A Conversation with Jim Ready

Linux may well play a significant role in the future of the embedded systems market, where the majority of software is still custom built in-house and no large player has preeminence. The constraints placed on embedded systems are very different from those on the desktop. We caught up with Jim Ready of MontaVista Software to talk about what he sees in the future of Linux as the next embedded operating system (OS).

Ready has a 20-year history with embedded OS development. He founded Ready Systems in 1981 and pioneered the development of one of the first commercially viable, realtime operating system (RTOS) products, the VRTX realtime kernel. After merging with Microtec Research and eventually with Mentor Graphics, Ready began his next push by forming MontaVista Software to capitalize on open-source Unix/Linux and its use for the embedded systems market.

The interviewer is Randy Harr, who, at about the same time Ready was starting his first concern, was embedding data-collection instrumentation into sports equipment that enabled a more fundamental understanding and design of golf, skiing, and bowling equipment. He has been involved in both defense and commercial embedded systems for more than 20 years, culminating in an appointment at the Defense Advanced Research Projects Agency (DARPA) managing embedded digital signal processor (DSP) advanced development. Most recently, Harr cofounded a storage systems company using embedded Linux in a distributed environment for the enterprise.

RANDY HARR How would you characterize an embedded system today?

JIM READY Probably the easiest way to answer that question is in the negative—it's *not* a desktop computer and it's *not* a traditional IT server. Nor is it a general-purpose reprogrammable machine that you and I might recognize as a computer. It is virtually everything else—almost everything we touch, in our cars, in our homes, and elsewhere in our daily lives, including communication devices and everything that has intelligence in it. That's the simple definition for embedded computing today.

Everyone knows about Linux as a

DESKTOP OPERATING

SYSTEM, BUT FEW PEOPLE

KNOW THAT IT IS ALSO

MAKING SIGNIFICANT

INROADS INTO THE

EMBEDDED SPACE.

There are always gray areas, and we'll probably touch on those, but fundamentally embedded systems are where most of the world's microprocessors go.

RH Embedded systems

used to be defined as hav-

ing no graphical user interface (GUI), no visible software, appliances that had, at most, a basic button interface.

They usually had some realtime data acquisition aspects.

Is this still predominant or have other factors become more important as we get into devices such as PDAs and cellphones? Or are PDAs and cellphones even considered embedded systems?

JR The universe has gotten bigger. You described the classic definition that prevailed when we started Ready Systems in 1981 and that has probably been the center of the market until now. Embedded systems were very dedicated devices, with little human intervention or only

buttons at best. Those devices still exist, but clearly what's happened is that the visibility of the user and the user interface in these kinds of systems has gone way up.

Your car, for instance, now has telematics—mapping, navigation, and entertainment systems—that clearly present a very sophisticated user interface. Still, it is not the desktop and not even remotely connected to the desktop. The big change is that there are a lot more interesting systems that are called embedded, but the user—the human interface—is now a critical part of them.

Embedded systems are no longer just button-oriented, with no human interface. The neat part for me now is that embedded systems can consist of Linux applications with beautiful graphical interfaces—that was relatively rare with my previous system, VRTX. Although there were some medical instruments with user interfaces, GUIs in embedded systems are much more common today. Embedded Linux and GUIs are here to stay, it appears.

RH What makes the needs of embedded systems so different? In fact, is there a consistent set of needs for this market?

JR One of the most defining characteristics of embedded systems is the diversity of the underlying microprocessors, completely unlike the desktop market, which is dominated by the Intel architecture.

Today, we find the diversity of processors is as large as it's ever been. In fact, the market is much more diverse than it was even 10 years ago in terms of how many processor architectures are in use. That diversity has a fundamental effect on system software suppliers in the Linux embedded market. For example, we have to maintain Linux across more than eight major architectures and 26 variants of those architectures. If you're going to be in this business in any sustainable way, you need to be able to cover this wide range of hardware very efficiently. Just to compare, if you look at the '80s when we got started, and even into the '90s, the most popular embedded architecture was the Motorola 68K. That is no longer the case.

RH So you don't think there's any single dominant architecture?

JR No. If you look at unit shipments of processors, which a semiconductor company would, I think MIPS and ARM actually dominate because of game and phone applications. But if you look at the number of different designs in development, as a software company does, then you get a much broader picture that includes a broader range

of CPU architectures, including PowerPC, MIPS, ARM, SH, and, of course, embedded IA-32. That breadth of hardware is a factor you have to be able to address as an OS provider.

