```
+--------------------------+
|                  CS 330                  |
| PROJECT 3: VIRTUAL MEMORY      |
|             DESIGN DOCUMENT                 |
+-------------------------+
```

Use 1 token for 3-2

---- GROUP ----

>> Fill in the names and email addresses of your group members.

Sujin Jang <jsujin9603@kaist.ac.kr>
Haney Kang <haney1357@kaist.ac.kr>

---- PRELIMINARIES ----

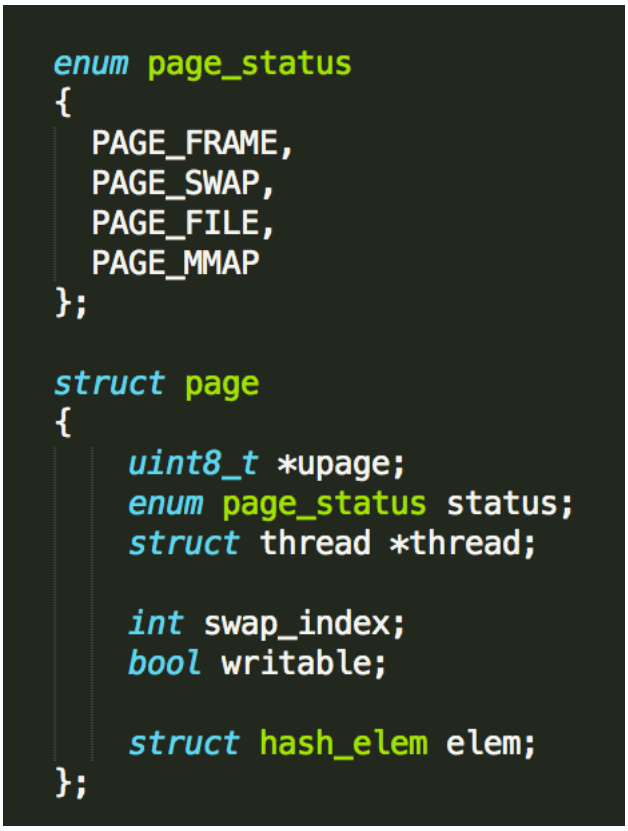
>> If you have any preliminary comments on your submission, notes for the
>> TAs, or extra credit, please give them here.

>> Please cite any offline or online sources you consulted while
>> preparing your submission, other than the Pintos documentation, course
>> text, lecture notes, and course staff.

PAGE TABLE MANAGEMENT
=====================

---- DATA STRUCTURES ----

>> A1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.



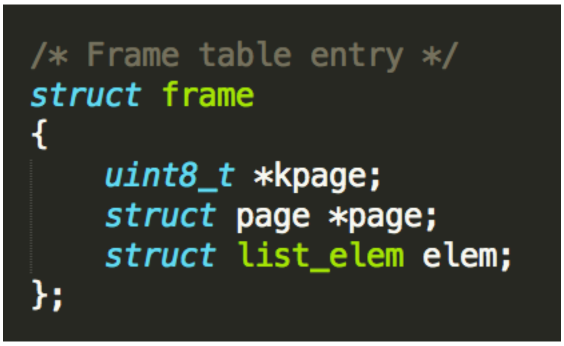
```c
enum page_status
{
  PAGE_FRAME,
  PAGE_SWAP,
  PAGE_FILE,
  PAGE_MMAP
};

struct page
{
    uint8_t *upage;
    enum page_status status;
    struct thread *thread;

    int swap_index;
    bool writable;

    struct hash_elem elem;
};
```

supplemental page table entry 로 upage (virtual memory address), status, thread (page 를 가진 user process thread), swap_index (swap 시 swap table 내의 index), writable 값을 가진다.



```c
/* Frame table entry */
struct frame
{
    uint8_t *kpage;
    struct page *page;
    struct list_elem elem;
};
```

frame table entry 로 kpage (physical memory address), 대응되는 page struct 를 가진다.

---- ALGORITHMS ----

>> A2: In a few paragraphs, describe your code for locating the frame,
>> if any, that contains the data of a given page.

