Teaching how to write device drivers

Driving Passion

maddog considers the benefits of teaching students how to write device drivers.

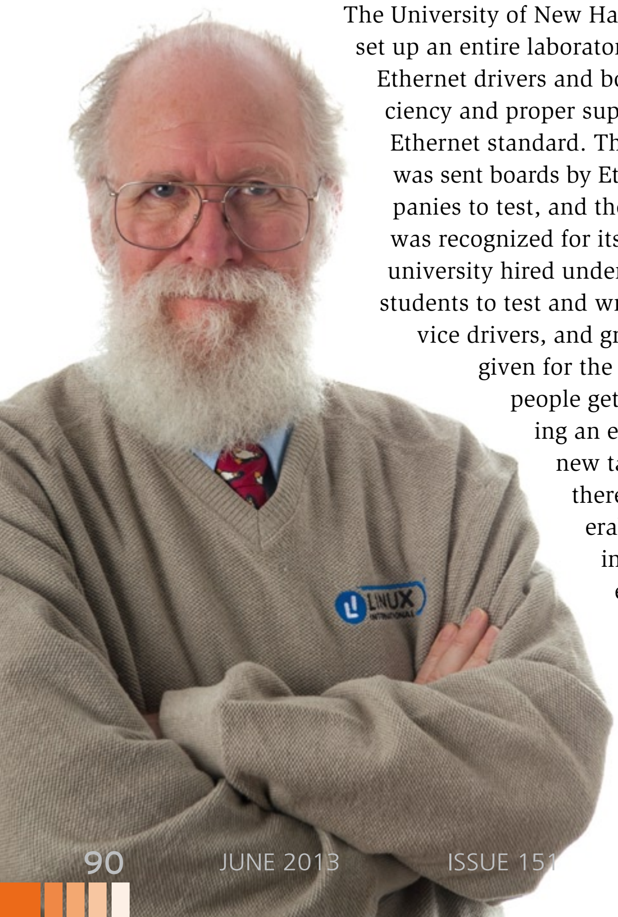
By Jon "maddog" Hall

Recently, I was working with an "upstream" developer of Linux kernel device drivers. This particular developer was working in the media subsystem section of the kernel, where there are a LOT of device drivers because the media subsystem is either the second or third largest driver subsystem in the kernel. The developer reviewed about 3,000 patches per kernel cycle in addition to doing about 100 patches per kernel cycle himself. He was not complaining, just letting me know the work load.

In the midst of our discussions, he lamented:

> *Not actually knowing why Academia doesn't contribute upstream, I would guess that writing a new driver is probably a too short project for them, as students are supposed to work on a 6-month to 2-year window, while writing a new driver takes only a few days with the proper datasheets and/or a reference driver and a knowledgeable developer.*

Perhaps the issue has not been proposed to academia in the correct way. While such a project might not be considered suitable for a Master's or PhD-level course, exactly this level of coding would be appropriate for an undergraduate course in operating systems design, or a course in device driver writing for a technical college or technical high school.



The University of New Hampshire once set up an entire laboratory for testing Ethernet drivers and boards for efficiency and proper support of the Ethernet standard. The university was sent boards by Ethernet companies to test, and the university was recognized for its work. The university hired undergraduate students to test and write the device drivers, and grades were given for the work.Few people get jobs designing an entire OS for a new target, but there are considerable job openings for talented device driver writers, particularly for embedded systems.

With the professor acting as a guide and first reviewer, you might get seven or eight device drivers from each class, while the class would learn several useful things, such as:

- How to write a device driver
- How to test
- How to update another person's code so it looks like one author wrote it
- How to collaborate
- How to review another programmer's work

This class could cover media devices as well as other USB devices. With USB 3.0 becoming more and more popular on motherboards (and with a revision of the standard supporting a theoretical 10Gbps due out in 2014), I foresee a lot of devices, including high-speed SSD and even multiple video screens, being USB instead of soldered onto the motherboard or in some type of motherboard slot.

Another reason for university classes to do these types of projects is that universities typically have the equipment to do the testing. My friend, the media subsystem developer, did not have a certain piece of equipment that could tell him how well his device driver was performing because it was too expensive for him to purchase, so he had to "guess" at the performance, given how other software used his driver. Universities often have test equipment on hand as part of their laboratories or will develop new equipment to test what needs to be tested.

When I was taking (and even teaching) operating systems design many years ago, hardware was still very expensive, and source code for the operating system or sample device drivers for "real" operating systems was still out of reach of smaller universities, colleges, and technical high schools. Therefore, we tended to work on "toy" operating systems created for academic use. We knew that the time and effort we put into those classes was of limited use in the real world. If we had known that we might be able to get jobs after graduation aided by the knowledge we had of a commercial operating system, the class would have been that much more interesting.

Fortunately, times have changed. Now, with hardware such as the Raspberry Pi and some inexpensive devices, a student could write a nice device driver for a USB media subsystem device for GNU/Linux and watch the code work its way "upstream," getting feedback not only from their professor, but also from experienced FOSS programmers. Imagine the pride that students could experience when their friends used a device driver they had written, and imagine the nice entry it would make in a portfolio of work to show to a potential employer.

It would also be nice (and drive the concept) if the school doing the work actually received credit for it … but that is the same with all FOSS. ■■■