔습니다.

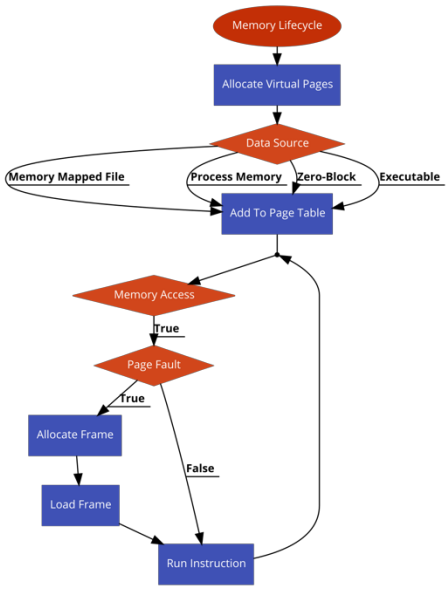
### C. 연구원 역할 분담

- 이규연: Page Fault 처리 구현 및 디버깅
- 허남규: 디버깅 지원 및 보고서 작성

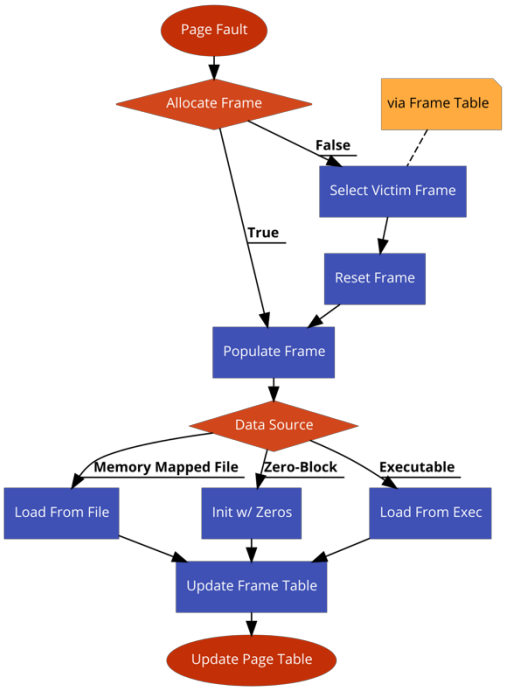
## IV. 연구 결과

### A. 합성 내용 (플로우차트)

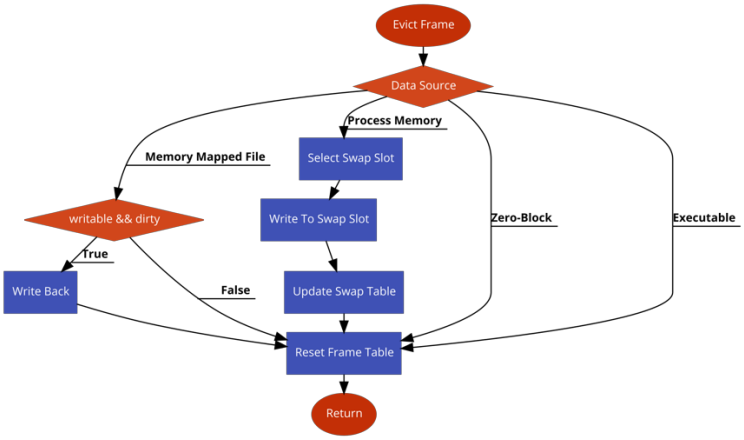
a) 전반적인 메모리 Life-Cycle



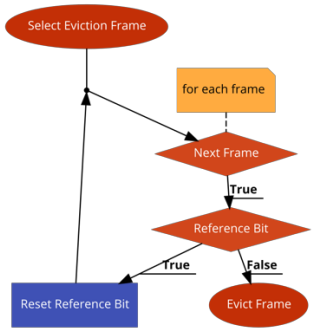
b) Page Fault 처리 과정




c) Frame Eviction 처리 과정




d) Second Chance 알고리즘



## B. 제작 내용

### a) src/userprog/exception.c

```
@@ -9,6 +9,9 @@
 #include "syscall.h"
 #include "lib/user/syscall.h"

+#include "pagedir.h"
+#include "threads/palloc.h"
+
 /* Number of page faults processed. */
 static long long page_fault_cnt;
```