Frame 이 allocation 되면 frame struct 를 생성한다. 그리고 대응되는 page struct 를 frame 의 page 에 저장하고, page 의 status 를 PAGE_FRAME 으로 설정한다. 이렇게 하면 frame 에서 page 에 접근할 수 있다. Frame 에서 physical memory address 를 얻기 위해서는 kpage 에 access 하면 되고, virtual memory address 를 얻기 위해서는 page 내의 upage 에 접근하면 된다. Page 에서 frame 에 mapping 이 되었는지는 status 값의 확인을 통해 알 수 있다.

>> A3: How does your code coordinate accessed and dirty bits between
>> kernel and user virtual addresses that alias a single frame, or
>> alternatively how do you avoid the issue?

한 page 와 한 frame 만 mapping 이 가능하다.

---- SYNCHRONIZATION ----

>> A4: When two user processes both need a new frame at the same time,
>> how are races avoided?

한 개의 page 만 pagedir 에서 mapping 이 되어있다. Mapping 되어있지 않은 page 는 frame 에 접근이 불가능하므로, race condition 을 고려하지 않아도 된다.

---- RATIONALE ----

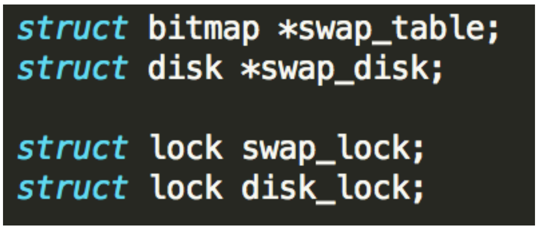
>> A5: Why did you choose the data structure(s) that you did for
>> representing virtual-to-physical mappings?

Page table: hash table 이 가장 access 가 빠르다.
Frame table: list 의 기능으로 충분하다.
Swap table: 각 index 에 대해 allocating 되었는지의 정보만 필요하므로 bitmap 을 사용한다.

PAGING TO AND FROM DISK
========================

---- DATA STRUCTURES ----

>> B1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.



```
struct bitmap *swap_table;
struct disk *swap_disk;

struct lock swap_lock;
struct lock disk_lock;
```

Swap table 은 swap 여부를 나타낸다 (0 = not swapped, 1 = swapped)

---- ALGORITHMS ----

>> B2: When a frame is required but none is free, some frame must be
>> evicted. Describe your code for choosing a frame to evict.

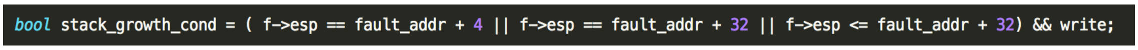
FIFO. Frame 은 frame list 에 먼저 allocate 된 순서대로 push 된다. Frame 이 가득 차면, 가장 앞에 있는 (가장 먼저 allocate 된) frame 부터 evict 한다.

>> B3: When a process P obtains a frame that was previously used by a

>> process Q, how do you adjust the page table (and any other data
>> structures) to reflect the frame Q no longer has?

Q 의 pagedir 에서 해당 page 의 mapping 정보를 제거한 뒤 swap out 시킨다. 이후에 해당 page 에 access 하면 page fault 내에서 다시 swap in 한다.

>> B4: Explain your heuristic for deciding whether a page fault for an
>> invalid virtual address should cause the stack to be extended into
>> the page that faulted.



```
bool stack_growth_cond = ( f->esp == fault_addr + 4 || f->esp == fault_addr + 32 || f->esp <= fault_addr + 32) && write;
```

push instruction 으로 인해 esp 가 fault address 보다 4 혹은 32 만큼 작을 수 있다. 따라서 esp 는 fault address 보다 최대 32 만큼 내려갈 수 있다. 또한 stack growth 를 하기 위해서는 write 접근이어야 한다. 이 조건을 만족할 경우 stack growth 한다.

---- SYNCHRONIZATION ----

>> B5: Explain the basics of your VM synchronization design. In
>> particular, explain how it prevents deadlock. (Refer to the
>> textbook for an explanation of the necessary conditions for
>> deadlock.)

swap in, out 을 할 때 swap table 과 disk 의 데이터 일관성을 지키기 위해, swap table 에 접근하는 경우 swap_lock 을 이용하고, disk 에 접근하는 경우 disk_lock 을 이용했다.

