```
           +---------------------------+
           | CS 140                    |
           | PROJECT 3: VIRTUAL MEMORY   |
           |  DESIGN DOCUMENT          |
           +---------------------------+
```

---- GROUP ----

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

Yuxin Miao <miaoyx@shanghaitech.edu.cn>
Fan Zhang <zhangfan5@shangahaitech.edu.cn>

---- 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.

------------------ page.h ------------------

```
struct page
  {
    void *addr;                       /* user virtual address. */
    bool writable;                    /* read only or read and write */
    bool swapable;                    /* true to write to swap. */
    struct thread *onwer;             /* thread using that page */
    struct frame *frame;              /* the frame related to this page */
    struct hash_elem pte;             /* to store in table */
    struct file *file;                /* file in page */
    block_sector_t swap_sector;       /* sector of swap area, -1 if no swap area
*/
    off_t offset;                     /* address offset of file in page*/
    off_t bytes;                      /* read (write) bytes */
  };
```

------------------ page.c ------------------

```
struct thread (Added)
  {
    struct hash *page_table;      /* supplemental page table */
  }
```

---- ALGORITHMS ----

>> A2: In a few paragraphs, describe your code for accessing the data
>> stored in the SPT about a given page.

Given a page, assuming it's the virtual address we want to access, we will first
use
this given address to go throught the SPT to find the page it belongs to. Given
this
found page, it should contain the struct we design.Thus, we can directly access
the

physical memory, aka frame by the pointer 'frame'.

If the frame is allocated previouly for this page, then the pointer shall not be null and
can be accessed directly. Otherwise, this pointer should be null and will enter page_fault
because this is an access to a unknown address.

In page_fault, we will go through a set of conditions to deside if this page or address is
suitable for allocating a physical frame to it, returning the frame allocated if true.

>> 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?

We think this problem is should be accessed by calling the pagedir function 'pagedir_set()',
which build a map between the user page and the kernel page, thus pagedir_set_dirty and
pagedir_set_accessed shalled be operated on both pages.

At the same time, we only accessing user data from user address, thus has nothing to do with
the kernrl address it mapps to.

---- SYNCHRONIZATION ----

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

We designed a lock for frame table, this lock should be accessed when ever our frame_allocate
function is called, and thus only one process can access the frame table management system once.

---- RATIONALE ----

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

Previously, it uses the pagedir to establish and maintain the virtual-to-physical mappings.
We use a frame pointer to directly point to the physical frame. This pointer makes things easier
and faster when we want to access a virtual address.

With this pointer we can directly access the frame when a locate the page the address belongs to.
Without this pointer we should use pagedir_get_page (), which firstly find the mapping calculation
relationship between virtual and physical address then do the translation, which is really slow.

The pagedir can be used partially in our lazily load, swap and mmap purpose, since we just do the
mapping one and a time instead of mapping them all at the first. However, this will not achieve the
needs like evict or re-allocate the virtual-physical mapping. Thus we implemented the SPT.

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.

```
-------------- frame.h --------------------
struct frame
{
    struct lock frame_lock;     /* The lock ensures critical section. */
    void* ker_base;             /* Kernel virtual page. */
    struct page *page;          /* The page this frame assigned to. */
    struct list_elem fte        /* Frame table entry. */
};

-------------- frame.c --------------------
  /* Use list to implement frame table, nodes as entries. */
  static struct list frame_table;

  /* Lock for accessing the frame table */
  static struct lock frame_table_lock;

  /* The loop indicator for eviction clock. */
  static size_t clock_loop;

-------------- swap.c --------------------
  /* The swap block on disk. */
  struct  block *swap;

  /* The lock for swap. */
  struct lock swap_lock;

  /* The bitmap used to manage the swap. */
  struct bitmap *swap_bitmap;
```

---- 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.

Firstly, we use a list to maintain the frame table, thus the entry is the node in the list. Further, we allocate all the physical frame available into our frame table at the frame_init() function. Whichever frame node in the list has a virtual-physical mapping, we set the page pointer in the struct frame into the page in the mapping, and null otherwise.

