News

Magazine
Online Articles
Features
Interviews
Issue
Event Spotlight
Subscription

Advertising
Advertise with Us

Contact Us
Contact Us
Our Location

Publisher
S&S Media Group

Write For Us!
Write for SDA Asia Magazine
>> click here

Online Articles

The Rise of Real-time Linux

Opening a New World of Possibilities with Linux

By Sven-Thorsten Dietrich

New real-time Linux enhancements open a whole new world of possibilities for Linux, ranging across the latest 3G technologies and as near as the mobile handset in your pocket. The purpose of modifying the Linux kernel with real-time functionality: to dramatically reduce interrupt and task preemption latency, thus enabling the 2.6 kernel for use in high-performance multimedia applications and those requiring extremely fast, task level reliable control functions. Real-time Linux has come a long way — where is it now and where is it heading?

Linux has survived many stages of its evolution as a disruptive technology. Today, it is alive, dynamic, and thriving on broad acceptance in a vast, rapidly growing array of application domains.

Reactions to the emergence of Linux have ranged widely, from scepticism to outright hostility. Linux has proliferated within the open source community, and risen over time to become a standard by which many of its predecessors are now measured.

Linux represents a well-tested, and reliable, open operating environment, which can be leveraged to rapidly create technology ranging from simple, battery-powered gadgets to complex, autonomous, AI-integrated motion control systems, like humanoid robots.

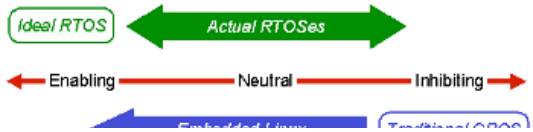
Supply-and-demand economics today validates Linux as the operating system of choice for developers and integrators combined. Developers seek operating systems that allow customisation, provide high reliability, and support application longevity by design. Linux is continuing to meet and exceed these requirements in a growing multitude of applications.

The Rise of Linux

Before Linux became generally accepted as a full-fledged operating system, other factors contributed to its evolution. Open source collaboration is the root of Linux's versatility, reliability and performance. The latter two are the easily quantifiable metrics that ultimately led to Linux's proliferation throughout the technology industry.

Many original Linux installations were Windows PCs, retired for newer, faster models. But the retired CPUs were useful running Linux, due to the efficiency and reliability of the fledgling Linux OS. Linux offered advanced networking capabilities and the applications to use them, long before Windows even supported TCP/IP networking. Linux outperformed the formerly inhabitant Windows OS in uptime and it became the ideal teaching and learning platform.

A majority of the basic design decisions that governed Linux development have favoured fairness, progress and resource sharing. This trinity of design characteristics ensured that even heavily overloaded systems continued to make progress and did not drop network connections or starve applications.



Search
Keywords

Subscribe e-news
Email Address

White Papers
SDA Asia White Papers Library
>> click here

Fig. 1: Dynamics of Real-time Enablement for RTOS, GPOS and Linux

The fairness- and resource-sharing design objectives have contributed to make Linux competitive and popular in the internet infrastructure, enterprise server, as well as application development segments. Companies such as Red Hat, Novell and SUSE have risen with Linux, and are offering competitive Linux solutions that are competing head-to-head with the offerings of internet giants like Microsoft, HP and Sun Microsystems.

Fairness, progress and resource-sharing efficiency are thus essential to the evolution of Linux, and are endemic of the UNIX ecosystem.

The Linux evolution has not ended with commercial viability as an alternative server, or desktop operating system. However, Linux is being swiftly and widely adopted as the embedded platform for innovative internet enabled applications.

In embedded systems design, product feature requirements are broad, system complexity is high, and testing is crucial. Furthermore, overlooked defects in hardware or software can quickly be terminal to a product line.

Embedded systems design pits hardware material cost, power consumption, and performance against one another, in an effort to provide maximum functionality in products competing in high-volume, low-margin markets.

Consequently, embedded devices are typically designed with minimal CPU and memory to reduce cost and power consumption. Embedded devices are highly integrated for compactness. Packed with software functionality, they require a versatile, multi-functional, low-maintenance, real-time operating system (RTOS) for reliable operation.

The OEMs and software vendors who are using Linux to develop embedded systems today are finding that their products are more reliable, versatile, and come to market faster, and at lower cost than competing products. Historically, Linux has fallen short in the time-critical response, or real-time performance category.

