Project 4a: Paging

Due April 3

1 Overview

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

In 4b, user processes will switch to a virtual address space which may not be contiguous or even in physical memory at all. You will also implement a page replacement algorithm, allow user code to Malloc, and mark code and read only data pages as read-only.

A functioning Fork, Pipe, and Serial port are *not* required for this project. Making Fork() work properly with page tables is a substantial challenge.

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 Requirements

Only pages corresponding to physical memory should be mapped, do not identity map the entire 4 GB virtual address space. Details on the memory contents can be found in "mem.c".

Leave the first page (addresses 0x0 - 0x1000) unmapped in order to catch NULL dereferences. You may have to modify code from previous projects to remove all NULL dereferences.

Paging should be enabled on all processors. Only one processor will run the code in Init_VM; secondary processors will instead call Init_Secondary_VM.

Make sure the helper function "Kernel_Page_Dir" returns a pointer to your kernel page directory.

4 Superpages

You may[1] use 4MB pages in the identity map, however, as above, 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.

5 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.

[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 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.

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.

6 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 on the *current* processor, 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 to 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.

7 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.

Enabling paging on a secondary core requires trapping both interrupt 14 *and* 46.