About Embedded Linux

Linux is an incredible piece of software. It’s an operating system that’s just as at home running on IBM’s zSeries supercomputers as it is on a cell phone, manufacturing device, network switch, or even cow milking machine. What’s more incredible is that this software is currently maintained by thousands of the best software engineers and it is available for free.

Linux didn’t start as an embedded operating system. Linux was created by a Finnish university student (Linus Torvalds) who was smart enough to make his work available to all, take input from others and, most important, delegate to other talented engineers. As the project grew, it attracted other talented engineers who were able to contribute to Linux, increasing the burgeoning project’s value and visibility and thus bootstrapping a virtuous cycle that continues to this day.

Linux was first written to run on the Intel IA-32 architecture and was first ported to a Motorola processor. The porting process was difficult enough that Linus Torvalds decided to rethink the architecture so that it could be easily ported, creating a clean interface between the processor-dependent parts of the software and those that are architecture independent. This design decision paved the way for Linux to be ported to other processors.

Linux is just a kernel, which by itself isn’t that useful. An embedded Linux system, or any Linux system for that matter, uses software from many other projects in order to provide a complete operating system. The Linux kernel is written largely in C (with some assembler) and uses the GNU tool set, such as make; the GCC compiler; programs that provide an interface to the kernel; and a host of others that you’ll encounter in this book. Much of this software already existed at Linux’s genesis, and, fortunately, much of it was written with portability in mind. The fact that this software could be used on an embedded system or could be modified to make it suitable for embedded deployment contributed greatly to the acceptance of Linux for devices other than desktop machines.

Note Linux exists in no small part because of the GNU (Gnu’s Not Unix) project, which was (and still is) developing an open source implementation of Unix. The GNU project provided a high-quality compiler and command-line make environment along with basic utilities expected on a Unix-like system.

This book takes you through using Linux for your embedded project. Because Linux and its associated projects are open source, you learn how to build everything you need for an embedded project from scratch. The entire Linux environment has advanced to the point that this undertaking is no longer a quixotic exercise; it falls squarely within the reach of any engineer willing to put in a reasonable amount of time and effort. Building a complete Linux system is the best training for creating a small Linux system; as a result, doing so is more than a morale-building exercise.
Why Use Embedded Linux?

Embedded Linux is just like the Linux distributions running on millions of desktops and servers worldwide, but it’s adapted to a specific use case. On desktop and server machines, memory, processor cycles, power consumption, and storage space are limited resources—they just aren’t as limiting as they are for embedded devices. A few extra MB or GB of storage can be nothing but rounding errors when you’re configuring a desktop or server. In the embedded field, resources matter because they drive the unit cost of a device that may be produced in the millions; or the extra memory may require additional batteries, which add weight. A processor with a high clock speed produces heat; some environments have very tight heat budgets, so only so much cooling is available. As such, most of the efforts in embedded programming, if you’re using Linux or some other operating system, focus on making the most with limited resources.

Compared to other embedded operating systems, such as VxWorks, Integrity, and Symbian, Linux isn’t the most svelte option. Some embedded applications use frameworks such as ThreadX for application support; the framework runs directly on the hardware, eschewing an operating system altogether. Other options involve skipping the framework and instead writing code that runs directly on the device’s processor. The biggest difference between using a traditional embedded operating system and Linux is the separation between the kernel and the applications. Under Linux, applications run in an execution context completely separate from the kernel. There’s no way for the application to access memory or resources other than what the kernel allocates. This level of process protection means that a defective program is isolated from kernel and other programs, resulting in a more secure and survivable system. All of this protection comes at a cost.

Despite its increased resource overhead compared to other options, the adoption of Linux continues to increase. That means engineers working on projects consider the increased overhead of Linux worth the additional cost. Granted, in recent years, the costs and power demands of system-on-chip (SOC) processors has decreased to the point that they cost no more than a low-power 8-bit microcontroller from the past, so using a more sophisticated processor is an option when it might not have been before. Many design solutions use off-the-shelf SOC processors and don’t run the leads from chip for the Ethernet, video, or other unused components.

Linux has flourished because it provides capabilities and features that can’t be made available with other embedded solutions. Those capabilities are essential to implementing the ever more sophisticated designs used to differentiate devices in today’s market. The open source nature of Linux means embedded engineers can take advantage of the continual development happening in the open source environment, which happens at a pace that no single software vendor can match.

Technical Reasons to Use Embedded Linux

The technical qualities of Linux drives its adoption. Linux is more than the Linux kernel project. That software is also at the forefront of technical development, meaning that Linux is the right choice for solving today’s technical problems as well as being the choice for the foreseeable future.

For example, an embedded Linux system includes software such as the following:

---

1 I can’t comment about the usefulness or features of ThreadX, but I can attest that the stuffed monkeys given away at every trade show for the last ten years are adorable. This company also sponsored a booth one year that had actors in Star Trek get-ups explaining what ThreadX did; I wasn’t paying attention, but I got another stuffed monkey after the presentation.