Project 1b: xv6 System Call

To develop a better sense of how an operating system works, you will also do a few projects inside a real OS kernel. The kernel we'll be using is a port of the original Unix (version 6), and is runnable on modern x86 processors. It was developed at MIT and is a small and relatively understandable OS and thus an excellent focus for simple projects.

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, **getreadcount()**, simply returns how many times that the **read()** system call has been called by user processes since the time that the kernel was booted.

Background

You might want to read this background section.

More information about xv6, including a very useful book written by the MIT folks who built xv6, is available here. Do note, however, that we use a slightly older version of xv6 (for various pedagogical reasons), and thus the book may not match our code base exactly.

The xv6 kernel will be running in a emulator called QEMU, which you can download from here and install with `make`.

After you have un-tarred the `xv6.tar.gz` file, and compiled xv6 using `make`, you can run `make qemu-nox` to run xv6 using the QEMU emulator. Test out the unmodified code by running a few of the existing user-level applications, like `ls` and `forktest`. To quit the emulator, type `Ctrl-a x`.

Your System Call

Your new system call should look have the following return codes and parameters:

```
int getreadcount(void)
```

Your system call returns the value of a counter (perhaps called **readcount** or something like that) which is incremented every time any process calls the **read()** system call. That's it!

Tips

One good way to start hacking inside a large code base is to find something similar to what you want to do and to carefully copy/modify that. Here, you should find some other system call, like **getpid()** (or any other simple call). Copy it in all the ways you think are needed, and then modify it to do what you need.

Most of the time will be spent on understanding the code. There shouldn't 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!

Adapted from WISC CS537 by Remzi Arpaci-Dusseau