

Embedded Linux

The 2.6 kernel is ideal for specialized devices of all sizes

By Dr. Inder M. Singh

For more than twelve years, the capabilities and popularity of *Linux* have been building exponentially. Started in 1991 as something of a pastime for creator Linus Torvalds, Linux has since become a robust operating system and a worthy competitor to commercial software such as Microsoft's *Windows* family and Sun's *Solaris*.

Much of the attention that Linux has received has been focused on its growing use in servers. However, Linux scales *down* as well as it scales up, and as a result, Linux has become an ideal operating system for a wide variety of systems. Nowhere has this been more evident than the world of *embedded computing*.

In the simplest terms, an *embedded system* is a special-purpose computer that's built in to a larger device to control that device.

Embedded systems are nothing new. In fact, embedded devices have been around since the 1950s — about as long as computers themselves. Compared to traditional computing (say, desktops, servers, and mainframes), the embedded computing universe is vast and diverse, encompassing computers of all sizes, from tiny wristwatch cameras, to personal digital assistants (PDAs), to telecommunications switches with thousands of nodes distributed worldwide.

Today, embedded systems can be readily found in your house (your microwave and TiVo), your workplace (industrial robots, network firewalls and gateways), your car (in your engine), and even on your person (your Blackberry, iPod, and cell phone).

Common to all embedded devices is a microprocessor and software dedicated to a single purpose. Memory requirements and the amount of ROM and RAM for an embedded system vary greatly, depending on the purpose and features of a particular system. Smaller systems with a single objective, for example, usually require less memory. Embedded systems can also be simple, only requiring small microcontrollers, or they can be fairly complex, requiring massive parallel processors with extraordinary amounts of computing power.

In the embedded software market, several commercial off-the-shelf (COTS) operating systems are available, including Wind River's *VxWorks*, LynuxWorks' *LynxOS*, and Green Hills' *Integrity*. Over the past several years, however, none have matched the explosive growth and popularity of embedded Linux. [Last month's "Out in the Open" featured the Magellan *StackTag* RFID reader, one example of an embedded Linux system.]

The Current State of Embedded Linux

According to several market research reports, Linux is the fastest growing operating system for embedded systems and the most popular embedded operating system for new designs. Without exaggeration, Linux will soon dominate the embedded marketplace, as it's an ideal operating system for a wide range of embedded device.

The phenomenal growth of embedded Linux is being driven by its unique and compelling benefits: developers appreciate having access to its source code; the source code is available for free, with no royalties or encumbrances; and there's a growing base of Linux software, both open source and licensed products, that reduce the time and complexity — and therefore the expense — of developing a new embedded system.

Indeed, more and more embedded system developers are turning toward Linux instead of building their software from scratch, because Linux has the drivers and resources needed to build a complete system. You can usually use find a Linux driver for a new device well before it is widely-supported by proprietary software vendors.

For its part, the semiconductor industry, led by Intel, has also played an integral role in the emergence of embedded Linux. Software support is crucial to the success of semiconductor devices aimed at embedded markets, and Linux has provided a common denominator with growing market momentum. The "freedom" and vendor-independence of Linux allows semiconductor companies to support a device directly or to align with one of several embedded Linux vendors. In fact, most new devices are being launched with Linux support already available. In comparison to other proprietary operating systems, Linux supports a wide variety of hardware devices of all kinds.

In addition to scalability, perhaps the greatest stride Linux has made is in its advanced real-time capabilities.

Hard Real-Time vs. Soft Real-Time

Over the past several years, there's been lively debates on whether or not Linux is suitable for *real-time* embedded system applications.

Real-time is the ability of a system to respond to external or clock events within a *bounded period of time*. Many embedded applications require real-time performance, which is provided by traditional real-time operating systems (RTOSs), such as Wind River's VxWorks or LynuxWorks' LynxOS.

Systems where responsiveness is important, but determinism isn't critical are traditionally classified as *soft real-time systems*. Missing a deadline in a soft-real time system is negotiable.

For example, a PDA or typical network device is considered soft real-time, because while fast response is desirable, occasional delays don't cause malfunctions.

Hard real-time systems must respond in time every time, never missing a deadline. Guidance systems and flight control systems are hard-real time systems by necessity.

