Using Linux in Embedded and Real-Time Systems

When you need an embedded operating system, Linux is a good place to start. Here's why.

by Rick Lehrbaum

Intelligent dedicated systems and appliances used in interface, monitoring, communications and control applications increasingly demand the services of a sophisticated, state-of-the-art operating system. Many such systems require advanced capabilities such as high resolution and user-friendly graphical user interfaces (GUIs), TCP/IP connectivity, substitution of reliable (and low-power) flash memory solid-state disk for conventional disk drives, support for 32-bit ultra-high-speed CPUs, the use of large memory arrays, and seemingly infinite capacity storage devices including CD-ROMs and hard disks. This is not the stuff of yesteryear's ``stand-alone'' code, ``roll-your-own'' kernels or ``plain old DOS''. No, those days are gone--forever!

Then, too, consider the rapidly accelerating pace of hardware and chipset innovation, accompanied by extremely rapid obsolescence of the older devices. Combine these two, and you can see why it has become an enormous challenge for commercial RTOS (real-time operating system) vendors to keep up with the constant churning of hardware devices. Supporting the newest devices in a timely manner--even just to stay clear of the unrelenting steamroller of chipset obsolescence--takes a large and constant resource commitment. If it's a struggle for the commercial RTOS vendors to keep up, going it alone by writing stand-alone code or a roll-your-own kernel certainly makes no sense. With the options narrowing, embedded system developers find themselves faced with a dilemma:

- On one hand, today's highly sophisticated and empowered intelligent embedded systems--based on the newest chips and hardware capabilities--demand nothing less than the power, sophistication and currency of support provided by a popular high-end operating system like Windows.
- On the other hand, embedded systems demand extremely high reliability (for non-stop, unattended operation) plus the ability to customize the OS to match an application's unique requirements.

Here's the quandary: general-purpose desktop operating systems (like Windows) aren't well-suited to the unique needs of appliance-like embedded systems. However, commercial RTOS, while designed to satisfy the reliability and configuration flexibility requirements of embedded applications, are increasingly less desirable due to their lack of standardization and their inability to keep pace with the rapid evolution of technology.

The Alternative

Fortunately, an exciting alternative has emerged: open-source Linux. Linux offers powerful and

sophisticated system management facilities, a rich cadre of device support, a superb reputation for reliability and robustness and extensive documentation. Best of all (say system developers), Linux is available with source code at no charge.

Unlike Windows, Linux is inherently modular and can be scaled easily into compact configurations--barely larger than DOS--that can even fit on a single floppy. What's more, since Linux source code is freely available, it's possible to customize the OS according to unique embedded system requirements. It's not surprising, then, that Linux has created a new OS development and support paradigm in which thousands of developers continually contribute to a constantly evolving Linux code base. In addition, dozens of Linux-oriented software companies have sprung up, eager to support the needs of developers building a wide range of applications, ranging from factory automation to intelligent appliances.

Which Linux?

Because Linux is openly and freely available in source form, many variations and configurations are available. There are implementations specifically for ``thin server'' or ``firewall'' applications, small footprint versions and real-time enhanced versions. There are also Linux ports for non-x86 CPUs, including PowerPC, RISC, 68*xxx* and microcontrollers.

How do you decide which distribution to use? First, realize that all Linux distributions are variations on the same theme--that is, they are collections of the same basic components, including the Linux kernel, command shells (command processors) and many common utilities. The differences tend to center around which of the many hundreds of Linux utilities have been included, what extras are included on the distribution CD and how the installation process is managed. Distributions may include either open-source or proprietary software such as, for example, StarOffice. Even though Linux is free, purchasing a ``commercial'' Linux distribution can have many advantages, including development tools, useful utilities and support. The many companies which are now building businesses around distributing and supporting Linux are busy investing in developing tools and services to differentiate their Linux offerings from the rest of the pack. Some of the special capabilities being developed include:

- Installation tools to automate and simplify the process of generating a Linux configuration that is tuned to a specific target's hardware setup. These can save weeks or even months of development effort.
- A variety of Windows-like GUIs that vary in size, appearance, features and capabilities to support a wide range of embedded requirements.
- Support for the specific needs of various embedded and real-time computing platforms and environments (e.g., special CompactPCI system features).

Yet despite this great diversity, all Linux implementations share a common core, including kernel, drivers, several popular GUIs and utilities.

Small-Footprint Linux

For many embedded systems, the main challenge in embedding Linux is to minimize system resource

requirements in order to fit within constraints such as RAM, solid-state disk (SSD), processor speed and power consumption. Embedded operation may require booting from (and fitting within) a DiskOnChip or CompactFlash SSD; booting and running without a display and keyboard (``headless'' operation); or loading the application from a remote device via an Ethernet LAN connection. There are many sources of ready-made, small-footprint Linux. Included among these are a growing number of application-oriented Linux configurations and distributions that are tuned to specific applications. Some examples are routers, firewalls, Internet/network appliances, network servers, gateways, etc. You may also opt to create your own flavor of embedded Linux, starting from a standard distribution and leaving out modules you don't need. Even so, you should consider jump-starting your efforts by beginning with someone else's working configuration, since the source code of their version will be available for that purpose. Best of all, this sort of building on the efforts of others in the Linux community is not only completely legal--it's encouraged!