시스템에서 페이지를 할당받고 페이지 테이블을 접근하기 위해 관련 모듈 import

```
@@ -154,18 +157,37 @@ page_fault (struct intr_frame *f)
   write = (f->error_code & PF_W) != 0;
   user = (f->error_code & PF_U) != 0;

-  /* If it is not a valid address, exit. */
-  if (!user || is_kernel_vaddr(fault_addr) || not_present)
-    exit(-1);
+#ifdef VM
+  uint8_t *new_page;
+  static void *new_size = (uint8_t *)PHYS_BASE - PGSIZE*2;
+
+  /* Stack growth. */
+  if (not_present && write && user && is_user_vaddr(fault_addr))
+  {
+      /* Non-growable region. */
+      if (pg_round_up (fault_addr) <= PHYS_BASE - (1<<23))
+          exit (-1);
+
+      else
+      {
+          new_page = palloc_get_page (PAL_USER + PAL_ZERO);
+
+          /* Page allocation failed. */
+          if (!new_page)
+              exit (-1);
+
+          else
+              pagedir_set_page (thread_current ()->pagedir, new_size, new_page, true);
+      }
+      new_size -= PGSIZE;
+  }
+
+  else
+      exit (-1);

-  /* To implement virtual memory, delete the rest of the function
-     body, and replace it with code that brings in the page to
-     which fault_addr refers. */
-  printf ("Page fault at %p: %s error %s page in %s context.\n",
-          fault_addr,
-          not_present ? "not present" : "rights violation",
-          write ? "writing" : "reading",
-          user ? "user" : "kernel");
+#else
+  /* To handle page faults in project USERPROG. */
+  exit(-1);

-  kill (f);
+#endif
 }
```

페이지-fault 발생 시 해당 메모리 접근이 유효한지 확인할 후, 프레임을 추가적으로 할당 받아 페이지 table로 하여금 그것을 가리키도록 합니다. 단, 메모리를 더 이상 할당할 수 없는 경우, 오류를 발생시킵니다.

```
@@ -214,12 +214,20 @@ process_exit (void)
   /* Close all open files */
   while (!list_empty (&cur->file_list))
   {
-    e = list_pop_front (&cur->file_list);
-    fw = list_entry (e, struct file_wrapper, file_elem);
+      e = list_pop_front (&cur->file_list);
+      fw = list_entry (e, struct file_wrapper, file_elem);

-    /* Why not close(fw->fd)? */
-    file_close (fw->f);
-    free(fw);
+      /* Why not close(fw->fd)? */
+      file_close (fw->f);
+      free(fw);
+  }
+
+  /* Let any waiting children exit. */
+  e = list_head (&cur->children);
+  while ((e = list_next (e)) != list_end (&cur->children))
+  {
+      child = list_entry (e, struct thread, siblings);
+      sema_up (&child->sema_exit);
   }

   /* In this part of code, parent is still suspended, which cannot
```

프로세스가 종료될 경우, 자식 프로세스로 하여금 종료할 수 있도록 semaphore를 풀어줍니다.

b) 미구현

이외의 사항은 시간 제약으로 인하여 구현을 완료하지 못했습니다.

### C. 시험 및 평가 내용

- 다음은 make check 수행 결과입니다: 구현을 완료한 부분에 한하여 모든 테스트를 성공적으로 수행한 모습을 확인할 수 있습니다.

```
pass tests/vm/pt-grow-stack
pass tests/vm/pt-grow-pusha
pass tests/vm/pt-grow-bad
pass tests/vm/pt-big-stk-obj
pass tests/vm/pt-bad-addr
pass tests/vm/pt-bad-read
pass tests/vm/pt-write-code
pass tests/vm/pt-write-code2
pass tests/vm/pt-grow-stk-sc
FAIL tests/vm/page-linear
pass tests/vm/page-parallel
FAIL tests/vm/page-merge-seq
FAIL tests/vm/page-merge-par
FAIL tests/vm/page-merge-stk
FAIL tests/vm/page-merge-mm
pass tests/vm/page-shuffle
```

## V. 기타

### A. 연구 조원 기여도

- 이규연 50%, 허남규 50%

## B. 소감

- 운영체제 본 강의에서는 swapping 시 성능 저하, 페이지 replacement policy (eviction policy) 알고리즘을 집중적으로 다룬 데에 반해, 이번 과제에서는 동작하는 시스템을 구현하기 위해 다음과 같은 자료 구조를 어떻게 구성해서 어떻게 사용할 것인가를 확실하게 파악할 수 있었습니다.
    - ■ Supplemental 페이지 Table
    - ■ 프레임 Table
    - ■ Swap Table
    - ■ Memory Map Table
- 이번 과제를 통해 memory-mapped file에 대해 더 자세히 알아볼 수 있었습니다. 단순히 “파일의 일부를 메모리 영역 일부에 대치시킨 것”이라는 제한된 이해에서 벗어나, 실제 구현 과정을 상세히 살펴보면서 마치 프로세스가 올린 VP와 같이 다루어진다는 사실을 직관적으로 이해할 수 있게 되었습니다.
- Pintos에서 제공하는 기본 자료구조 (Bitmap)을 사용하고 그 설계를 살펴보면서, 소스 파일과 함수를 사용해서 어떠한 작업 단위를 C언어 상에서 추상화시키는 설계에 관한 노하우를 얻을 수 있었습니다.
- Second-chance 알고리즘을 직접 구현하고 시스템 상에서 적용을 해보면서 그 작동 방식에 관해 좀더 직관적이고 확실하게 이해하게 되었습니다.