```
+---------------------------+
|     CSE 421/521           |
| PROJECT 3: VIRTUAL MEMORY |
|   DESIGN DOCUMENT         |
+---------------------------+
```

---- GROUP: pint-US-pa3 ----

Allen Daniel Yesa <allendan@buffalo.edu>
Bharadwaj Parthasarathy <bparthas@buffalo.edu>
Nimisha Philip Rodrigues <nr57@buffalo.edu>

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 thread. h
struct hash sup_page_table - each thread's supplementary page table

In page.h

struct page_entry - Supplementary page table entry

```
{
    uint8_ptr_t user_page;
    file_ptr_t file;
    off_t ofs;
    uint32_t read_bytes;
    uint32_t zero_bytes;
    bool writable;
    struct hash_elem elem;
};
```

In frame.h - Frame table entry

```
struct frame_entry
{
    page_entry_ptr_t page;
    thread_ptr_t thread_ptr;
    void_ptr_t frame_ptr;
    uint32_t order;
    struct hash_elem elem;
};
```

---- ALGORITHMS ----

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

1. During page fault the Supplementary page table is looked up using the virtual address.
2. If there is an entry in SPT, then a frame has to be allocated to it.
3. If there is a frame available then it is allocated otherwise the frame is evicted according to an algorithm.
4. The virtual address is mapped to the corresponding physical address
5. When the kernel runs the faulting instruction again it will map to the correct memory allocated

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?

In Pintos, every user virtual page is aliased to its kernel virtual page. We could avoid the problem by making the kernel to access user data only through the user virtual address.

---- SYNCHRONIZATION ----

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

When two user processes require a new frame we use lock to lock the frame and provide the frame one process after another.

---- RATIONALE ----

A5: Why did you choose the data structure(s) that you did for representing virtual-to-physical mappings?
We are using Hash tables to implement the Supplementary page table. Hash data structure is used for ease of insertion and deletion. And also it is easy to find an element in as hash table. This is used to find a supplementary page table entry with the help of virtual address.

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.

---- 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 are planning to implement clocks algorithm. We sequentially scan the frames, looking for frames that are not in use and which has a page that has not been accessed recently. This is done by using the accessed and dirty bits. Each frame that has a page that has been accessed recently will have its accessed bit cleared, and then that page will be skipped. Once the algorithm finds a frame that has not been accessed recently, the associated page will be evicted, and that frame will then be used for the desired page.

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 process P obtains a frame that was previously used by a process Q we know the frame is from Q using the page table entry. We will free the page from the process Q and remove the Q's reference to the frame.

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.

Memory access beyond PHYS_BASE and esp - 32 is invalid. We check to be sure that the faulting address occurs between 32 bytes before the current stack pointer (to handle some edge cases with embedded assembly push instructions) and extend the stack by one page else the access is considered invalid and the user program is killed.

---- 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 will maintain a Boolean to identify if the frame is allocated. If a fame is allocated, then the algorithm will not consider it for eviction.

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 eviction the first thing to do clear the pages so that none of the data can be accessed by the second process.

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?

Same as B5, the Boolean value will prevent the frame from getting evicted.

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 the pages are checked and the pages will be brought back if they are missing and a flag will be set to the corresponding frames to prevent evection. While checking paged-out pages the invalid addresses will also be checked and detected and rejected and the process will be made to 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 are yet to implement. We will get to know this once we implement the design.

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.

---- 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.

Mmap and munmap will be handled via system calls. Mmap will load the file into the memory and get the corresponding details and the munmap will free the memory clearing all pages.

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

We are planning to keep a list of mappings, and each mapping will contain the base address and size. We can use this information to determine whether a new file mapping overlaps
any existing segment

---- 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.
We are planning to share the code for two situations because both have similar semantics and it also prevents code redundancy.

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?