Standard Linux doesn't have hard real-time capabilities and has traditionally been better suited to applications such as consumer electronics devices where soft real-time performance is acceptable. However, many embedded Linux vendors have adapted Linux to address other real-time requirements. Vendors such as TimeSys and MontaVista have modified the Linux kernel to reduce latencies and support real-time scheduling.

Another approach to real-time Linux is to effectively replace the Linux kernel with a hard real-time kernel that has the same software interfaces as Linux.

For example, LynuxWorks' LynxOS RTOS is compatible with Linux at both the source and binary level. LynxOS is therefore suitable for more complex applications requiring hard real-time, and provides the major benefits of Linux, such as software portability and reuse, although LynxOS is not an open source solution.

The Impact of the Linux 2.6 Kernel

The introduction of the Linux 2.6 kernel earlier this year improved Linux's real-time capabilities and will undoubtedly fuel even greater adoption. The latest version of the kernel is easier to port to new computers, supports large memory models and microcontrollers, and offers a greatly-improved I/O system. But perhaps the most significant enhancement in 2.6 is Linux's improved responsiveness.

Before 2.6, it was necessary to apply special patches to achieve greater responsiveness. Often this required a customer to buy a special Linux implementation from a vendor — essentially a proprietary system, albeit a familiar one. However, with the availability of enhanced real-time performance in the standard Linux 2.6 kernel, customers are no longer required to choose

one of the special configurations available from a specific vendor. The 2.6 kernel takes Linux into the domain traditionally dominated by proprietary RTOSs.

Improved responsiveness in 2.6 is largely due to three significant improvements: a *preemptible* kernel, an efficient scheduler, and enhanced synchronization.

► **A PREEMPTIBLE KERNEL.** Linux has always been similar to most general-purpose operating systems in that its process scheduler is prohibited from running when a process is executing. This has meant that once a task is in a system call, the task controls the processor until the system call returns, no matter how long that might take. While this design is simple to implement, it can delay more important tasks.

The new Linux 2.6 kernel is now preemptible to some degree, making it more responsive and giving designers better control over the timing of events. While 2.6 still cannot be considered a true RTOS, it is certainly less “jumpy” than previous kernels because preemption points have been inserted to enable the scheduler to run and possibly block a current process so that a higher priority process can be scheduled.

The Linux 2.6 kernel can now also be built without a virtual memory system. Software that has to meet deadlines is incompatible with virtual memory demand paging, because slow handling of page faults can ruin responsiveness. Of course, removing virtual memory means it now becomes the software designer's responsibility to ensure that there will always be enough real memory available to meet application demands and get the job done.

► **AN EFFICIENT SCHEDULER.** In addition to being preemptible, the new 2.6 kernel has a new process scheduler that replaces the slow algorithms of earlier kernels. Previously, to decide which task should run next, the scheduler had to look at each ready task and score its relative importance. After all of those computations were made, the task with the highest score would be chosen. Worse, the time required for this algorithm varied with the number of tasks, causing complex multitasking applications to suffer from slow scheduling.

In Linux 2.6, the scheduler no longer scans all tasks each and every time. Instead, when a task becomes ready to run, it is sorted into position in a queue called the *current queue*. Therefore, scheduling is done in a constant amount of time, because the scheduler only has to choose the task at the most favorable position in the queue. When the task is actually running, it is given a specific period of time it may use the processor before it has to proceed to another thread. When its time period expires, the task is moved to another queue, called the *expired queue*, again sorted according to its priority.

Eventually, all tasks on the current queue execute and move to

the expired queue. At that time, the expired queue becomes the current queue and the empty queue becomes the expired queue. Since the tasks on the new current queue have already been sorted, the scheduler can once again resume its simple algorithm of selecting the first task from the current queue. Not only is this procedure substantially faster than in the past, but it also works equally well whether there are many tasks or just a few. *Figure One* compares the response times between Linux 2.4.18 and Linux 2.6.

► **SYNCHRONIZATION.** Oftentimes, multiprocessing applications need to share resources, such as shared memory or shared devices. Software designers are able to avoid race conditions by using a *mutex* to ensure that only one task is using the resource at a time. Until now, the Linux implementation of mutex has always involved a system call to the kernel to determine whether to block the thread or allow it to continue. The time-consuming system call was often unwarranted and unnecessary.

