

Wind River Linux and VxWorks Real-Time Capabilities: A Comparison

Glenn Seiler

General Manager Linux Solutions, Wind River

Table of Contents

Introduction	1
Conditional Real-Time vs. Guaranteed Real-Time	1
Guaranteed Real-Time Application Examples	2
Aerospace and Defense	2
Industrial Control, Instrumentation, and Robotics	2
Telecommunications	2
Mobile Handheld	2
Real-Time Responsiveness: Requirements and Measurement	2
Interrupt Latency	2
Scheduling (or Context Switch) Latency	3
Transforming Linux into a Real-Time Operating System (RTOS)	3
PREEMPT_RT and the Kernel Preemption Patches	3
Wind River Real-Time Core for Wind River Linux and the Real-Time Executive Approach	4
Summary of Real-Time Core for Wind River Linux and PREEMPT_RT	4
The Wind River VxWorks Real-Time Operating System	4
Introduction of the Real-Time Process (RTP) Model	4
Comparisons of Linux and VxWorks	4
Summary	5
For Additional Information	5

Introduction

Wind River has long been the leading provider of real-time solutions to the device software market—first with the market share-leading VxWorks platform, and more recently with its Linux solutions.

With the growing number of product choices within each solution category, it may be useful to compare the advantages and disadvantages of using VxWorks vs. Linux for meeting real-time application requirements. This white paper reexamines the definition of “real-time” in terms of application requirements.

In this paper, we address the differences between “conditional” real-time and “guaranteed” real-time requirements, provide examples of applications that require each type, and compare how VxWorks and Linux achieve real-time responsiveness.

Conditional Real-Time vs. Guaranteed Real-Time

The term “real-time” means different things to different developers. After all, some applications need only an average response time within certain bounds, while others require that every deadline be met every time.

One of the most common ways to define real-time is by distinguishing between conditional real-time and guaranteed real-time. Although those definitions will vary significantly, listed below are general guidelines.

- Conditional real-time provides a guarantee of a specific level of central processing unit (CPU) bandwidth per specific unit of time. For example, an application that has the requirement of 10 percent of overall CPU bandwidth but must have that requirement met within each 100ms unit of elapsed time has a conditional real-time requirement. In other words, the application must be guaranteed 10ms of run-time per 100-ms interval for correct operation, but is otherwise insensitive as to how the 10ms of CPU time is made available in the 100ms unit time.

- Guaranteed real-time is considered a response time rather than a bandwidth guarantee. It is required when an application must respond to an asynchronous event within a specific bounded time to assure correct operation of the application. An example would include responding to a periodic interrupt where the worst-case bounded response time must be less than the interrupt period in order to assure no interrupt events will be lost. A more extreme guaranteed real-time requirement would be a response to an event in a control application such as a vehicle anti-lock braking system, where failure to meet a guaranteed response time could lead to catastrophic results.

Guaranteed real-time is a specialized, non-trivial requirement that may, in some application scenarios, be relaxed to the domain of conditional real-time. A prime example is the rendering of audio in a white-box PC. Essentially at the popular 44.1KHz frame rate, the application requirement is to transfer 32 bits worth of data into hardware each 22.7 μ s. Since other operating system services could delay this operation by being non-preemptive for far longer than 22.7 μ s, buffering is employed to tolerate the preemption delays present in the operating system. Adding a 4096-deep, 32-bit buffer reduces the application timing requirement to essentially filling this buffer approximately each 1/10th second.

Guaranteed Real-Time Application Examples

Many applications have demanding, exacting response-time requirements that cannot be relaxed or accommodated with conditional real-time. Examples of applications that require guaranteed real-time include:

Aerospace and Defense

Aerospace and defense (A&D) applications today include next-generation avionics, command-and-control, navigation, safety, security, signals, targeting, vehicle control, weapons, and other systems. Military, civilian aero-space, and commercial-space-based systems require a robust, high-performance and highly connected operating system, and have some of the most demanding and exacting requirements for embedded platform software.



