+---------------------------+

| CSE 521 |

| PROJECT 3: VIRTUAL MEMORY |

| DESIGN DOCUMENT |

+---------------------------+

---- GROUP ----

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

Vigneshwaran Vasanthakumar <vvasanth@buffalo.edu>

Muthuvel Palanisamy <muthuvel@buffalo.edu>

Siddharth Selvaraj <ss623@buffalo.edu>

---- 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.

* <http://bits.usc.edu/cs350/assignments/project3.pdf>
* http://bits.usc.edu/cs350/assignments/Pintos\_Guide\_2016\_11\_13.pdf

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 frame.h:**

**struct frame /\* frame table holds the information about**

**{**

**uint32\_t \*frame\_ptr;**

**struct thread\* owner;**

**struct sup\_page\_entry\* aux;**

**void \*physical\_frame;**

**struct list\_elem elem;**

**};**

**/\* Frame table - holds info about frames \*/**

**struct list frame\_table\_list;**

**In page.h:**

**struct supplemental\_page\_table**

**{**

**void \*page\_uaddr;**

**struct hash\_elem spt\_elem;**

**void \*physical\_frame;**

**off\_t file\_ofs;**

**struct file \*file;**

**uint32\_t read\_bytes;**

**uint32\_t zero\_bytes;**

**void \*page\_owner;**

**bool read\_or\_write;**

**};**

**In thread.h:**

**struct thread**

**{**

**struct hash spt\_hash\_table;**

**};**

---- ALGORITHMS ----

>> A2: In a few paragraphs, describe your code for locating the frame,

>> if any, that contains the data of a given page.

**First we will check if the corresponding frame (page – user virtual address) is present in the page directory of the process, if it is present then we can directly locate it using the page table.**

**If it is not present, then page fault will occur. In this case we will check the supplemental page table and if the reference is valid then the data will be located based on whether it is in the file system or in a swap slot. In case of an invalid access the process will be terminated and all its resources will be freed.**

>> 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 our implementation this issue will not occur as we have decided to access the user data only through user virtual address as suggested in the pintos guide.**

---- SYNCHRONIZATION ----

>> A4: When two user processes both need a new frame at the same time,

>> how are races avoided?

**We will acquire a lock on the function which is used to allocate a new frame. In our case this will be palloc\_get\_page(). This will eliminate the race conditions.**

---- RATIONALE ----

>> A5: Why did you choose the data structure(s) that you did for

>> representing virtual-to-physical mappings?

**This was the easiest to implement since most of the functions used to access and manipulate these data structures is already present in pintos itself.**

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.

**We are still working on the actual implementation.**

---- 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.

**Inorder to identify the** **frame which has to be evicted we will be implementing the second chance list algorithm i.e we will be maintaining an active list and a second chance list. Actual eviction is done as follows: when a frame is required but none is free then, palloc\_get\_page returns a NULL. The information of the page mapped by the frame into the disk is written and the frame is cleared using pagedir\_clear\_page. Then the frame is reallocated to another page using palloc\_get\_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?

**We will remove the page from Q’s page directory, we can use pagedir\_clear\_page function to remove the page from Q’s page directory. This will remove any association of the page from process Q.**

>> 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 will first limit the size of the stack to 8MB as suggested in pintos guide. If the address is below the 8MB of the stack, then we will not extend the stack. The current stack pointer will be retrieved from the esp member of the struct intr\_frame. Based on this we will be able to identify whether the page fault address is beyond the stack size. Also we will check whether the pointed address is a kernel address or not as a safety mechanism using is\_kernel\_vaddr function.**

---- 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 are going to use only one lock variable to lock the commonly used method/resources across the process and restrict their usage to a single thread, throughout our synchronization design. Multiple locks are used only on methods that do not share a resource with another method holding a different lock. This will eliminate the possibility of multiple threads causing a deadlock over a shared resource.**

>> 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?

**We will design our code in such a way that any process which is accessing a frame has to acquire a lock on the supplemental page table until the the action gets completed (in this case it is eviction). So now Q will have to wait for P to complete the eviction process before it looks up the supplemental page table for information about the desired frame.**

>> 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 will be using a lock or using a variable to denote that the particular data is still being read and any page with this lock or variable will not be chosen for eviction in our replacement algorithm. We are still working on the implementation specifics.**

>> 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?

**As of now we have decided to use page fault itself to bring in the pages. In system calls, the virtual address pointer will be checked for its validity in the system call handler before it is used.**

---- 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 going to design our locks in such a way that our code will use a lock on a resource/method only when it is different (when they are mutually exclusive) from another lock on another method i.e. when the methods do not share any resources which could cause a deadlock.**

**Hence our program will not be restricted to having just one lock, reducing parallelism while it doesn’t also complicate synchronization.**

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.

**The entire file will be mapped into consecutive virtual pages starting at addr.** **During eviction** **a page mapped by mmap will be written back to the file it was mapped from but other pages will not be written back to the file.**

>> C3: Explain how you determine whether a new file mapping overlaps

>> any existing segment.

**Before we map a file we will determine the number of pages required for it to be mapped and based on that will know the required address limits. Then we will check the page directory and supplemental page table to identify whether that particular address is already mapped to any other frame. We will do the mapping only when it is not used by any other frame. If successful, the mapping ID will be returned otherwise we will return -1.**

---- 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.

**Implementation is still in progress.**

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?