RH What do you feel are the biggest issues facing embedded systems developers today?

JR In some sense, the key issues haven't changed at all. There's more software to create with less time. We've always been on this curve. It's just that the curve itself is getting steeper: the amount of software you need to get the job done has gone way up. You think about devices today in terms of protocols—communications protocols, multimedia, security. The infrastructure around the systems that you choose is important because you can't write the whole software stack yourself. No one vendor can do it all. It depends upon the availability of many pieces from many sources for a customer to get the job done. The fundamental driver is the size of the software development effort, with software that is increasingly more complex. So how do you solve that problem?

The notion of a highly networked, standard platform is really hot right now. That is what's driving the industry.

RH You touched on it briefly, but can you elaborate on how big and important the standardization of networked embedded systems is today? Is that becoming the predominant factor that's causing complexity?

JR It's a key element. Devices are no longer isolated. They are connected via multiple kinds of media, and there's a greater tendency to have more complex user interfaces.

Next-generation devices allow multiple, independent applications to run, and devices are reprogrammable or field upgradable to add new services. All of these trends imply a move away from the traditional RTOS of the '80s. In other words, in the past we were solving lots of problems, but basically moving toward a more sophisticated basic operating system. The push for multiple applications, networking, a much larger software base, and the requirement for supporting multiple processor architectures bring Linux into play.

A combination of factors has moved the center of gravity and design techniques toward a more sophisticated, more modern OS, such as Linux, and away from the traditional, dedicated RTOS-style architecture. It's just that the average design is now more complex and has a higher networking requirement. That's what's shaping the market.

RH If you look at avionics today, you are down to just a few boxes in the cockpit, each with multifunction capability and multifunction screens and interfaces. How does that dominance of a GUI and multifunction capability affect the realtime integration aspects?

JR We've had a lot of experience in the area of realtime. Those systems are being designed so they fundamentally multiplex the processor in a secure way—i.e., memory partitioned into protected spaces. As far as I understand it, the folks in aerospace and defense have different levels of certification within a single system, in the sense of what runs in memory and how it's protected from other pieces.

In comparison, the older systems were each completely dedicated to a single major function and had multiple boxes performing different functions. To relate it to what we're doing now, it's the utilization of the fundamental processor architectures for protection and so on, to provide an environment that mixes applications. Again, that's a trend away from very dedicated, high-performance RTOS environments toward something more general purpose.

Although flying an airplane is a realtime application, the actual realtime rates are not that high. Primary avionics, at least when I was exposed to it, iterated about 24 or 25 times a second. That is not a great burden on today's processors in terms of how much you could do within each of these intervals.

We, and the Linux community in general, have made efforts to improve Linux's realtime performance, so that, in fact, those kinds of data rates are perfectly within the realm of what Linux can do.

So I think there are a couple of things going on. One is that the requirements and system design, in the case of avionics, are moving toward a greater level of sophistication and utilization of underlying hardware.

Second is the trend to move the system software downward toward the hardware, in the sense of increasing the realtime capabilities of Linux. These two trends are coming together.

With Ready Systems' VRTX, we had a lot of avionics and other realtime design wins. Although it's still early, we are seeing designs in avionics systems as well. Essentially, the industry is the same in the breadth of devices. But the center of gravity of what folks want in an OS has moved toward Linux and away from a traditional RTOS. And that's why I'm in this business, obviously. We saw this happening and started up our business around this trend.

RH It seems we see more of this multiplexing of some sort of wireless network communication or other signal processing with some back-end or user processing. Traditionally in cellphones, the architecture has consisted of two processors: a DSP and a low-end microcontroller. But it seems that that's changing and merging onto a single platform. What trends do you see in mixing high-performance, realtime signal processing with a more appliance-like GUI and back-end functions?

JR Believe it or not—to give people a flavor for what's going on with Linux—our system has been designed into a half-dozen cellphone applications with NEC, Motorola, and other manufacturers. These devices build on either a stand-alone CPU with dedicated baseband hardware or a combination of a traditional high-performance, general-purpose CPU core and a DSP. So the heavy lifting on the signal part still gets done in hardware, which is fine. That's what the [extra] silicon is for. But the human interface and general control of the phone is done on the general-purpose processor.

That is a normal architecture, where you bifurcate things that way. We had a very interesting e-mail from one of our communications customers—this was actually before they were a customer—several years ago. They said that they had historically built their communications systems based on 68K and, in fact, needed a very high-performance RTOS as well, because the processor was actually in the loop in the sense of packet handling. Therefore, every microsecond counted.

