```
            +---------------------------+
            |          CS5600           |
            | PROJECT 3: VIRTUAL MEMORY |
            |      DESIGN DOCUMENT      |
            +---------------------------+
```

---- GROUP ----

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

Wanting Jiang            <jiang.wa@husky.neu.edu>
<wanting>
Heng Xu                     <xu.he@husky.neu.edu>
<hengx>
Douglas Johnston     <johnston.5douglas@gmail.com>
<douglasj>

---- 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.
Refer to this online sources:
https://github.com/ryantimwilson/Pintos-Project-3

## 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.

In “page.h”:
```
// spt_entry is the supplement page which records the information
// of the user page
struct spt_entry {
  void *upage;
  uint8_t *frame;       // frame entry
  struct file *file;    // file
  off_t ofs;            // file offset
  bool writable;        // whether file can be writable or not
  uint32_t read_bytes;  // read bytes
  uint32_t zero_bytes;  // zero bytes
  bool loaded;          // whether page is loaded or not
  uint32_t swap_sector; // if page is swapped, point out the
swap sector
  bool swap;            // whether page is swapped or not
  bool mmap;            // whether it is a memory mapped file or
not
  int mapid;            // if it is a memory mapped file, point
out the map id
  bool pinned;          // prevent other processes from
accessing it when page is being used

  struct hash_elem elem;
};
```

In “frame.h”:

```
struct list frame_table;
struct lock frame_lock;

struct frame_entry {
  uint8_t *frame;               // frame address
  struct spt_entry *spte;       // page entry
  struct thread *owner;         // thread id
  struct list_elem elem;
};
```

In “thread.h”, we added the following fields:

```
struct hash spt;  /* Supplemental page table. */
struct list mmap_list;   //a list of mmap_entry
int mapid;  // map id, used when creates a new mmap_entry
```

---- ALGORITHMS ----

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

There are 4 steps to locate the frame:
1. Round down the user virtual address to the corresponding memory page address.
2. Then call palloc_get_frame() to allocate a frame for this supplemental page. If no free frame is available right now, c all frame_evict() to free a frame.
3. Call install_page() to map user virtual address to physical frame address.
4. Read data from the file if user virtual address is from a file or mmap, or from the swap partition if user virtual address is from swap.

>> 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 start by checking the pointer to make sure it is valid in syscall.c and exception.c. We record the user virtual address in “frame” and “upage” to find the supplemental page table, and kernel virtual address is never used.

---- SYNCHRONIZATION ----

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

We have a new struct frame_lock and added it to three functions:
1. add_to_frame_table();
2. free_frame();
3. frame_evict();

The lock ensures that at any given time, only one process can acquire the lock and get page index. So when two processes successfully obtain pages, they will add the new frame to the frame table sequentially since we add lock in add_to_frame_table().
Same condition happens to free_frame() and frame_evict(). When two
or more processes get pages, every action on frame will

be performed sequentially, thus races are avoided.

---- RATIONALE ----

>> A5: Why did you choose the data structure(s) that you did for
>> representing virtual-to-physical mappings?
We added “upage” and “frame” in spte_entry and frame_entry correspondingly.
As described in A3, only user virtual address is used and kernel virtual address
is never used, thus alias problem can be avoided.

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.

In “swap.h”:
```
struct block *swap_block;   // a device where swapped pages are located
struct bitmap *swap_bitmap; // indicate whether the sectors on swap block is occupied or free
struct lock swap_lock;
```

---- 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.
We used a clock algorithm to choose which frame to evict.
Going through all the frames in frame table:
1. If the page is not pinned, go to Step 2, otherwise, continue to check the next frame;
2. Check the frame, if the page has been accessed recently, set the accessed bit to false and continue to check the next frame(Go to step 1);
3. Otherwise, if the page has a dirty bit, there are two cases:
   1> if it is a mmap file, write it back to the file
   2> if it is executable file, swap it out.

Go to step 4;
4. Remove this frame from the frame list, free the page and set loaded to false;
5. Call palloc_get_page(flag) and return a free frame.

>> 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 a frame is freed or evicted, we will call
pagedir_clear_page() and palloc_free_page() to clear it.
Then if the frame is obtained by other processes, we will
set the “owner” in frame entry to current_thread(),
set new “spte” and new frame address. In this way, no old
process information will be left.

>> 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.

If an supplemental page is not found in page table, we will call grow_stack().
We set the heuristic to be 32, because PUSHA might cause an
access to be 32 bytes below stack pointer. So we check the user
memory address, if it is between PHYS_BASE and esp - 32, it is valid.
But the max size of a stack is 8MB, so it is beyond this size,
it is invalid.
(refer to https://github.com/ryantimwilson/Pintos-Project-3/blob/dd973a541d9c6b4009b00423e05a1f377ef5c11a/src/userprog/syscall.h#L14)

---- SYNCHRONIZATION ----

>> 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?

As we mentioned in B2, a process will only evict a frame when
its
“pinned” field is false.
So when process Q is using frame A, it will set its “pinned”
field
to true, thus process P cannot evict this frame.
If Q is not using frame A, then process P can evict it.
After evicting it, process P will set the “loaded” field to be
false, so next time, when process Q wants to use the
supplemental
page, it needs to reload it.

>> 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?

If a frame is still being read in, its “pinned” field will be
set to true, so process P cannot evict it.

>> 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?

During system calls, we will check the pointer before we call
any function. So if there exists a page fault, we will call
load_page() or grow_stack() to get the page. We
also set “pinned” to lock the pages. If a virtual address
is invalid, process will exit.

---- 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 use several locks in order to achieve parallelism. As
described
before, we have a boolean field to detect some page faults.

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 “page.h”:

```
// mmap_entry is am entry to record the supplemental page of
// a memory mapped file.
struct mmap_entry {
  struct spt_entry *spte;    // point out the related
supplemental page table
  int mapid;                 // map id
  struct list_elem elem;
};
```

In “exception.c”:

```
struct list mmap_list;  // list of mmap entry;
int mapid;              // Incrementes every time when a file is
mapped.
```

---- 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.

Memory mapped files are lazy loaded. When a page fault
occurres, it will read from the file just like the executable
files. This is the same for loading page. The only difference
is in eviction and munmap.
Mmap files are recorded in the mmap_list, so in munmap, we get
the page from mmap_list, and write dirty page back to disk.

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

The file is mapped page by page. When a new page is mapped,
before adding it to the page table, we will check if it
already exists or not. If it already exists, return false.
Otherwise, create a new map file page, and call hash_insert()
to insert it to the page table.

---- 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.

Both mmap file and executable files are lazy load, so we use
the same load function in these two situations.
But in frame_evict(), if the page has a dirty bit and is a mmap
file,
we write it back to disk. Otherwise, if it is an executable
file, we call
swap_out(), this is an extension.

## 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?
```