Project 4 part 1 (4A): Paging

Consult the submit server for deadline date and time

1 Overview

[Background: The *virtual address* generated by an instruction consists of a 16-bit segment selector and a 32-bit offset, which together yield a 32-bit *linear address*. (Specifically, the selector points to a segment descriptor which yields a segment base address to which the offset is added). Without paging, the linear address is the same as the physical address. With paging, the linear address is mapped to a physical address via 2-level page table.]

In this part of the project, you will construct a page table that directly maps linear addresses to physical addresses. That is, linear and physical addresses in the kernel will be identical.

In the next part, linear addresses from user processes will be mapped to physical addresses that may not be contiguous or even in physical memory at all. You will also implement a page replacement algorithm, allow user code to map pages on demand in response to recursive calls in the stack, and mark code and read only data pages as read-only.

A functioning Fork, Pipe, Signals, and blocking pipe are *not* required for this project or any later project. Making Fork() work properly with page tables is a bit of a challenge. Consider adding a "return EUNSUPPORTED" at the top of Sys_Fork().

2 Reference

The key reference for this project is the intel manual volume 3A, `http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html`

Download and enjoy, in particular chapter 4.3.

3 Superpages

You may[1] use 4MB pages in the identity map, however, do not map addresses that are beyond the memory in the machine (except for the APIC as below) and so not physically present, and do not map the 0th page.

Note that by not mapping the 0th page, you may expect to find null pointer dereferences in old code.

4 The APIC

The APIC controls interrupts, including inter-processor interrupts, and is used in the implementation of CURRENT_THREAD to determine which processor is currently running the thread.

The APIC and IO-APIC pages will need to be identity mapped into the address space of all processes at locations 0xFEE00000 and 0xFEC00000. These pages should be mapped READ/WRITE but only from ring 0 not ring 3. That is, they will not be accessible from user code.

[1] "may" means you may choose to if you like, but are not required to. I think it's a nice little challenge, yields a more compact TLB footprint, and is how Linux did kernel-side page tables for a long time.

The page fault handler is already equipped to terminate a user program that faults while accessing these regions; it will be your task, eventually, to ensure that user code cannot access these pages while the system calls that the user code invokes (running with the same page table) can.

5 TODOs

Generally, the TODO macros associated with this project are VIRTUAL_MEMORY_A. Some features tagged _A may not be required as part of the first milestone.

To set up page tables, you will need to allocate a page directory (via Alloc_Page) and then allocate page tables for the entire region that will be mapped into this memory context. You will need to fill out the appropriate fields in the page tables and page directories. The definition of paging tables and directories are to be found in paging.h (structs pte_t and pde_t). Finally, to enable paging for the first time, you will need to call the routine Enable_Paging(pdbr) which is already defined for you in lowlevel.asm. It takes the base address of your page directory as a parameter.

The final step of this function is to add a handler for page faults. A default one named Page_Fault_Handler in paging.c has been provided for you. You should install it by calling Install_Interrupt_Handler. You need to register this as a handler for interrupts 14 and 46. You should then add a call the Init_VM function from your main.c (after Init_Interrupts).

You should be able to do this step and test it by itself by temporarily giving user mode access to these pages - set the flags fields in the page table entries to, for now, include VM_USER. Once you have this running, you can submit it as your intermediate submission (project 4a).

6 Hints

This is a preliminary assignment meant to prepare you for a more elaborate demand paging assignment, and to make sure you don't procrastinate too much. Use this opportunity to develop helper functions that you may find useful.

Functions in uservm are generally variations on the functions in userseg. To use uservm instead of userseg, modify Makefile.common. This switch is necessary for part 4B.

I believe the following statement was a workaround for a buggy Qemu, but is no longer accurate: "To enable paging on a secondary core requires trapping both interrupt 14 *and* 46." I mention it because perhaps you may see an unexpected interrupt 46.

7 Test

There is only one test: does "checkPaging" return / print that paging is enabled? Call it in main(). Paging must be enabled on each processor core.

8 Submission

To submit this intermediate submission, change your .submit file to submit "4a", and to submit the final submission, change your .submit for "4b".