



## RTOS Roundup

### Ambiguity Abounds in Device Software

This week, Wind River announced their version of a fall fashion lineup for device software and tools. First down the runway, Whitney is looking wild this season in the latest commercial-grade Linux. Whitney's embedded design is royalty free thanks to Linux's open-source roots, but she's happy to have a fully-tested, qualified and supported version from a commercial vendor. Next up, Desiree is decked out in the latest version of VxWorks. Her embedded system design demands reliable response time with no excuses, so she picked a commercial, hard-RTOS to handle her interrupts. Next, turn your attention to Penelope, whose productivity is really being boosted by...

With all the new things going on in embedded device software and support, we thought now would be a good time to pour the RTOS chaos out on a big table, sort out the pieces, and see how the whole thing fits together (or doesn't) for the average system designer. After all, confusion abounds in the RTOS realm. With hard- and soft-RTOS, device software, embedded OS, open-source, commercial, commercial open-source, commercial-grade open-source, royalty-free... there is a confluence of ambiguous labels and categorizations constantly conspiring to confuse us as we seek to select the best OS for our embedded system.

Wind River's announcements may help to tame some modicum of the mayhem. With the company's recently revised direction now coming into solid focus, the leading RTOS vendor is telling us that they'll be doing two major announcements and releases per year now, one in the fall, and one in the spring. They further break down their RTOS offering into two major categories – Commercial Linux and Commercial RTOS.

A few weeks ago, in "The People's RTOS", we took an in-depth look at Wind River's Linux offering, but that's only half of the story. In their most recent announcement, Wind River points out that they also lay claim to a full 42% of the 50% of the RTOS market that makes up commercial RTOS, while their

commercial embedded Linux offering has skyrocketed to a full 39% of the 26% of the market that's using commercial Linux. Competitor Green Hills is happy to help our understanding with the further observation that their market share is growing faster than any other vendor, and analysts chime in that the entire market is now growing at an annual rate of 16-20%. Confused yet? Just whip out a PO and buy one already. The pain won't stop until you do.

However, if we want to delve deeper than market share estimates in making one of our most critical embedded system design decisions, perhaps we should look at the RTOS question from a more technical point of view. Let's pocket that PO for awhile and do a bit more research. RTOS is one of the most over-applied and misused terms floating through the embedded technology space, and it pays to pause and take technical stock of what we're all talking about.

"RTOS" stands for Real Time Operating System, of course, but current common usage has expanded to include pretty much any operating system that isn't running on a traditional, general-purpose computer. Some have migrated to the more general "embedded" operating system terminology, but even the "embedded" concept has fallen out of favor in some circles, in favor of "device" software or OS. Speaking of circles (and maybe in circles), just about every OS runs on a "device" (hey, a supercomputer is a device, right?) so we'll probably just continue to abuse the "RTOS" term until someone comes up with something clearly better.

You can slice and dice the RTOS world several ways, and it's important to understand a few of these. First is the concept of "hard" versus "soft" RTOS. Hard RTOS is where the "Real Time" components are in all capitals. If your application requires a predictable latency (always, really, no kidding, no matter what), and the consequences of missing an event deadline are catastrophic, you're probably in the market for a hard RTOS. If you don't mind if the RTOS you're using takes an occasional brief vacation to manage memory or do some other housekeeping task (as long as it's pretty snappy in interrupts most of the time), you're probably more on the soft RTOS side of things.

Hard RTOS, such as VxWorks (from Wind River), QNX, pSOS, eCos, OSE and LynxOS (which offer a hard RTOS overlay/extension for real-time operation of Linux applications), INTEGRITY (From Green Hills Software), ThreadX (From both Express Logic and Green Hills Software), Microware OS-9 (from RadiSys),

Windows CE, and Nucleus (from Mentor Graphics's/Accelerated Technology), go to great lengths to provide 100% predictable response to interrupts.