Linux and its UNIX legacy are inherently NOT real-time operating environments. Far to the contrary, fairness- and resource-sharing stand in stark contrast to the requirements of embedded or time-critical applications.

Rise of Preemption and Bounded Scheduling
The demands of embedded system applications on the Linux environment have spurred improvements and design work throughout the Linux community.

Memory constraints and complex software requirements in embedded systems, dictate heavy application reliance upon the vast functionality of the Linux kernel and its drivers.

To meet the need for robust time-critical performance in the Linux kernel, MontaVista introduced the O(1) scheduler, along with the “preemptable kernel” concepts in Linux 2.4. Derivatives of these original implementations are found in the community’s 2.6 kernel today.

The new O(1) scheduler in the Linux 2.6 kernel effectively imitates the behaviour of earlier Linux schedulers, while providing bounded-time task scheduling, which is essential for any time-critical applications.

The advent of the 2.6 kernel series brought to mainstream many improvements and features evolved from embedded system requirements, along with the new scheduler and preemption option.

Expectations on the real-time performance of the Linux 2.6 kernel are running high, based on the scheduler and preemption enhancements. Though the scheduler is performing well, response time in the early 2.6 kernel continues to be impacted by delays encountered when running tasks have locked critical resources within the kernel.

Early Linux 2.6 kernels did not generally achieve reliable soft real-time performance, and fell far short of deterministic hard real-time processing. For audio processing, the 2.4 kernel series even outperformed the early 2.6 kernel’s.

Additional improvements were necessary and they were possible.

The broadened versatility of Linux in the embedded sector, combined with the constantly increasing demand for time-critical functionality, required re-examination of the original Linux design principles. Real-time requirements, like VIP treatment, are not generally compatible with fairness- and resource-sharing as in the sense of general admission.

But even today, the emergence and handling of time-critical response characteristics in software design is poorly understood.

Real-time Processing in the Mobile Handset
The mobile handset is representative of the challenges faced by embedded system designers, who are under pressure to squeeze every “bit” of performance out of minimal hardware in the least development time.

The design objectives for mobile handsets are typically governed by market pressure to offer the broadest possible feature set, provide a diverse array of add-ons and user customisation options, and provision the increasingly complex user interface with an easy-to-use, responsive feel.

The phone user is directly affected by the effects of a short battery life, versus the physical burden of carrying a heavy mobile set. Batteries are a major contributor to the weight of a handset, and have to be minimised in size and weight. The mobile handset must operate with only the power in its necessarily small battery, and survive the longest possible interval between recharging cycles.

All devices in the mobile handset, including the CPU, minimise the use of power. The CPU is a costly part, materially, but also with respect to power consumption. In accordance with the battery constraints, the smallest CPU parts available must be chosen for the design to maximise battery life.

The mobile handset designer thus arrives at a conundrum. Choose too large a part, and the cost and power consumption will go up, consequently the profit and battery life will decrease. Choose too small a part, and the feature set will be limited, or the responsiveness of the GUI will suffer, audio will skip, or worse yet, the device might drop calls.

In order to achieve a marketable compromise amongst competing objectives, the CPU must operate as efficiently as possible, and thus at a very high rate of utilisation. High utilisation implies that a large number of tasks could be competing for attention by the microprocessor simultaneously.

High CPU utilisation and numerous tasks, in the presence of time-critical functionality, necessitate the establishment of real-time response specifications. These software timing requirements are commonly referred to as real-time requirements.

In the example of the mobile handset, the must guarantee that the operating system gives precedence to the essential, time-critical tasks in the mobile handset includes:

• Radio communications
• Audio (video) capture and reproduction
• Display output
• GUI responsiveness

The priorities are obvious in the handset application: not dropping a call, or clipping audio in either direction, and not missing updates of text, image or video data on the display. Finally, the user must not wait too long for an expected response after a key is pressed.

Everything else is negotiable. Display of text messages, voicemail notifications, reminders, or an email can all be delayed by as much as a few seconds, without any perceptible performance degradation by the user.

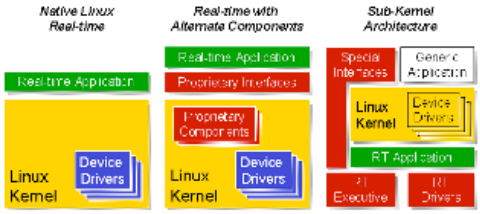


Fig. 2: Native Linux vs. other real-time architectures

The Real-time Linux Kernel
A real-time operating system must recognise assigned task priorities, and respond to time-critical events by switching to the corresponding tasks within a prescribed amount of time. The RTOS will allow all other system tasks to run whenever there are no high-priority events pending, but its performance should not degrade with numerous active tasks.

