

# Linux 2.6 for Embedded Systems— Closing in on Real Time

While not yet ready for hard real-time computing, Linux 2.6 introduces many new features that make it an excellent platform for a wide range of embedded computing tasks

by Ravi Gupta, LynuxWorks

**W**ith its low cost, abundant features and inherent openness, Linux provides fertile ground for creativity in embedded computing. As its importance grows, we can even expect Linux to become the platform where progress first happens. The question is, could Linux 2.6 be the breakthrough version we've been anticipating for embedded systems—the version that opens the floodgates to Linux acceptance? The answer is “yes.”

The embedded computing universe is vast and encompasses computers of all sizes, from tiny wristwatch cameras to telecommunications switches with thousands of nodes distributed worldwide. Embedded systems can be simple enough to require only small microcontrollers, or they may require massive parallel processors with prodigious amounts of memory and computing power. Linux 2.6 delivers enhancements to provide support across the spectrum of these embedded needs as well enhancements in the general areas of determinism, reduced memory contention, and the leveraging of POSIX for thread creation and management and task scheduling.

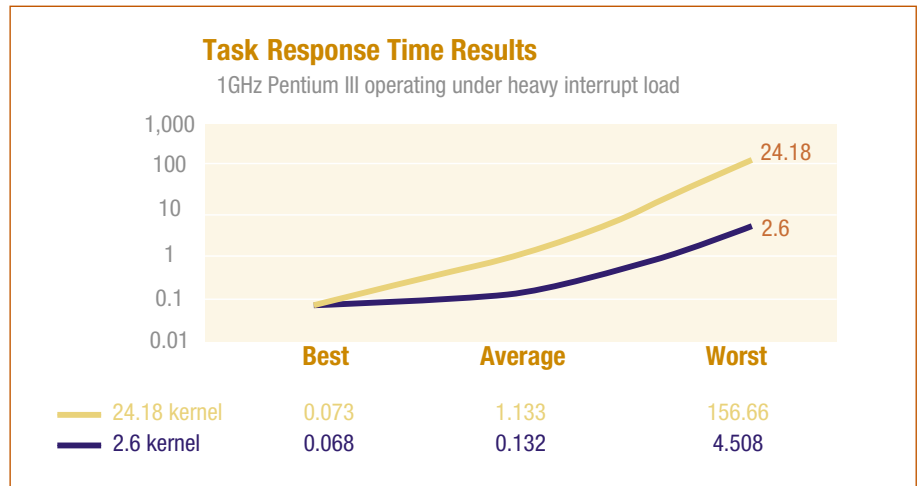


Figure 1 Linux 2.6 shows marked improvement in response time as loads are increased.

Taken together, these enhancements serve to both “firm up” Linux for embedded computing while making it a more attractive alternative for a wider range of embedded applications.

### Firming Things Up

On the “firming up” front, timing constraints are endemic in many embedded applications and must be met

reliably. Linux is not yet a true real-time operating system, yet Linux 2.6 introduces improvements that make it a far more worthy platform than in the past when responsiveness was an issue. The three most significant improvements are:

- Preemption points in the kernel
- An efficient scheduler
- Enhanced synchronization

As with most general-purpose operating systems, Linux has always forbidden a process scheduler from running when a process is executing in a call. Once a task was in a system call, that task controlled the processor until the call returned, regardless of how long it took. This can cause more important tasks to be delayed while waiting for the system call to complete.

With Linux 2.6, the kernel is now preemptible to a degree, making 2.6 more responsive than 2.4 and giving implementors better control over the timing of events. Moreover, in Linux 2.6, kernel code has been salted with preemption points that enable the scheduler to run and possibly block a current process so as to schedule a higher priority process.

The 2.6 kernel can also be built with no virtual memory system, which solves a major sticking point in regard to responsiveness. Software that has to meet deadlines is incompatible with virtual memory demand paging, in which slow handling of page faults can ruin responsiveness. However, getting rid of virtual memory means the developer must ensure that there will always be enough real memory available to meet application demands.

### Efficient Scheduling

Along with becoming somewhat preemptible, Linux 2.6 features a rewritten process scheduler developed to eliminate the slow algorithms of past versions. Previously, the scheduler had to look at each ready task and score its relative importance. The time required for this algorithm varied with the number of tasks and thus complex multitasking applications would suffer from slow scheduling.

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

Eventually, all of the tasks on the current queue will have been executed and

moved to the expired queue. When this happens, the queues are switched: the expired queue becomes the current queue, and the empty current queue becomes the expired queue. As 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. A comparison of test response times between 2.4 and 2.6 is shown in Figure 1.

