Intro To Kernel Hacking

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.

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

Before hacking xv6, you need to succesfully unpack it, build it, and then modify it to make this project successful. Xv6+Qemu is strongly recommended, you can get start by following this small guide:

1. Get the latest xv6 source.

   ```
   prompt> git clone git://github.com/mit-pdos/xv6-public.git
   ```

2. Test your compiler toolchain.

   ```
   prompt> objdump -i
   ```

   The second line should say elf32-i386.

   ```
   prompt> gcc -m32 -print-libgcc-file-name
   ```

   The command should print something like /usr/lib/gcc/i486-linux-gnu/version/libgcc.a or /usr/lib/gcc/x86_64-linux-gnu/version/32/libgcc.a
   If both these commands succeed, you're all set, and don't need to compile your own toolchain. Otherwise, trouble shooting.
3. Compile xv6.

   ```
   prompt> ./Xv6-master/make
   ```

4. Install qemu.

   ```
   prompt> sudo apt-get install qemu
   ```

5. Run xv6.

   ```
   prompt> ./Xv6-master/make qemu
   ```

   If success, you can run **ls** to see all command files.

You might want to read background.html first. More information about xv6, including a very useful book written by the MIT folks who built xv6, is available here.

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!

Running Tests

Before running tests, get tcl, tk, expect installed in your working operating system:

```
prompt> sudo apt-get install tcl tk expect
```

Running tests for your system call is easy. Just do the following from inside the `initial-xv6` directory:

```
prompt> ./test-getreadcounts.sh
```

If you implemented things correctly, you should get some notification that the tests passed. If not ...

The tests assume that xv6 source code is found in the `src/` subdirectory. If it's not there, the script will complain.

The test script does a one-time clean build of your xv6 source code using a newly generated makefile called `Makefile.test`. You can use this when debugging (assuming you ever make mistakes, that is), e.g.:

```
prompt> cd src/
prompt> make -f Makefile.test qemu-nox
```

You can suppress the repeated building of xv6 in the tests with the `-s` flag. This should make repeated testing faster:

```
prompt> ./test-getreadcounts.sh -s
```