Preemption
When a time-critical event occurs in a real-time system, the task responsible for processing the time-critical event must be scheduled, after the currently running task has been suspended. Suspending the currently running task is known as preemption.

Preemption can be delayed if the currently running task is accessing shared system data. Shared data must be in a stable state before tasks are switched; since another task (or another CPU) may need to access the same shared data. For this reason, shared data is protected to guarantee that only one task will operate on it at a time. The protection establishes a “critical section”. In the Linux-2.6 kernel, preemption is disabled while a shared data operation is underway in a critical section. When preemption is disabled, the scheduler is not allowed to change tasks, and therefore the running task is guaranteed to have no contention while in the critical section. To further compound the problem, if data was shared with an interrupt handler, interrupts also have to be disabled, along with preemption, rendering the system entirely unresponsive while executing in that critical section.

Efforts had been underway for years, to reduce the number and duration of critical sections in the Linux kernel. Instrumentation was developed by MontaVista to identify the longest critical sections in the 2.4 kernel series, and it is applied to the 2.6 kernel.

There were over 10,000 critical sections in the early 2.6 kernel. Modifying them all to conform to a maximum execution time boundary was a daunting, if not impossible task.

However, there was another way. The critical sections could be protected by an alternative mechanism, known as a Mutex. The Mutex would permit the kernel to allow preemption while executing in most critical sections. Since the kernel was already preemptable, when tasks executed outside the critical sections, Linux would become fully preemptable. Real-time Linux.

MontaVista developed a prototype and the performance was impressive. The complexity of the problem was great and those following the action realised that several others outside MontaVista were working on the same problem. One constraint of the new design was that preemption could not be disabled before a mutex was locked. However, preemption had to be disabled at certain places and mutexes are nested in the kernel. The combination of nesting and disabled preemption regions turned out to be something like a moebius strip. It was clear that some existing code had to be rewritten. The decision was made to go public with the prototype and ask for feedback from the community. In response to the announcement, Linux developers already working on voluntary preemption, spent a week on an epic programming effort. Combined input from those who had formerly been working on the problem independently, completed the conceptual real-time Linux kernel only 6 days after MontaVista announced the prototype.

Development has continued at a frantic pace, and substrate definitions required by the real-time patch began to appear in the Linux release candidates almost immediately. The real-time concept radically changes the operation of the Linux kernel internals. Like the O(1) scheduler, it effectively imitates the behaviour of earlier Linux kernels, and does not alter the kernel's behaviour towards user applications. Most drivers work flawlessly under the new model, and those who fail, seem to deviate from implementation conventions in unusual ways.

Overall, the real-time effort is helping to expose existing problem areas in the kernel, stimulating discussions about efficiency and optimisation, in addition to guiding development efforts towards a continually higher standard of implementation and performance for the evolving Linux ecosystem.

The new real-time enhancements in Linux open a world of new possibilities for Linux, ranging further than Mars Rovers and roaming as near as the mobile telecommunications in our pockets. Real-time requirements will never be alleviated by improvements in hardware performance, since the rapidly growing demands on software applications virtually promise that hardware resources will always be utilised to the limits of their capacities.

Continually, more demanding real-time requirements are guaranteed as a consequence of increased audio and video streaming to and from mobile, and internet connected devices.

Real-time Linux also creates development opportunities for the migration of carrier-grade functionality from plain old telephone switches to cellular base stations, and spanning the gaps between domestic set-top box networks to internet routers. Media systems and data switching carriers using Linux today already have real-time requirements, to reliably process video and audio streams. Similar streaming and bandwidth promise requirements are increasingly imposed on internet routing, cellular and telephony equipment, as the various technologies are integrated into a seamless, media streaming network system. Real-time functionality on the host operating systems and carrier equipment assures seamless integration of the near endless variety of multi-functional clients, routers, and transmission links requiring quality-of-service guarantees.

Sven Thorsten-Dietrich worked for Trimble Navigation for several years, refining his knowledge of GPS and hard real-time systems before accepting a position with MontaVista Software. He has been working in a R&D capacity at MontaVista Software for the last 18 months, focusing on improving the real-time capabilities of the Linux kernel.

Stomfi Would servers and graphics rendering benifit from Real Time Linux?