

Linux is Free?

Thomas E. Besemer

Utter nonsense. I just completed a project doing embedded Linux on a PowerPC based system, and it was an eye opening experience. I've been interested in embedded Linux for a few years now, invested some time to learn the technology, and when the opportunity was presented to me to do a project with Linux from "soup to nuts" I jumped on it. As a consultant, I've spent the last 15 years making my bread and butter doing VxWorks based projects, and relished at the chance to deploy a project with Linux. The founders of the company I went to work for were particularly attracted to Linux due to the pricing structure (at first glance, free), not to mention the numerous benefits of working with Open Source.

First, a kernel was not available for our particular strain of the PowerPC, so we had to investigate our options. We looked at vendor support from several of the established embedded Linux vendors, and their schedules showed availability *after our product was to ship*. In order to accelerate development, we'd have to spend a lot of cash. The group I was working for had considered having me do the port, but it's not my area of expertise, and more so, I had other pressing matters to deal with – our actual application code! Fortunately, I was able to bring in a consulting group I had worked with in the past to do the port to our processor (for a reasonable fee).

Embedded Linux was no longer free. We did get stabilized Linux in the time frames which we required, but we had to rub money on the project.

Our Linux port was stabilized on a vendor supplied Evaluation Platform. Unfortunately, our proprietary hardware did not follow the configuration of the Evaluation Platform, so we had to port. This itself is not an unusual situation for many, even those who use commercial solutions such as VxWorks. The trap we got into was we ported the first release of Linux we had to our hardware, and then each time our consultant group released a new version, we had to do the port again. Developers using commercial solutions such as VxWorks seldom have this problem, as VxWorks is stable, and it's not often they have to deal with new releases from Wind River in the middle of a project. In such, "managing" our embedded Linux environment started to take time. I'd guess that over a two month period, while stabilizing our hardware, and receiving updates to our Linux kernel, 30% of my time was spent "managing Linux." There were at least five updates to the operating system over a two month period, all while we were trying to ensure the stability of our proprietary hardware. At this point, not only had we paid for a kernel port, we now paid in time.

Keep in mind that in the middle of all of this, we are booting Linux over the network, and mounting the Linux Root Filesystem NFS. We are testing our application by putting it in the NFS mounted area, and invoking it from telnet sessions. We're not even in ROM yet, we are feebly running, and in two months, we've got to drag this box up to a potential investor for a demo!

After a couple of months, our hardware is stable, our Linux port is stable, and our application is somewhat operational (which, after all, as my boss often reminded me of, this was the point of the project – make the application run!). And our investor demo looms in a week. If our path had been VxWorks, we'd already be ready for ROM; the Wind River supplied build environment allows you to easily boot over the network, or simply embedded the image you have booted over the network. With Linux, and the NFS Root Filesystem, we had a huge directory structure that needed to be “tweaked” to fit into our 4Mb Flash device. Since we had gone the route of no vendor support, we were on our own.

The Linux build environment, at least for the PowerPC, does support building images suitable for ROM. However, what is not supported is creating the ramdisk image which contains all of the executables in /bin type directories, or /etc and /lib content. Thus, I had to cobble together a set of scripts to do this. These scripts allowed us to pull in a subset of the existing /bin and /sbin type directory binaries, and then through old world craftsmanship (by hand), I established what libraries needed to be included. A couple of weeks at this, and we've got a ROM based image that our application does not run under, and it's slightly bigger than 4Mb. Would vendor support have helped? Most likely, but that comes at a price. We went to 8Mb Flash, and continued to struggle to configure the environment to allow our application to run.

Even at this point, our system did not “auto-boot”; it ran from Flash, but we still had to start it by hand. The next step was configuring system initialization files to start everything up, and make sure the application ran as a daemon. Not a lot of work, but again, I'll compare it to the VxWorks environment; the development environment rolls right into Flash pretty painlessly. We got it going, eventually.

By the way, we missed our demo for the investors.

Other trappings? Well, Linux is a protected namespace, MMU based environment. You can't see your devices from the application level code unless you write device drivers. We did. They took work. In VxWorks, it's a flat model, and often, device drivers are extensions of the application. In our case, I was the “kernel man”, so a lot of this low-level work got serialized through me. Sure, we had examples due to all the Open Source, but we didn't want to give our application away, so had to write modules, which did not link statically with the kernel, in order to retain our proprietary engineering. For some of the chips on our hardware, we had “Linux drivers” from the vendors. These drivers worked under x86, but not PowerPC. Trying to port them was work, as Linux is not Linux (everybody has a different version of the kernel on their desk), and we had endian issues.

Cross tools? I had to build them. Not a big deal, but not for the faint hearted. I was able to find a web link on how to build the PowerPC cross environment with GCC, but ran into hassles on building glibc. Persistence paid off, and in the end, after some time, had stable cross tools. Certainly, these would have come from a vendor, if we had gone the route of vendor support (but vendor support costs money!).

Is embedded Linux viable? You bet. Is it free? No way. If “time to market” matters, you get a lot from companies like Wind River with stable environments, or perhaps the current embedded Linux vendors. Drawn from my long experience with VxWorks, I concluded that we had added 30% to our schedule through going the Linux path, and keep in mind, we spent some cold hard cash in the process to get the kernel stable. Yet now that it’s done, the actual cost of goods is kept down, as there’s no license fee.

The hot spots that the developer should be aware of, should they choose to go their own path? Many, but the key ones are:

1. Basic system configuration, such as getting the cross tools going, and getting their development environment setup, takes time, experience and specialized knowledge.
2. Installing Linux on proprietary hardware takes work, and without knowledge of the kernel environment, can be a tough road to go.
3. Building ROM/Flash based images is not trivial.
4. Understanding of Linux drivers is essential.
5. Evaluate Vendor Support carefully; it could save you a lot of time, and get you running quickly, but make sure you know what you are buying.

Embedded Linux is real, it’s viable, but it’s not free. It’s easy to get side tracked with embedded Linux, and find yourself in the “O.S. Support Business” instead of the “Application Development” business. Would I do it again with Linux? You bet! This time, with eyes wide open.

Author Bio. Thomas E. Besemer is an independent consulting who specializes in Embedded Systems. You may contact him through his web site: <http://www.tbcorp.com>.