Lab 2: Virtual Memory fun

Handed out Thursday, Feb. 11, 2016

Due Tuesday Feb. 23, 2016 (8 pm)

Credit: based almost completely on assignment by Daniel Myers

Project 2: The Null Pointer

Objectives

There are two objectives to this assignment:

To familiarize you with the xv6 virtual memory system.

To add a new VM features to xv6 that is common in modern OSes.

Overview

In this project, you'll be changing xv6 to support a feature available in virtually every modern OS:

```
raising an exception when your program dereferences a null pointer.
```

Sound simple? Well, there are a few tricky details.

Details

In xv6, the VM system uses a simple two-level page table. If you do not remember the details, read Section 20.3 of OS 3 easy steps. However, you may find the description in Chapter 1 of the xv6 manual sufficient (and more relevant to the assignment).

> As it currently is structured, user code is loaded into the very first part of the address space. Thus, if you dereference a null pointer, you will not see an exception (as you might expect); rather, you will see whatever code is the first bit of code in the program that is running. Try it and see!

Thus, **the first thing you might do is create a program that dereferences a null pointer.** It is simple! See if you can do it. Then run it on Linux as well as xv6, to see the difference.

Your job here will be to figure out **how xv6 sets up a page table.** Thus, once again, this project is mostly about understanding the code, and not writing very much. **Look at how exec() works** to better understand

how address spaces get filled with code and in general initialized. That will get you most of the way.

You should also look at fork() , in particular the part where **the address space of the child is created by copying the address space of the parent.**

> What needs to change in there?

The rest of your task will be completed by looking through the code to

> figure out where there are checks or assumptions made about the address space.

Think about what happens when you pass a parameter into the kernel (using a system call), for example; if passing a pointer, the kernel needs to be very careful with it, to ensure you haven't passed it a bad pointer.

> How does it do this now? Does this code need to change in order to work in your new version of xv6?

One last hint

you'll have to look at the xv6 makefile as well. In there user programs are compiled so as to set their entry point (where the first instruction is) to 0. If you change xv6 to make the first page invalid, clearly the entry point will have to be somewhere else (e.g., the next page, or 0x1000). Thus, something in the makefile will need to change to reflect this as well.

You should be able to demonstrate what happens when user code tries to access a null pointer. If you do this part correctly, xv6 should trap and kill the process without too much trouble on your part.

The Code

Download a fresh copy of the kernel.

You may also find the following readings about xv6 useful: xv6 book.

Particularly useful for this project: Chapter 1 + anything else about fork() and exec(), as well as virtual memory.