Project 1: Introduction to xv6

Overview

Kernel Intro: to be done in our xv6 hacking environment, so you can learn more about what goes on in a real kernel.

We will be doing kernel hacking projects in `xv6`, a port of a classic version of UNIX to a modern processor, Intel's x86. It is a clean and beautiful little kernel, and thus a perfect object for our study and usage.

This first project is just a warmup, and thus relatively light on work. The goal of the project is simple: to add a system call to xv6. Your system call, `getsyscallinfo()` , simply returns the value of a counter.

Specifically, you need to add one counter that increments every time a system call is made. The counter should be incremented before each time a system call is issued; when `getsyscallinfo()` is called, the calling program can thus discover how many total system calls have been made.

Details

Your new `syscall` should look like this: `int getsyscallinfo(void)`.

Your system call returns the number of calls made since the system booted on success, and `-1` on failure.

The count for a system call should only be incremented **before** the call is finished executing, not after.

Tips

Find some other system call, like `getpid()`. Basically, copy it in all the ways you think are needed. Then modify it to do what you need.

Most of the time will be spent on understanding the code. There should not be a whole lot of code added.

Using `gdb` (the debugger) may be helpful in understanding code, doing code traces, and is helpful for later projects too. Get familiar with this fine tool!

The Code

The source code for `xv6` is include in the ***ICS462.ova*** VirtualBox image in the `~/ICS462/CODE/xv6-public` directory. Everything you need to build and run and even debug the kernel is in there; start by reading the README.

You may also find the following book about xv6 useful, written by the same team that ported xv6 to x86: **book** . **Particularly useful for this project: Chapters 0 and 3 (and maybe 4).**

General Advice

Start small, and get things working incrementally. For example, first get a program that simply reads in the input file, one line at a time, and prints out what it reads in. Then, slowly add features and test them as you go.

Testing is critical. One great programmer I once knew said you have to write 5-10 lines of test code for every line of code you produce; testing your code to make sure it works is crucial. Write tests to see if your code handles all the cases you think it should. Be as comprehensive as you can be. Of course, when grading your projects, we will be. Thus, it is better if you find your bugs first, before we do.

Keep old versions around. Keep copies of older versions of your program around, as you may introduce bugs and not be able to easily undo them. A simple way to do this is to keep copies around, by explicitly making copies of the file at various points during development.

Keep your source code in a private directory. An easy way to do this is to log into your account and first change directories into `private/` and then make a directory therein.

Submission

Copy all your source files (but not `.o` files, please, or binaries!) into your projects' private directory (i.e., `~/ICS462/CODE/project1/xv6-private`). A simple way to do this is to copy everything into the destination directory, then type `make` to make sure it builds, and then type `make clean` to remove unneeded files.

```
shell% cp -r . ~/ICS462/CODE/project1/xv6-private

shell% cd ~/ICS462/CODE/project1/xv6-private

shell% make

shell% make clean
```

Finally, into your …`/project1/xv6-private` directory, please make a README file. In there, describe what you did a little bit. The most important bit, at the top, however, should be the authorship of the project.

Zip your …`/project1` directory and upload it into **D2L ➔ Assessments ➔ Assignments ➔ Project #1:…** folder.

Note: you are only allowed to submit a single file (the last uploaded file).