Soft RTOS, such as embedded versions of Windows (except CE), Linux, and other modified or ported versions of general-purpose operating systems may have excellent response to interrupts, but do not guarantee that there will never be a longer than expected interrupt latency. It is important to remember the difference between performance and predictability in this case. A hard RTOS may not always have the best performance in terms of interrupt latency, but it will have predictable response so you can reliably plan your system design around it. A soft RTOS, even though it may sometimes have a shorter average interrupt latency, does not absolutely guarantee the timing.

In addition to interrupt latency, it is also important to consider context switch time in evaluating RTOS performance. Context switch time is the amount of time your RTOS takes to save state for one task, load the context for another task, and begin execution. Measurement techniques for context switch time vary significantly, however, as there is no clear standard on what to measure, and also a large variation in performance depending on what amount of state information must be saved.

It also pays to understand the difference between a static and a dynamic RTOS. In a dynamic RTOS, the system can change on-the-fly without stopping and reloading. A dynamic RTOS has a dynamic scheduler that can re-compute task priorities while running depending on the criticality of the task.

Scheduling algorithms used in RTOS vary widely, and a detailed discussion is beyond the scope of this article. However, the selection of the most appropriate scheduling approach depends on the particular requirements of your application. For example, do some tasks have more flexibility in latency than others? Are some tasks regular while others happen very infrequently or irregularly? Are the consequences for missing a deadline dramatically different for different tasks that are being scheduled? All of these considerations come into play in picking the best scheduling scheme, which may affect your choice of RTOS.

Next, we need to look at memory footprint, because embedded OS and RTOS sizes can vary dramatically. Some systems are designed for use in very low-cost, space-constrained portable devices with miserly memory allocations. Others are designed for very large and complex systems, rivaling (and sometimes exceeding) even the complexity of many general-purpose

computers. A few are scalable depending on the complexity of your application and the size of the footprint you can tolerate. You can decide which OS components you need and custom-configure an OS that meets your needs (or is the best compromise) in space, speed, and capability.

One final consideration is the choice of an open-source, commercial-grade open-source, or commercial OS. Open source RTOS (like embedded Linux) exploded in popularity a few years ago due to the lack of licensing and royalty fees, the unlimited customizability, and the easy access to expert developers. Unfortunately, many design teams discovered the ugly side of this tradeoff as well – proliferation of many disparate versions, unlimited customizability, and complete lack of support. These deficiencies led to the creation of “commercial” and “commercial grade” open source RTOS.

Wind River tells us that these commercialized versions of Linux are rapidly gaining acceptance and market share, primarily at the expense of in-house development (based mostly on non-commercial open-source RTOS). Many teams see the best of both worlds in the commercially supported and tested open-source segment.

Interestingly, the commercial RTOS market has largely held its own lately against the open-source incursion. For many companies, the additional level and single point of accountability has high value. For companies involved in high-security or defense-related design, there may be additional requirements (such as mandatory domestic content) that cannot be served by any open-source system. Commercial RTOS are also available on royalty-based and royalty-free variants, and the licensing business model may affect our engineering decision as well.

Speaking of security, all forms of protection are increasing in importance these days. Consumer electronics companies want to protect their valuable IP from unscrupulous copycats. Companies developing networked applications want solid security strategies to reassure customers that their data is safe in the hands of their embedded devices, and military and government applications still see system security as highest priority. Wind River’s recent announcement, for example, prominently mentions a number of security-related enhancements. Security can’t be an afterthought in a new system design, so we should always make a careful assessment of our system’s particular security requirements before we start locking ourselves into a particular system architecture.

So what RTOS will your embedded design be wearing this fall? With the fantastic array of available options, the hardest part is sometimes just choosing one. If nothing grabs your attention, though, don't despair. Spring is just around the corner and we're sure to see a whole new lineup of device software wonderment just in time for the change of season.

*Kevin Morris, Embedded Technology Journal*

*November 8, 2005*