### Synchronization

Oftentimes multiprocessing applications need to share resources including memory and devices. Programmers are able to avoid race conditions by using a mutex function to ensure that only one task is using the resource at a time. Until now, the Linux mutex implementation has always involved a system call to the kernel to decide whether to block the thread or allow it to continue executing. But if the decision happens to be to continue, why make a time-consuming system call?

The new implementation in Linux 2.6 supports fast user-space mutexes, which check from user space to see if blocking is necessary and only perform the system call when blocking the thread is required. These functions also use scheduling priority to decide which thread gets to execute when there is contention.

### Sharing Memory

Large multiprocessor systems introduce proportionally large contention issues. Embedded systems are sometimes extremely large systems of processors, as in telecom networks or mass storage systems. Here, numerous processors share memory, either as symmetric or loosely coupled multiprocessors. Symmetric multiprocessing designs are inherently limited because the shared memory has to be equally accessible to all processors, making competition for memory the limiting factor in both scalability and processing efficiency.

Linux 2.6 supports a different way of achieving multiprocessing, called Non Uniform Memory Access (NUMA). With NUMA, memory and processors are interconnected, but for each processor some

memory is “close” to that processor and other memory is “farther away.” This means that when memory contention occurs, the nearer processor has superior rights to the memory. The 2.6 kernel provides a set of functions that are aware of the memory/processor topology. The scheduler is able to use this information to favor tasks when they are using local memory, thus ameliorating the memory contention bottleneck and improving throughput.

### Support For Custom Designs

Until now, we have discussed fundamental enhancements that make Linux 2.6 better suited to embedded systems, particularly those systems requiring some degree of determinism and predictable performance. In discussing how Linux 2.6 supports large-scale multiprocessing systems and their applications, we have also started to touch on how 2.6 can meet the needs of a wide spectrum of embedded requirements in terms of system size and configuration.

Pursuing this thread further, hardware designs for embedded systems are often customized for special applications and it is common for designers to need to solve a design issue in an original way. For example, a purpose-built board may use a different IRQ management scheme than a similar reference design. In order to run on the new board, Linux has to be ported, or altered to support the new hardware. This porting is made easier when the operating system is made up of components that are well separated. Hence it is only necessary to change the code that actually needs to change. The components of Linux 2.6 that are likely to be altered for a custom design have been refactored in line with a concept called Subarchitecture.

With Subarchitecture, components are clearly separate and can be individually modified or replaced with minimal impact on other components of the board support package. This means more efficient development and faster time-to-implementation and time-to-market for Linux-based embedded solutions (Figure 2).

### Devices, Buses and I/O

Linux is fast becoming the first choice of operating system for embedded products aimed at consumer markets.

## The Importance of POSIX

There's a lot to be said about the utility of POSIX. For example, the POSIX standard describes a set of functions for thread creation and management called POSIX threads, or pthreads. This functionality has been available in past versions of Linux, but its implementation has been much improved in 2.6. The Native POSIX Thread Library (NPTL) has been shown to be a significant improvement over the older LinuxThreads approach, and even improves other high-performance alternatives that have been available as patches.

Along with POSIX threads, 2.6 provides POSIX signals and POSIX high-resolution timers as part of the mainstream kernel. POSIX signals are an improvement over UNIX-style signals, which were the default in previous Linux releases. Unlike UNIX signals, POSIX signals cannot be lost and can carry information as an argument. Also, POSIX signals can be sent from one POSIX thread to another, rather than only from process to process like UNIX signals.

Embedded systems also often need to poll hardware or do other tasks on a fixed schedule. POSIX timers make it easy to arrange any task to get scheduled periodically. The clock that the timer uses can be set to tick at a rate as fine as one kilohertz, so that software engineers can control the scheduling of tasks with precision.

This is largely due to its cost-effectiveness in what are often low-margin commodity devices. Linux 2.6 delivers support for several technologies that are key to the success of many types of consumer products. For example, 2.6 includes the Advanced Linux Sound Architecture, or ALSA. This state-of-the-art facility supports USB and MIDI devices with fully thread and multiprocessor-safe software. With ALSA, a system can run multiple sound cards, play and record at the same time or mix multiple audio streams.

USB 2.0 also makes its debut on Linux 2.6. We can expect that high-speed devices will proliferate in the near future, and that Linux will be a leading platform for USB 2.0 products.

Video4Linux, the system for supporting video, is also new in Linux 2.6. Although it is not backward compatible with the previous video paradigm, it is intended for the latest generation of radio and TV tuners, video cameras and other multimedia devices.

### Headless Operation

Deeply embedded systems have their own challenge and are often built with no

user interface and sometimes with no operator interface. While previous Linux versions made it possible to build a headless system, some of the support software was not removable, giving the kernel more bulk than was necessary or desirable. Linux 2.6 however can be configured to entirely omit support for unneeded displays, keyboards or mice.

