COP4610: Introduction to Operating Systems
Project 1: Adding a System Call to xv6

1 Overview

In this assignment, you will get to know xv6, a simple Unix-like teaching operating system from MIT. You'll do three things in this assignment:

- You'll get xv6 to run on a system emulator, QEMU; relax, this is a pretty quick and painless process.
- You'll trace the `read()` system call of xv6 and document what goes on at each step.
- You'll implement the `top` program which lists the processes in the system by memory consumpsion; for this to work, you'll have to carefully introduce a new system call.

Words of Wisdom: First, please start early! Second, please make minimal changes to xv6; you do not want to make it hard for us to grade!

2 Part 1: xv6 on QEMU (20%)

The first part is easy. You need to download xv6, compile it and make a disk image, and get that disk image to run on QEMU. Grabbing the xv6 source here (MD5 Checksum: `729d72a29f64673a5ddb51f7174a03fa`). Extract the sources, then take a quick look around.

Compiling xv6 should work fine by just saying `make` in the xv6 directory. You'll see a lot of lines fly by, and eventually you'll have an image file `xv6.img`. Running xv6 on `QEMU` should work fine by just saying `make qemu` in the xv6 directory. And there you have it. Try running `ls` and `mkdir` and friends. You can even run `sh` to get a new shell process (use `CTRL-a x` to leave it as there's no `exit` command).

Please document this process in the submission document. You won't submit the images or anything, we'll just have to see some kind of indication that you actually got things to run by, say, taking a screenshot of the running xv6.

Troubleshooting: one of the important skills for a computer science student is to address problems using online search engine. It is expected that you will be able to complete this project by yourself. However, here are some hints you may find useful.

1. The projects in this course are best to complete in a recent version of the Linux operating systems, such as `Ubuntu LTS 14.04`. It is relatively easy on Mac OS X by following the instructions here. For students that have Windows, it is suggested to install the Ubuntu system in a virtual machine. You may use the freely available VirtualBox for this purpose. If your keyboard/mouse is "trapped" in the `qemu` window, press `Ctrl-Alt` to exit mouse grab.

2. You can also work on the project on (`linprog.cs.fsu.edu`). Remotely login to the server through `ssh`. To compile and run xv6, use the make qemu-nox command, which compiles xv6 and runs it in QEMU *without* X window.

3. The kernel is generated as a 32-bit binary. Make sure your Linux system can compile 32-bit code if your Linux installation is 64-bit.

4. You need to install `qemu` to your system using the online repository of your Linux distribution or building your own. You need to update the Makefile of xv6 so `make qemu` can find the `qemu` binary (usually `qemu-system-i386`) if `make qemu` still fails after installing it.

3 Part 2: A read() Encounter (30%)

The second part is a bit harder, but luckily you've seen us trace a system call before in lecture. You'll just have to write it up in a little more detail for this problem.

You'll trace the `read()` system call. Assume that a user program executes the line `read(10, buf, n)` and nothing else, i.e. it never `open()` ed any files (hint: `argfd` returns `-1`.) You need to document in the submission exactly **which lines in which source files are executed** as a result of this system call. Please don't skip anything. Try to explain each coherent chunk of lines, i.e. tell us why that chunk runs and what purpose it serves. **Your description must start in the userspace with the fictitious line above, go to the kernel space, and return back to the userspace.**

4 Part 3: Top (50%)

Type `man top` into a Unix/Linux system somewhere and read the documentation. Read it to understand what `top` does. Now relax, there is no need to implement all those command line options: we'll simply print one fixed set of interesting values about each process. But you should become aware of all the stuff a real Unix system keeps track of, xv6 is a lot simpler than that.

If you examine the source code for xv6 you'll find the file `proc.h` somewhere. In it, you find what we are looking for: `struct proc`. When xv6 is running, all that information is around for each process, **in the kernel**! Since `top` is a user space program, it cannot access the process table in the kernel. In modern Unices, for example in Linux, the `/proc` file system provides all the information to implement `top`, but we don't want to add a whole new file system to the kernel. Instead we'll have to add the next best thing, a new system call. But before we get to that, let's agree on the output `top` will make for each process. You'll print the following pieces of information for each process, separated by two spaces each:

`process id` (as decimal integer)
`state` (as string, all caps)
`size` (as decimal integer)
`name` (as string)

You'll print one line for each process sorted in decending order by memory size and then by name if the sizes match. You don't need to print a header at the top of the table. And that's it, that's our output. Note that we left out some internals, but that's okay: `top` is a user program after all.

Now, you will add just one system call. Your call will not always be able to print the correct information. In C, the system call you need to add to xv6 has the following interface:

```
int getprocs(int max, struct uproc table[]);
```

struct uproc is defined as (add it to a new file `uproc.h`):

```
struct uproc {
        int pid;
        int state;
        uint sz;
        char name[16];
};
```

Your `top` program calls `getprocs` with an array of `struct proc` objects and sets `max` to the size of that array (measured in `struct uproc` objects). The kernel copies up to max entries into your array, starting at the first slot of the array and filling it consecutively. The kernel returns the actual number of processes in existence at that point in time, or `-1` if there was an error. Note that the kernel defines the process state as an enumerate (`enum procstate`). An enumerate is just an integer in disguise. However, your `ps` program should print it as a string.

The problem with this call is that if there are more processes than you have space for, you will miss some. But then if you use the returned integer to allocate a new table and call again, some processes may have died or (worse!) new ones may have been created in the meantime. So you'll have to be lucky to get the perfect listing for a busy system.

5 Deliverables

Parts 1 and 2: Please turn in the first two parts by submitting the document you write in the blackboard. Use plain text `.txt` or `pdf`; do not use `Microsoft Word` or other rich format. You can convert `Word` file to `pdf` with this plugin from Microsoft.

Part 3: Submit in the blackboard your modified source code for `xv6`, including all files necessary for a successful build that includes your `top` program as a `gzip` compressed tarball. The name of your attachment should be

`cop4610-project1-yourname.tar.gz`

with `yourname` replaced by your CS account name.

Your submission will be graded by compiling and running it. We will check the presence of the `top` command and that it returns the correct results.

- Please make sure your source code can compile. Absolutely no credit for part 3 if it does not compile.
- Please don't include the binary files. Do a `make clean` before submission. You're make grading harder for us if you do.
- Please don't leave out any files! You're make grading harder for us if you do.
- Please don't modify any files you don't need to! You're make grading harder for us if you do.
- Please don't send us the meta-information from your revision control system! You're make grading harder for us.

And if we have a hard time grading, you'll have a hard time getting points!

6 Useful References

☞ Xv6 book/commentary, Chapter 3
☞ Xv6 Homepage

7 Acknowledgment

This project is derived from 600.318/418: Operating Systems at John Hopkins University.