When we want to find a frame to evict, we use the second chance/clock page replacement policy.
Totoally we travel the list twice due to the policy. During the process, as soon as we meet a frame with a null pointer of page, we return this frame. Otherwise we check if this frame is accessed recently, if so, we set the page mapped with the frame to not recently accessed and go on to next frame.
If no one condition above mathes, it indicates a frame which has a mapped and page and is not recently accessed (whether it's set in the previous loop or not),
and this is the frame mapped page we want to evict.

>> 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?

When Q no longer use a frame, we will automatically 'free this frame',
which set the page pointer in the frame pointer to null, indicating no one
holds this frame any longer.

>> 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.

We first verify if the address is a virtual address.
We then judge if the address which need to be extened to is in a 32 bytes
bytes range from current uesp which is self-defined property.
We also judge if the address if the address is above the lowest bottom of
user stack.

---- 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.)
We have several locks and no semas or conditional variables in our VM system.
we have a lock for frame table, a lock for swap_bitmap and locks for every frame
struct in our frame table.

First of all, every time a process wants to access a frame, it needs fisrt
acquire
locks of frame table and the frame it wants to modify.
We don't actually allow a looply acquire of lock, that is acquiring a lock
during the critical section except for the frame table lock and the frame lock
of
each frame. However, this process is not a typical looply lock acquire because
one
process can only acquire the lock for the frame after it acquires the lock for
the
frame table which only has one in the whole system.
Thus we think we are capable of avoiding dead lock.

>> 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?

When we call frame eviction we will first try to lock the frame if success
then we do eviction and skip the frame if faided. And we release the
lock after eviction so there will be no race come when evicting.

>> 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?

We have a lock for each frame and each function that is tring to modify
the frame will acquire the frame lock first and release the lock before
return. And each time we call frame allocatation and eviction will first
try to acquire the lock if fails then skip the frame.

>> 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?

When process accessed to a paged-out page it will come into page fault

and we will verify whether the fault address is valid or not, and when it is accessing to a paged-out page we will call page in function to bring in the page from proper scope. In page fault we check fault address and the running status returned by allocation or page-in or set function. If any false get then terminate the process immediately.

---- 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.

We have use multiple locks in our VM system. There are three locks in frame table, single frame and the swap area. We think that giving each frame a lock can avoid processes modify the frame simultaneously which may cause bugs.Locks for frame table and swap bit map is also to avoid simultaneously scan and modify. It is quite hard and tricky to maintain single lock in whole VM system. We can't call function that modify vitural page and function that modify frame seperately.

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.

In "syscall.h" newly declaration:

```
/*struct that stores map information and file pointer*/
struct map
{
  int mapid;              /*map id*/
  size_t page_count;      /*amount of pages*/
  uint8_t *index;         /*begin position of map memory*/
  struct file *file;      /*file address*/
  struct list_elem elem; /*convenient for storing*/
};
```

In "thread.h" additional declaration:

```
struct list map_list;    /*map list*/
```

---- 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.

Each process has its own memory map list which contains the information of the memory mapped files and virtul memory address as well as its own mapid. Each page has a swapabel and swapable varibale to identify swap pages and other pages.When a process start, we will initialize a map list and when calling mmap it will go into page fault and it will allocate a virtural page for buffer if there is free frame then allocate it to the virtural page. Call eviction if not. After frame allocation page fault choose to fill the page with file, swap data or zeros.

>> C3: Explain how you determine whether a new file mapping overlaps
>> another segment, either at the time the mapping is created or later.

If the vitural page has been taken aka it is in my supplemental page table
then we return NULL which means allocation failed and clear the information
stored in map and free the map created, return -1 to indicate an error occured.

---- 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.

Since our mmap calls file length and allocate "demand" page from memory
for file to map. It is kind of like demand-paged from executables.However
data frome executables stored in stack and flush after process exit there
is no need to write them into origianl files just allocate a swap area as
"additonal memory" in disk to temporarily store the data but in mmap you
have to write back to disk after modification.

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?