CS 537 - Introduction to Operating Systems - Fall 2017

Site /

KernelIntro

Overview

We'll 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.

Details

Your new syscall should look like this: int getppid(void)

Your system call returns the process ID of the parent of the calling process.

New processes (except the first one) are created by other processes calling `fork()`. After a successful call to `fork()`, the newly created process is a child of the caller process, and the caller process is considered the parent of the new process. `fork()` returns the pid of the child to the parent, but returns 0 to the child. One way for the child process to get the pid of its parent is to call `getppid()`, which is available in Linux but missing in xv6.

Note that the first process is constructed by the kernel directly and does not have a parent. In this case, your system call should return -1.

The Code

The source code for xv6 (and associated README) can be found in `~cs537-1/ta/xv6/`. Everything you need to build and run and even debug the kernel is in there.

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. (Please ignore the message saying `Could not open option rom 'sgabios.bin': No such file or directory`) 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`.

Using `gdb` (the debugger) may be helpful in understanding code, doing code traces, and is helpful for later projects too. Look at the Makefile to see how to start up the debugger. Get familiar with this fine tool!

You will not write many lines of code for this project. Instead, a lot of your time will be spent learning where different routines are located in the existing source code.

You may also find the following book about xv6 useful, written by the same team that ported xv6 to x86: book. However, note that the kernel version we use is a little different than the book.

Particularly useful for this project: Chapters 2 and 5.

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.

Testing

First, let's make sure you have a working test program. The idea here is simple: we write a `testppid.c` program for the Linux we are using, and then port it to xv6.

Your program should first save its own pid in an variable, and then forks into two processes: the parent simply waits for the child to exit, while the child calls `getppid()` and compares it with the saved pid. The child should print success or failure. Compile and run it under Linux. If there is an error, you should check your implementation of `testppid.c`. This also helps you understand how `fork()` and `wait()` works.

Once your `testppid.c` works correctly under Linux, you can port it to test xv6. Note that system calls and library functions have slightly different signatures under Linux and xv6. For example, `wait()` in xv6 does not have any parameters. Take a look at the `user/` folder to see how to write and add user programs in xv6. Do not forget to change the C headers in your programs as well!

If your modified `testppid` works in xv6, you are ready to move on to our real test cases. This `testppid.c` does not need to be submitted and will not be graded, but it gives you an idea of how testing should work.

To run the test cases, change directory into the root of your xv6 directory. The tests run with `~cs537-2/ta/tests/1b/runtests`. (NOTICE: directory starts with **~cs537-2**) If you failed for a specific test, you can find it in `~cs537-2/ta/tests/1b/ctests/`. Open it and find out what aspect it is testing for, how is your xv6 reacting, and why it does not execute as expected.

Turning it in

For the xv6 part of the project, copy all of your source files (but not .o files, please, or binaries!) into the @xv6/@ subdirectory of your p1 directory. A simple way to do this is to copy everything into the destinaation directory, then type make to make sure it builds, and then type make clean to remove unneeded files.a

```
shell% make clean
shell% cp -r . ~cs537-1/handin/$USER/p1/xv6
shell% cd ~cs537-1/handin/$USER/p1/xv6
shell% make
shell% make clean
```

Finally, into your p1 directory, please include a README file. In there, describe what you did a little bit (especially if you ran into problems and did not implement something). The most important bit, at the top, however, should be the authorship (name and cs login) of the project.

Page last modified on September 21, 2017, at 08:21 PM