Industrial Control, Instrumentation, and Robotics

Industrial control demands a mix of high reliability, predictability, and performance. Applications in manufacturing, chemical engineering, power generation, transportation, and other electro-mechanical systems are by definition mission-critical: Failures can cost millions of dollars or even human lives. Whether you are building an assembly-



line robotic arm, a remote-controlled unit, a free-roving wheeled robot, or even a bipedal humanoid model, your project will have exacting performance and responsiveness requirements.

Telecommunications

Modern data and voice applications present a mix of guaranteed and conditional real-time requirements along with the need for ultra-high packet traffic. Voice and other media streams demand a quality of service that surpasses the capabilities of off-the-shelf, general-purpose operating systems. High throughput requirements and heavy traffic can also swamp the simple networking stacks that accompany general-purpose operating systems. One common example in data communications is routing, which requires real-time operation to quickly route thousands of IP packets.



Mobile Handheld

Real-time is one of the key requirements for new single-core cell phone designs. Single-core designs enable cell-phone vendors to lower their costs. However, these designs also create the demanding need to support both the base-band protocol (with its strong real-time requirements) and end-user applications, including streaming media and other applications with real-time requirements.



Real-Time Responsiveness: Requirements and Measurement

The one thing all of the applications above have in common is a requirement for single-microsecond level interrupt and scheduling latency.

Interrupt Latency

Real-time responsiveness is usually measured by interrupt latency. Interrupt latency is defined as the sum of interrupt blocking time during which the kernel is pending to respond to an interrupt. This saves the tasks context, determines the interrupt source, and invokes the interrupt handler.

In reality, interrupt latency is very hardware-dependent; it is based on the speed of the hardware. The real variable in software is the speed of the operating system (OS) to service the interrupt with the Interrupt Service Routine (ISR). But with most real-time applications, before the system can respond to the interrupt, some process, task, or user thread must execute and respond to the interrupt. This is called scheduling or context-switch latency.

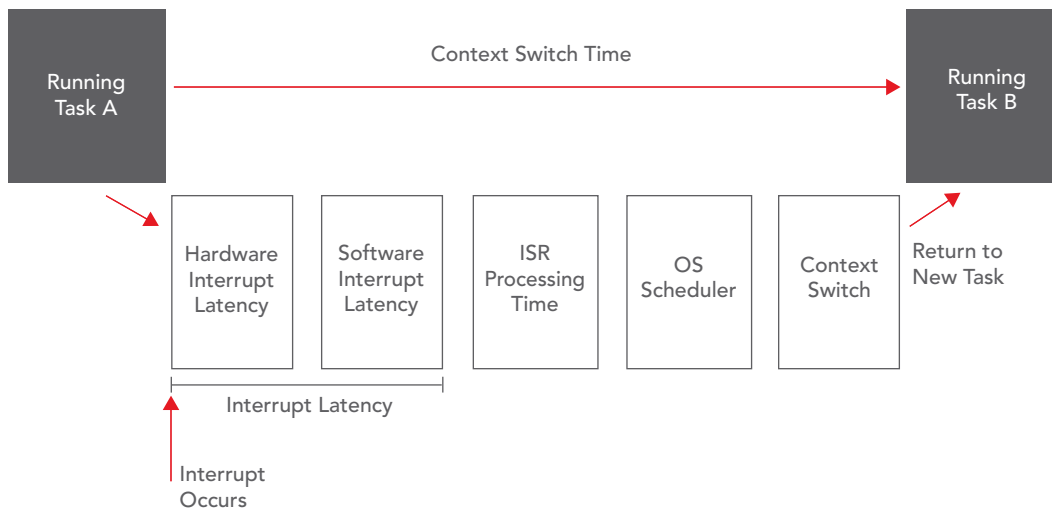


Figure 1: Interrupt and context-switch latency phases

Scheduling (or Content-Switch) Latency

Context-switch latency is the amount of time it takes to switch control to a high-priority process in response to an event that makes it runnable. Part of scheduling latency is the context switch from one process to the higher-priority process. The total real-time responsiveness is the combination of interrupt latency, scheduling, and context switching.

The diagram above illustrates the phases of latency.