For portable products, Linux 2.6 debuts the Bluetooth wireless interface, which is now taking its place next to 802.11 as a protocol option for wireless communications. With both the SCO datalink for audio and the L2CAP for connection-oriented data transfers available, Linux 2.6 is an excellent choice wherever no-fuss, short-range wireless connectivity is a key requirement.

### Size Matters

On the other end of the spectrum are computers that provide exceptionally large resources, such as very large memory sizes or high-throughput multiprocessors. These heavyweights have numerous embedded applications, such as mass data storage systems and special compute engines. Embedded Linux developers who need

very large memory sizes have their choice of 64-bit microprocessors with Linux 2.6.

The Intel Itanium 64 architecture was treated in a previous releases of Linux, and support continues in 2.6. Linux 2.6 also continues to cover the AMD64 architecture with support of the AMD Opteron microprocessor. Nor is the PowerPC left out, as PPC64 support is also available. Clearly, as 2.6 illustrates, the Linux community has the momentum to keep up with innovations in large-bus, large-memory computing.

Microcontrollers, on the other hand, have been something of a frontier for Linux. Now they are supported on the mainstream Linux 2.6 kernel. In most cases, previous instances of Linux required a full-featured microprocessor with a memory management unit (MMU). But simpler microcontrollers are typically the more appropriate choice when low cost and simplicity are called for.

There have been ways to put Linux on MMU-less processors prior to version 2.6. The Linux for Microcontrollers project has been a successful branch of Linux for some important small systems. Version 2.6 integrates a significant portion of uClinux into the production kernel, bringing microcontroller support into the Linux mainstream.

The Linux 2.6 version supports several current microcontrollers that don't have memory management units. These include Motorola m68k processors such as Dragonball and ColdFire, as well as Hitachi H8/300 and NEC v850 microprocessors. The ETRAX family of networking microcontrollers by Axis Communications is also supported.

As a caveat, Linux running on MMU-less processors will still be multitasking, but will obviously not have the memory protection provided on fully endowed processors. Consistent with the lack of true processes on these small platforms, there is also little in the way of security.

In the area of middleweight systems, sometimes an embedded system may use a conventional Intel-architecture microprocessor, but may need to manage more RAM that can be addressed in the usual 32-bit address space. Intel's Physical Address Extension (PAE) makes it possible for 32-bit computers to access up to 64 Gbytes of memory through page frames. With PAE, Linux 2.6 systems running on newer x86 machines view memory through a movable

“window,” allowing the system to address up to 64 Gbytes of RAM. Linux 2.6 support of PAE is especially useful on systems that perform store-and-forward service for images, video and applications where large datasets must be handled quickly.

### Looking Ahead

Linux is easily the fastest growing operating system in the embedded world and Linux 2.6 will undoubtedly accelerate that growth. Still critical to success of individual embedded projects, however, is expertise in embedded design and Linux/POSIX (see sidebar: “The Importance of POSIX”, p. 33). There is a lot on the way of expert support and guidance that needs to be wrapped around an embedded Linux implementation. But with the proper vendor, this is not so much a challenge as

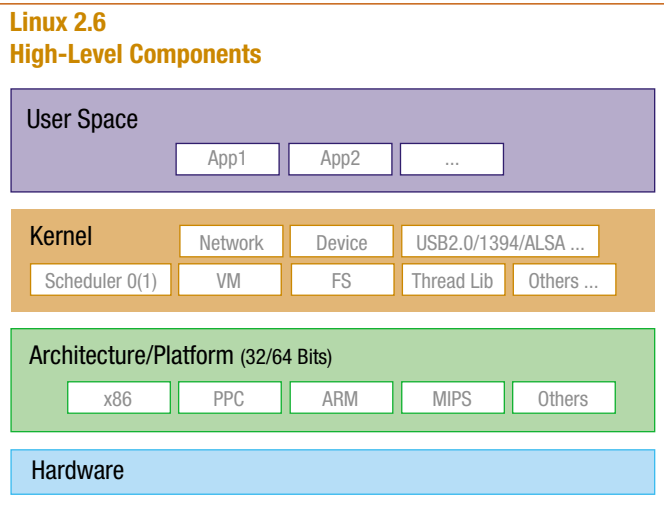


Figure 2 The increased modularity of the Linux 2.6 kernel makes it easier to create custom configurations for specific embedded applications.

it is an opportunity for competitive differentiation and time-to-market advantage for the embedded project.

Developers also need to bear in mind that while Linux 2.6 is not a true real-time operating system, it’s possible to see one from there. That is, should the need arise, 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-compliant RTOS. Hence embedded developers can turn to Linux 2.6 with the confidence that they can go real time, in almost no time. ▲

LynuxWorks  
 San Jose, CA.  
 (408)979-3900.  
 [www.lynuxworks.com].