>> B6: A page fault in process P can cause another process Q's frame
>> to be evicted. How do you ensure that Q cannot access or modify
>> the page during the eviction process? How do you avoid a race

>> between P evicting Q's frame and Q faulting the page back in?

evict 과정은 frame allocate 함수 내에서 이루어진다. 따라서 allocate 함수에 동시에 여러 process 가 access 하지 못하도록, 함수를 시작할 때와 끝날 때 evict_lock 을 걸어주었다.

>> B7: Suppose a page fault in process P causes a page to be read from
>> the file system or swap. How do you ensure that a second process Q
>> cannot interfere by e.g. attempting to evict the frame while it is
>> still being read in?

lock 을 통해 방지한다.

>> B8: Explain how you handle access to paged-out pages that occur
>> during system calls. Do you use page faults to bring in pages (as
>> in user programs), or do you have a mechanism for "locking" frames
>> into physical memory, or do you use some other design? How do you
>> gracefully handle attempted accesses to invalid virtual addresses?

System call 에 들어오는 pointer 들은 모두 handler 내에서 valid 한지 확인한다. (그 매커니즘은 page fault handler 와 동일하다.)

---- RATIONALE ----

>> B9: A single lock for the whole VM system would make
>> synchronization easy, but limit parallelism. On the other hand,
>> using many locks complicates synchronization and raises the
>> possibility for deadlock but allows for high parallelism. Explain
>> where your design falls along this continuum and why you chose to
>> design it this way.

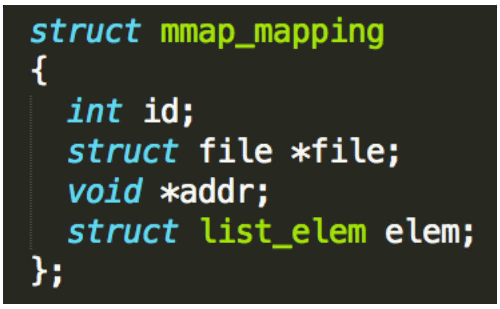
후자이다. 왜냐하면 parallelism 을 구현하지 못하는 system 은 많은 process 를 실행하면 실행할수록 불완전하며 한계점이 존재하기 때문이다.

MEMORY MAPPED FILES

====================

---- DATA STRUCTURES ----

>> C1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration.   Identify the purpose of each in 25 words or less.



```
struct mmap_mapping
{
  int id;
  struct file *file;
  void *addr;
  struct list_elem elem;
};
```

mmap table 의 entry 로 mapid, file, address 값을 가진다.

---- ALGORITHMS ----

>> C2: Describe how memory mapped files integrate into your virtual
>> memory subsystem.   Explain how the page fault and eviction
>> processes differ between swap pages and other pages.

Page 와 frame 을 allocate 하고 해당되는 file 의 data 를 저장한다. Eviction 은 다른 page 와 동일하다.

>> C3: Explain how you determine whether a new file mapping overlaps
>> any existing segment.

Frame list 에 segment 가 존재한다면 evict 하고 다시 insert 한다.

---- RATIONALE ----

>> C4: Mappings created with "mmap" have similar semantics to those of
>> data demand-paged from executables, except that "mmap" mappings are
>> written back to their original files, not to swap. This implies
>> that much of their implementation can be shared. Explain why your
>> implementation either does or does not share much of the code for
>> the two situations.

Munmap 을 호출하거나 process exit 할 때만 write back 한다.

SURVEY QUESTIONS
================

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

>> In your opinion, was this assignment, or any one of the three problems
>> in it, too easy or too hard? Did it take too long or too little time?

>> Did you find that working on a particular part of the assignment gave
>> you greater insight into some aspect of OS design?

>> Is there some particular fact or hint we should give students in
>> future quarters to help them solve the problems? Conversely, did you

>> find any of our guidance to be misleading?

>> Do you have any suggestions for the TAs to more effectively assist
>> students, either for future quarters or the remaining projects?

>> Any other comments?