There are many methods for achieving real-time responsiveness in operating systems. Real-time operating systems (RTOS), such as the Wind River VxWorks operating system, were designed specifically to solve this problem. VxWorks was designed from the ground up to be an RTOS. On the other hand, Linux was originally designed to be a general-purpose operating system (GPOS).

Transforming Linux into an RTOS

There have been many attempts over the past several years to improve the real-time performance of Linux so that the benefits of Linux, such as open-source software availability, can be leveraged for real-time applications. In this section, we look at the two basic methods of making Linux an RTOS.

PREEMPT_RT and the Kernel Preemption Patches

In the kernel preemption approach, the Linux kernel is modified to reduce the amount of time the kernel spends in non-preemptible sections of code before it can respond to interrupts and subsequent task scheduling operations. This approach requires cooperative modification of the Linux kernel to achieve minimized non-preemptive paths. Therefore, it is an ongoing effort and potentially requires revalidation whenever a kernel modification is made. If code that exceeds the non-preemptible path limit of the kernel is found, it must be restructured to function within the preemption latency goals.

Moreover, the process of finding non-preemptive paths of excessive length is empirical. It also requires exhaustive traversal of all kernel paths under all load conditions in order to guarantee discovery of the worst-case scenario. This is realistically impossible. While competitive preemption latency figures are available in this work, the complexity of the overall kernel makes an absolute guarantee unrealistic.

Many open-source projects have been formed to create a low-latency kernel using a variety of preemption approaches. The most significant project was started by Ingo Molnar and is being mainstreamed into the standard Linux kernel over time. Wind River includes this feature in our Wind River Linux platforms.

There are certainly benefits to this approach; probably the most notable is that the PREEMPT_RT project is widely considered to be mainstream Linux, and the quality and maturity is improving over time. Many code paths in the kernel (such as device drivers) are being modified in main-line to support preemption.

However there are still some limitations with PREEMPT_RT support—particularly in code—receiving less community coverage. This includes architecture-specific code and device drivers.

Therefore, there is variability in the level of performance available among Linux target architectures, and even dependencies per target board are possible. In many cases device drivers will port seamlessly into the PREEMPT_RT model. But preemptive performance, as well as correct operation of legacy and new device drivers, present ongoing validation issues.

Wind River Real-Time Core for Wind River Linux and the Real-Time Executive Approach

With the real-time executive approach, a small real-time kernel coexists with the Linux kernel. Wind River uses this approach in Wind River Real-Time Core for Wind River Linux. Real-Time Core uses a simple real-time executive that runs the non-real-time Linux kernel as its lowest-priority task. It also routes interrupts to the Linux kernel through a virtual interrupt layer. All interrupts are initially handled by Real-Time Core and are passed to standard Linux only when there are no real-time tasks to run. Real-time applications are loaded in kernel space and receive interrupts immediately, resulting in near hardware speeds for interrupt processing.

Wind River Real-Time Core and Linux user tasks communicate through lock-free queues and shared memory in the current system. From the application programmer's point of view, the queues look very much like standard Linux character devices, accessed via Portable Operating System Interface for UNIX (POSIX) read/write/open/ioctl system calls. Shared memory is currently accessed via the POSIX mmap calls.

Real-time tasks are implemented as POSIX threads, using a standard format that does not require user knowledge about Linux kernel modules or features. Therefore, real-time application development is both simplified and nonintrusive on standard Linux operations.

Summary of Real-Time Core for Wind River Linux and PREEMPT_RT

The PREEMPT_RT feature delivers the benefit of being part of the mainstream Linux kernel development and is well suited to certain types of real-time application development such as audio or video streaming. However, the feature is incapable of yielding 100 percent guaranteed real-time performance.

On the other hand, Wind River Real-Time Core for Wind River Linux is not a part of the standard Linux OS and is not licensed by the General Public License (GPL). It is immediately capable of providing a guaranteed real-time environment for appropriately written applications within a Linux environment.

The Wind River VxWorks Real-Time Operating System

Both of the approaches to real-time Linux described above take a GPOS and make it an RTOS. A fundamentally different approach is to add non-real-time services to a basic real-time kernel. This is the approach used by VxWorks.