The new implementation in Linux 2.6 supports fast *user-space mutexes*, which check user space to see if blocking is necessary and only perform a system call when blocking the thread is required. When blocking isn't required, avoiding the unneeded system call saves time. The user space functions also use scheduling priority to determine which thread is allowed to execute in the case of a conflict.

Beyond the individual technical advancements found in the recent Linux kernel, companies are also leveraging the portability of Linux. Compared to mainstream computing, embedded computing teems with a wide variety of processor architectures and specialized, proprietary operating systems. With Linux as the base platform, developers can reuse more code, speeding and simplifying design and development on new devices.

Additionally, Linux avoids “lock-in,” or being dependent on

DEBUNKING THE POSIX MYTH: POSIX CONFORMANCE VS. POSIX COMPLIANCE

POSIX conformance means that the *POSIX.1* standard is supported in its entirety. In some instances, such as the *lynxWorks lynxOS* real-time operating system, the routines of the *POSIX.1b* and *POSIX.1c* subsets are also supported. *Certified POSIX conformance* exists when an accredited, independent authority certifies conformance.

POSIX conformance is what real-time embedded developers are usually looking for. The *POSIX conformant* approach is the most cost-effective solution for porting existing applications and software tools for faster development and implementation.

POSIX compliance is a less powerful label, and could merely mean that a product provides partial *POSIX* support. If a vendor claims *POSIX compliance*, they are simply making documentation available that identifies which *POSIX* features are supported and which are not.

True *POSIX conformance* enables open standards-based Solaris, Linux, HP-RT, HPUX, and UNIX applications, for example, to be easily migrated to other *POSIX conformant* systems.

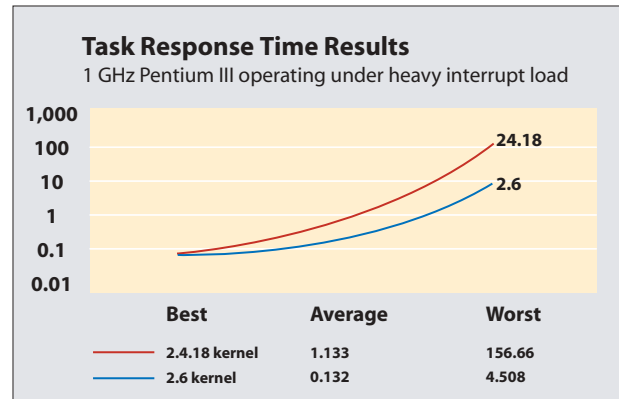


FIGURE ONE: Response times in Linux 2.4 vs. Linux 2.6

a single solution or vendor. Linux code can be moved easily — either to another Linux-based embedded system or to UNIX.

Support of the *POSIX* open standard in embedded systems assures code portability, and is increasingly being mandated in the embedded world, from commercial applications to government projects. Since Linux is 90 percent or more *POSIX*-compliant, the time and effort spent developing an embedded application on Linux can be leveraged into the real-time space with the choice of a Linux-compatible, *POSIX*-conformant RTOS. Hence, embedded developers can turn to Linux 2.6 with the confidence that they can protect the time and investment they have made in developing applications and go real time, in almost no time, should the need arise in the future.

Linux in the Military

Linux and other open standards continue to gain significant momentum, especially in the military, which presents perhaps the greatest growth opportunity for Linux as the move to open standards-based technology ensues.

Just like enterprises have discovered, the military is realizing that closed operating systems are expensive to acquire and maintain. By leveraging the ever-growing world of open standards and open software such as Linux, the military hopes to protect their investments and avoid the high-costs and time-to-market penalties that changing embedded software normally incurs.

For example, the Navy Open Architecture Computing Environment (OACE) has mandated that all future software development be open standards-based. The move to open standards-based operating systems ensures that all future hardware and software upgrades will be seamless, reducing cost and development time for future enhancements to new and unique war-fighting capabilities on ships, aircrafts, submarines, and other platforms.

Currently, testing for full *POSIX* conformance is not widely enforced in the government and military (currently, only the Navy OACE and Allied Standard Avionics Architecture Council test for it), although more government and military agencies are advocating stringent testing within military programs to enforce full *POSIX* conformance.

