TCSS 420, Fall 2015 -- Project #1

Due: Midnight, 10/21/2017

The objective of this project is to add a custom system call to the Linux kernel and call it using a simple user-space program. This requires modifying the Linux kernel, rebuilding with your changes, and booting your modified kernel. Then you will need to build a small user-space test program to test your new system call and ensure it works properly.

This is not a coding-intensive project. Like many kernel-enabling projects in industry, there are only a few minor kernel code changes that are necessary. Similarly, the user-level test program should require less than ~10 lines of code. The critical part is identifying how the kernel manages system calls and understanding how they work sufficiently well that you can modify the mechanisms to enable new functionality.

Your first task is to ensure you can download and build a Linux kernel. The kernel used on the Raspberry Pis is maintained in a github repository, so you'll need to clone the kernel tree from the Raspberry Pi github site as outlined in the instructions at the following link:

https://www.raspberrypi.org/documentation/linux/kernel/building.md (Links to an external site.)Links to an external site.

You should first go through the process of building a new kernel for your Raspberry Pi before embarking on modifying your kernel as required for this project.

Once you have built a new kernel and ensured you are able to boot it, the next task is to modify your Linux kernel to add a new system call into the system call table with a pointer to your system call function (incorporated into an existing C file or a new C file – your choice) that implements the system call. For this project, you will minimally need to issue a simple 'hello world' type printk() function in kernel space that writes your name and a message into the system log. In other words once your system call is called, you should see the printed message (including your name) at the tail of the system log when you issue the dmesg command on the command line.

The third part of this project is to write a simple user-space application (again, think hello world-like program) to exercise your new system call. Please use the man pages on your linux system and look at the documentation for syscalls. Man pages are great references that can help get you started or serve as a quick refresher for the details of how commands work as well as library calls.

The kernel is a large software project and figuring out where to start looking can be intimidating. Consequently, we will review the layout of the

In this file, you'll find examples of existing system calls that constitute the system call table used by Linux. Choose a syscall and see how the syscall number is defined and then linked to a handling function. For example, a good reference would be to look at the "fork" syscall.

Using diff to generate patches: As I will describe in class, to create a diff or patch file to capture the changes you've made to the kernel for this project, you will need two things:

- An original, unmodified copy of the kernel tree you are working with
- Your version of the kernel tree

In other words, you will need to unwind a second kernel tree to diff against to capture your changes. For this you should do two things:

- Rename your original kernel tree. To do so, simply issue the command:

Note that if you do not rename the directory containing your original kernel tree, when you unwind the second, "new" tree, it will overwrite the contents of the original tree.

At the end of this little exercise, the command ls should confirm you have two kernel trees now:

- linux
- linux-yourname. In my case, I'd have linux-matt; Note that you could use some other token besides your name. You just need something to distinguish between the two directories.

Once you have these two trees, make your changes in linux-yourname. Do the entire project using that tree. Once everything works and you are ready to submit, you're ready to generate a patch that captures your changes. Now, do the following:

- In your linux-yourname tree, issue the following command: make mrproper
  - Make mrproper will delete the config file you used to build the kernel as well as all binaries built during the kernel build process. If you want to ensure you retain your config file, be sure to copy it to another directory (recall, the config file is hidden and called .config).
- cd into the directory that contains your kernel trees, such that you can see both directories
- issue the following command: diff –urNp linux linux-yourname > mychanges.patch
  - With these arguments, diff will recurse all directories and compare all files, noting any and all differences. By generating a diff file, others can use that difference file to "patch" another tree with your changes.

Project Submission Details:

Please submit the following via the Canvas site for the class by the deadline:

- Diff file for your kernel changes. Diff files are convenient ways to exchange source code changes to large projects and constitute the standard mechanism to submit changes to the Linux kernel.
- Source code for your user-space application you used to exercise your new system call.
- A copy of the output from dmesg on your machine showing the output from your system call.
- Single page summary of the challenges you encountered doing this project. Also, because there are many alternative way to pass data between kernel and user space, I'd like you to investigate

and describe an alternative method you could have used instead of integrating a custom, new system call.

- Please ensure the names of all team members are listed on the summary sheet and/or Canvas project description. Only one copy of the submission tarball needs to be submitted via Canvas, so please designate someone on your team to be the “submitter” for your team.