VxWorks is by far the most widely adopted commercial RTOS in the embedded industry. VxWorks was developed by Wind River with the intention of designing an operating system with fast, efficient, and deterministic context switching. The VxWorks micro-kernel can support preemptive and round-robin scheduling policies, as well as an unlimited number of tasks with as many as 256 priority levels.

VxWorks provides a real-time kernel that interweaves the execution of multiple tasks employing a scheduling algorithm. The VxWorks scheduler determines which task will own the CPU time. By default, the scheduler runs a preemptive algorithm. The preemptive scheduler guarantees that the highest-priority task preempts a lower-priority task. VxWorks uses a single common address space for all tasks, avoiding virtual-to-physical memory mapping.

Introduction of the RTP Model

Historically, VxWorks has offered only kernel-mode tasks, trading off abstraction and protection of kernel components for direct access to hardware, tightly bound interaction with the kernel, and much higher responsiveness and performance. This trade-off requires that kernel development operate at a more sophisticated programming level, where the developer is more aware of subtle interactions with the hardware and the risk of hitting unrecoverable fault conditions.

With VxWorks 6.0 and beyond, Wind River has introduced the real-time process (RTP) model that creates a user and kernel-mode demarcation for VxWorks. The RTP model enables the user-mode application environment without eliminating the traditional model of VxWorks programming. Where available, it also enables memory protection without compromising performance for traditional applications that are part of the kernel space. Application designs based on the RTP model will benefit from the high-performance VxWorks kernel through the addition of reliable resource reclamation and memory protection.

The RTP model also greatly increases the facilities available to the application programmer. User mode applications run in memory contexts even without memory management unit (MMU) support, and run in protected memory contexts where MMU support is available and enabled, benefiting from the protection and reliability this affords.

The RTP model preserves the existing VxWorks values of scalability, performance, and determinism; makes all appropriate parts of the VxWorks kernel-mode API available; greatly improves compliance with industry standards such as POSIX; and provides a uniquely tailored implementation of a time-proven programming paradigm: the process model.

Comparisons of Linux and VxWorks

Wind River Real-Time Core provides real-time in Linux by providing a fairly small real-time "micro-kernel" with its own scheduler. Applications that link into the micro-kernel will get guaranteed real-time for interrupt and context switching. For other services, the application must pass over to a Linux non-real-time task that will do the work. Contrasted with VxWorks, where the whole system is built to be real-time, the application may have more capabilities at its disposal in real-time mode. All of this depends on the independence of the application.

VxWorks still creates a much smaller footprint than Wind River Real-Time Core. Since Linux runs as a low-priority process in Real-Time Core, there are still foot-print constraints on the Real-Time Core solution that do not restrict VxWorks. Linux will never meet the extreme small footprint that VxWorks can meet.

There is also performance degradation for any non-real-time processes that may run in user mode in Linux. VxWorks user-mode RTPs will have much better performance than standard Linux processes. However, they probably have about the same performance as Linux applications running in the Wind River Real-Time Core executive, which runs in the Linux kernel space.

VxWorks determinism and reliability has been proven over decades of use while the PREEMPT_RT patches are relatively immature compared to VxWorks. Another consideration for VxWorks is that it is much more easily certified for military and government applications that require strict certifications; the size of the Linux operating system makes it very difficult and costly to certify. VxWorks was designed from the ground-up for high performance and guaranteed real-time determinism for all applications, while Linux solutions are based on making a general purpose Linux operating system real-time.

When determining the right solution, you must also consider the availability of the third-party software for the application, the overall costs (including developer fees and per unit fees), and the availability of the right hardware support.

Summary

The increasing diversity of real-time development tools and device-application requirements has made life more complicated for developers. Determining whether to use VxWorks or Linux to meet real-time requirements should not be an additional source of stress and risk. Whatever your decision criteria and architectural requirements, Wind River has the right real-time solution to meet your specific application needs.

For Additional Information

For details about Wind River's VxWorks or Wind River Real-Time Core for Linux offerings, visit www.windriver.com.