Embedded Linux Challenges

While the promise of Linux is great for the embedded industry, several challenges still remain, with the most obvious issue being the business model. Small system integrators or software services businesses can do pretty well in the embedded Linux market, but it isn't clear if you can build a large, profitable business purely around an open source embedded Linux operating system and tools. Additionally, several other challenges exist, including security, market fragmentation, and embedded Linux standards.

Just as pervasive computing and network connectivity have been impetuses for the explosive growth of embedded systems, it has also made information technology more susceptible to security attacks. Recent virus attacks have demonstrated that there is a significant need for secure operating systems that cannot be hacked or compromised. The issue is clearly critical for increasingly network-centric defense systems. Achieving the highest level of security for embedded systems is attainable, however, and Linux, POSIX and other open standards-based software are key to enabling the country's future security systems.

Fragmentation remains another challenge for embedded Linux. The open source developer community, the open, multi-vendor nature of embedded Linux, and the wide range of choices available to users are major strengths. On the other hand, they also present the greatest risk, both real and perceived, of divergence and fragmentation. The memory of the "UNIX Wars" is still fresh, and vendors of proprietary systems are threatened enough by Linux to propagate this threat of fragmentation in campaigns of fear, uncertainty, and doubt (FUD).

All embedded Linux distributions start with the same kernel from Linus Torvalds, so there is, in fact, a high level of compatibility between them. However, different vendors add value in their own unique enhancements and subset the system in different ways to reduce footprint. Developers can further modify and configure the system in different ways.

The most exciting aspect of the emergence of embedded Linux is its potential to become the unifying force for this fragmented industry. The embedded industry has never had the equivalent of a software platform like DOS or Windows around which a software industry could grow. Linux provides, for the first time, a common, open, multi-vendor platform around which a software industry can develop.

Standards

For Linux to achieve its potential as the unifying open standard for embedded applications, it's critical to provide assurances that any embedded middleware or application software that follows an appropriate set of guidelines will be supported by all conforming embedded Linux products from any vendor. Historically, embedded developers have had to develop

NO SECURITY IN OBSCURITY

The newest designs for security infrastructure are based on open standards instead of "security through obscurity." In fact, even security requirements are being standardized

Common Criteria (<http://csrc.nist.gov/cc>) is an international framework for developing a set of security requirements for IT products. It defines seven hierarchical assurance levels of security, with *EAL-7* being the highest. Certification to *EAL-7* reflects that a software product has been formally designed, verified, and tested. Today, no commercial, off-the-shelf (COTS) operating system is certified to *EAL-7*.

Companies such as *lynuxWorks* are currently developing a *Common Criteria level EAL-7 separation kernel* in concert with the NSA and others to realize the highest level of security ever. The separation kernel would ensure that any operating system, including Linux and other open standards-based software, could run on top of the separation kernel in its own secure partition in an *EAL-7* system environment with no vulnerabilities.

Most importantly, since the application can run in an open standards-based Linux environment, embedded software tools and applications currently being used, whether in the commercial or government sectors, can be easily ported to an *EAL-7* secure environment.

most of their software from scratch, with far fewer choices of tools and middleware than their mainstream counterparts, and there has been much effort expended reinventing the wheel for each project.

The *Embedded Linux Consortium Platform Specification* (ELCPS) will further reinforce the momentum of Linux as the leading platform based on open standards for the embedded industry, providing a very attractive alternative to closed, proprietary environments from Microsoft and many RTOS vendors.

Also, the *Consumer Electronics Linux Forum* (CELLF) is working on solving the needs of the consumer electronics industry, and the *Open Source Development Labs* (OSDL) has enabled Linux to address high availability and other requirements of the communications infrastructure with *Carrier Grade Linux* (CGL).

Nowhere To Go But Up

Linux has captured the attention of the embedded community like nothing ever seen before. Without a doubt, the growth of embedded Linux has only begun. It won't be long before Linux becomes the operating environment of choice in the embedded market.

Dr. Inder M. Singh is the CEO and chairman of lynuxWorks and board chairman and president of the Embedded Linux Consortium (ELC). Dr. Singh holds Ph.D. and M.Phil. degrees in computer science from Yale University, and an MSEE from Polytechnic Institute of New York.