

CSIS0230A Principles of Operating Systems (Class A)

Tutorial 3

The making of a Linux system call

We will modify the kernel to make our own system call. Our focus is on the process needed to create the system call, so we make our call extremely simple to write:

Specification. Add an initially 0 counter (i.e., a global variable) to the kernel, which can hold a non-negative 31-bit number (i.e., with type `int`). There should be three system calls to manipulate it: `inccount` to increment the count, `deccount` to decrement it, and `getcount` to get the current value. Successful changes to the counter should also return the new value of the counter. If an attempt to modify the counter results in overflow (i.e., the resulting value would be negative), the system call should **not** modify the counter, but instead generating an error `EINVAL` (Invalid argument).

It is usually a good idea to **start with a kernel tree that has been successfully compiled**, as that makes sure that if something went wrong it is due to our modifications. Compiling a kernel the first time **normally** involves copying `/usr/src/linux-2.4.18-3` to your own directory, `cd` to that directory (called “top-level kernel directory”), copying one of the config files in the `configs` directory (usually `kernel-2.4.18-i686.config`) to `.config` in the top-level kernel directory, and edit the `Makefile` there to change the `EXTRAVERSION` as desired. Then one would type `make oldconfig` (or `make menuconfig` if configuration needs to be changed), `make dep`, and `make bzImage modules` to build a new kernel. But that will **take a long time (around 30 minutes in our computers) to complete**.

So instead, all these routine steps are done in advance, and the result is archived in the file `linux-2.4.18-3.tar.gz` for you. Follow the steps below to proceed:

1. Use `tar xzvf linux-2.4.18-3.tar.gz` to extract the files in the archive. You will find a new directory `linux-2.4.18-3` that contains the kernel source tree, compiled in advance.
2. Create a new file `kernel/counter.c` in the kernel directory. Edit the file to implement the system calls as defined above, in three functions `sys_inccount`, `sys_deccount` and `sys_getcount` (as a convention, system calls with name `XXX` is implemented in a function named `sys_XXX`). Don't forget to include the `<linux/kernel.h>` and `<linux/errno.h>` headers, as the `asm linkage` macro and the `EINVAL` constant are defined there.
3. Modify `kernel/Makefile` to add `counter.o` as an object file to build (in the `obj-y` list).
4. Modify `arch/i386/kernel/entry.S` to add three new entries at the **end** of the system call table. Remember the system call number of each system call you added.
5. Change to the top-level kernel directory, and type `make bzImage` to recompile the kernel. This step is relatively quick, because files that are not modified are not compiled again (thank to the `Makefile` mechanism).
6. A new kernel can be found in `arch/i386/boot/bzImage`. To make it a bit easier to boot the kernel, **copy it to the home directory**. We will **not** install that kernel as a default kernel to boot, since the kernel is only for experiment. (To install it we will type `make modules_install` to install the modules, then copy the kernel file to `/boot`, and finally modify the GRUB menu list in `/boot/grub/menu.lst` to boot the new kernel).

7. Switch to the text console using Control-Alt-F1, and reboot using Control-Alt-Del. When the GRUB screen appears, type `e` (edit boot entry).
8. The second line specifies what kernel to boot. Use up and down arrows to move to that line, modify it by typing `e`, move to the place of the original kernel file, and replace it with `/home/os/bzImage`. Type `Enter`, and then `b` to boot your kernel.
9. After logging in again, **write a user program that calls your new system calls**, in the sequence *inccount*, *getcount*, *deccount*, *deccount* again, and finally *getcount* again.
10. If the results are not what you expect, debug it, and repeat the relevant parts of the above steps.

Hint: You might find it beneficial to use *printk* to print things for debugging. However, the output will only appear in the console, so if you need to do this, try to write your program in text-mode.

Happy kernel hacking!