Real-Time Linux

Many embedded systems require predictable and bounded responses to real-world events. Such ``real-time'' systems include factory automation, data acquisition and control systems, audio/video applications and many other computerized products and devices. The commonly accepted definition of real-time performance is that real-world events must be responded to within a defined, predictable and relatively short time interval. Although Linux is not a real-time operating system (the Linux kernel does not provide the required event prioritization and preemption functions), several add-on options are available that can bring real-time capabilities to Linux-based systems. The most common method is the dual-kernel approach. Using this approach, a general-purpose (non-real-time) OS runs as a task under a real-time kernel. The general-purpose OS provides functions such as disk read/write, LAN/communications, serial/parallel I/O, system initialization, memory management, etc., while the real-time kernel handles real-world event processing. You might think of this as a ``have your cake and eat it too'' strategy, because it can preserve the benefits of a popular general-purpose OS while adding the capabilities of a real-time OS. In the case of Linux, you can retain full compatibility with standard Linux while adding real-time functions in a non-interfering manner.

Of course, you could also dive in and modify Linux to convert it into a real-time operating system, since its source is openly available. But if you do this, you will be faced with the severe disadvantage of having a real-time Linux that can't keep pace, either features-wise or drivers-wise, with mainstream Linux. In short, your customized Linux won't benefit from the continual Linux evolution that results from the pooled efforts of thousands of developers worldwide.

Who Needs Real-Time?

In any case, how many applications actually require real-time enhancements to Linux? Bear in mind that ``real-time'' is a relative, not an absolute, expression. As mentioned before, a real-time system must handle real-world tasks within acceptable--and predictable--time windows. Although CPUs run at ever-increasing speeds (approaching 1GHz), the world around them goes on at a constant speed. Therefore, real-time performance is becoming ever easier to achieve. Back when the ``traditional'' RTOS were first developed, embedded systems depended on 4- and 8-bit CPUs clocked at single-digit

megahertz speeds and running out of kilobytes of RAM. Now, with CPUs speeding along at up to 1GHz and with memories measured in the hundreds of megabytes, real-time performance is becoming less of a concern. The greater concern has become speed to market and sophistication of functionality. Where execution efficiency was the watchword of the CPU-bound past, protocols are the key to the Internet-centric future.

Going Soft

There's a term to describe the use of ordinary operating systems in real-world applications with acceptable results: ``soft real-time''. In many systems, you can ensure that the real-world constraints of your application can be achieved without resorting to using a specialized RTOS. However, this tends to be practical only when required response times are in milliseconds--not microseconds. Assuming that's the case, a minimally configured Linux on a reasonably fast processor (486-133 or faster) without special real-time add-ons may well suit your needs. If soft real-time sounds like what you need, you may want to check out a Linux add-on called Linux-SRT (soft real-time). On the other hand, your system may indeed require microsecond-level response times. In that case, you can either dedicate an inexpensive microcontroller or DSP to handling the time-critical events, or you can use one of several available real-time Linux add-ons (e.g., RTLinux or RTAI).

Embedded and Real-Time Linux Solution Providers

Since Linux is free, how can anyone build a profitable business based on offering commercial Linux distributions? It's a lot like bottled water--basically, what you pay for is services: packaging, delivery, quality assurance, etc. A word of caution: don't assume that every Linux-related program you download from the Web or obtain from a Linux CD can be freely reproduced and incorporated into the devices you develop. Some commercial Linux distributions that target embedded and real-time applications include proprietary third-party tools and utilities that require licensing and royalty payments if you incorporate them into multiple systems. In other words, read the fine print. For now, however, licensed Linux system software is the exception rather than the rule. With the market placing such a high value on Linux and its associated software being open source and royalty-free, most Linux software companies serving the embedded and real-time Linux market have opted to build their businesses based on selling tools, offering engineering services and providing technical support.

Given the strong position of Microsoft Windows in the end-user desktop/laptop market, it's unlikely the ``average'' desktop/laptop PC user will be running Linux any time soon. On the other hand, in embedded and real-time applications where the OS is an underlying and hidden technology supporting appliance-like operation of a non-computer device, several key features of Linux are making it a growing preference among system developers:

- Source is available and free.
- There are no runtime royalties.
- Linux supports a vast array of devices.
- Linux is truly a global standard.
- Linux is sophisticated, efficient, robust, reliable, modular and highly configurable.

Time will tell, but it certainly looks as though Linux has already altered the embedded and real-time operating system landscape in a fundamental and irreversible way. The result? Developers now have greater control over their embedded OS; manufacturers are spared the costs and headaches of software royalties; end users get more value. And the penguins of the South Pole are celebrating.

Resources

Rick Lehrbaum (rick@linuxdevices.com) co-founded Ampro Computers, Inc. in 1983. In 1992, Rick formed the PC/104 Consortium and served as its chairman through January 2000. In October 1999, Rick turned his attention to embedded software, founding LinuxDevices.com--``the Embedded Linux Portal''.