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

| CS 140 |

| PROJECT 3: VIRTUAL MEMORY |

| DESIGN DOCUMENT |

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

---- PHOENIX ----

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

Akshay Chopra <achopra6@buffalo.edu>

Samraddha Dubey <samraddh@buffalo.edu>

Sandeep Kumar <skumar28@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.

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.

struct supplemental\_page\_table {

void \*page;

bool writable;

struct file \*file;

size\_t offset;

size\_t read\_bytes;

size\_t zero\_bytes;

struct supplemental\_page\_table \*next\_elem;

};

---- ALGORITHMS ----

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

frame,

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

We will check it if virtual address has a entry in page table.

that entry will give us the frame location.

If it is not mapped than we can load it using supplemental page

table.

>> 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?

The kernel will only access user data

through user virtual address thus avoiding this issue.

---- SYNCHRONIZATION ----

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

time,

>> how are races avoided?

Allocation of new frame will be synchronized using the lock.

---- RATIONALE ----

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

>> representing virtual-to-physical mappings?

We used linked list for supplemental page table. It is easy to maintain.

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

---- 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.

I would be done using a page replacement algorithm, the clock-algorithm implementation of second-chance algorithm. If frame is free, the frames used for user pages should be obtained from the "user pool," by calling palloc\_get\_page(PAL\_USER). When none is free, a frame must be made free by evicting some page from its frame. We will choose a frame to evict, using your page replacement algorithm by using "accessed" and "dirty" bits in the page table. On any read or write to a page, the CPU sets the accessed bit to 1 in the page's PTE, and on any write, the CPU sets the dirty bit to 1. The CPU never resets these bits to 0. Remove references to the frame from any page table that refers to it.

>> 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?

Remove references to the frame from any page table that refers to it by the process Q. We can use pagedir\_clear\_page method. Marks user virtual page UPAGE "not present" in page directory PD. Later accesses to the page will fault. Other bits in the page table entry are preserved. UPAGE need not be mapped

>> 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.

To distinguish stack accesses from other accesses. We will need to be able to obtain the current value of the user program's stack pointer. Within a system call or a page fault generated by a user program, we can retrieve it from the esp member of the struct intr\_frame passed to syscall\_handler() or page\_fault(),

respectively and we will verify user pointers before accessing them. We will check whether the address lies between 32 bytes below the stack pointer. If so, then the page fault has been caused because of the stack and we will extend the 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.)

It would be implemented by using locks for synchronization and use it efficiently to prevent deadlock.

>> 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 can use synchronization techniques such as lock or semaphore

to ensure this behavior.

>> 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 can have a variable in page table entry data structure to

avoid this situation.

>> 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?

If paged-out pages occur, We will bring back the page.

Invalid virtual addresses will be checked and process will be

exited.

---- 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.

Many locks will give parallelism at the same time we can avoid possibility of deadlock.

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.

Maps the file open as fd into the process's virtual address space. The entire file is mapped into consecutive virtual pages and a page table entry is created.

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

overlaps

>> any existing segment.

If the page addresses needed by new file has been already used by the thread (we can check it in page directory of thread) then new file mapping overlaps.

---- 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.

Yes, much of their implementation will be shared but paging out is different as a memory mapped page goes right to file whereas the file system page gets swapped out.

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?