They were basically doing all of the in-band (data plane) and the out-of-band (control plane) processing on one CPU, in this case on a Motorola MPC860. What's happened is that the wire speeds have gotten so high that they now do the packet handling and everything else with a dedicated ASIC [application-specific integrated circuit] or network processor. Therefore, the actual load on the control-plane processor in terms of these realtime requirements has become softer, and so we could now afford to have something like Linux running on the control processor with all the benefits that it brings. We always have this great relationship, one way or the other, with the silicon side of things. But clearly the introduction of ASICs or FPGAs [field programmable gate arrays] into the design and the relaxing of requirements on a more general-purpose processor has been a great enabler for Linux.

It's not that realtime requirements have completely gone away but that, in general, the trend is toward more high performance in silicon and softer realtime requirements for the application and embedded OS.

RH Let's shift gears to talk more directly about open source and Linux. One item that you reminded me of earlier was the accelerating Japanese industry adoption of Linux. For example, the Sharp Zaurus is a full Linux system in a PDA. It appears this may open up the market for porting desktop applications to a PDA. Will we start seeing Linux and desktop applications in our TV sets as well?

JR Down the hall, we have a Linux-based PDA that one of our Taiwanese customers has built. The other announcement that we saw recently was the Sony Cocoon at CES [Computer Electronics Show], which is Sony's first element of a whole family of products for the home. It's a personal video recorder (PVR) that sits on the network and is based on MontaVista Linux.

Japan is a very interesting situation. You may not know that Sony, Matsushita, Yamaha, and Toshiba are investors in our company. They've also announced products based on our Linux and so have others in Japan.

You asked: "What's going on in Japan?" The single, most successful RTOS in Japan historically is micro iTRON. This is an indigenous open specification led by Dr. Ken Sakamura of the University of Tokyo. It is an industry standard there. You look at any market survey and see that iTRON represents close to 50 percent of the embedded designs in Japan. Now, there is no one iTRON OS, it turns out. Everybody implemented iTRON in different ways, with modest interoperability. But the principle of having an open specification runtime system is very popular in Japan. It was a VRTX- or VxWorks-like system—very useful, and defined almost 20 years ago.

What's happened since is that the requirements have outstripped what the RTOS business and in-house iTRON code could traditionally provide. In Japan, Linux has become the perfect successor. It meets all the functionality requirements, but it's also very open. No one owns it; everyone owns it. It has many of the psychological and actual real principles and notions surrounding iTRON. It's a very natural upgrade for the Japanese to go from iTRON to Linux.

The good news is that Linux is sufficiently complex so

that having a vendor and their support is important—at least, most folks think so. Therefore, our business in Japan has done extremely well because we support Japan's "iTRON 2"—i.e., Linux. The dynamics of Japan and Linux and MontaVista have come together remarkably well. It's the fastest-growing part of our business.

For similar reasons, the Asia-Pacific region is also a very strong supporter of Linux. Part of Linux's strength in Asia comes from not wanting to hand the keys of the kingdom to a proprietary OS vendor. Neither Sony nor anybody else is going to let Microsoft define what a television, PVR, PDA, or phone is, although Microsoft would love to do it. There's an "ain't going to happen" aspect driving the Linux adoption in the consumer space, meaning people are not going to hand the market over to a sole-source provider. The beneficiary is Linux, and we are a key supplier of embedded Linux. This is a very, very

strong market phenomenon, no question about it.

RH I find it interesting that the Japanese companies can adopt an open-source OS and still use it to help retain their competitiveness on an otherwise closed product?

JR Leading Japanese companies today understand very well how to work with open source and the GNU

General Public License (GPL) and deliver products that take advantage of the best Linux has to offer. They use Linux to enhance their competitive edge, get products to market faster and at lower development costs. And at the end of the day, they also have access to the source code. Now that's a winning combination.

There are well-established methods to allow for retaining your IP on the application side and also on the device driver side. In particular, if you build applications that link only to LGPL libraries (such as glibc) and encapsulate your drivers inside modules, you have a lot of freedom in how you license the code inside your device. On the other hand, Sony and Matsushita recently announced their own, very much open-source initiative, named Linux CE, for consumer devices. As you might expect, we're very close to this work too. The idea is to have a standard platform that benefits the industry, one that offers a "safe" home to CE applications.

In strategic terms, they've publicly laid down the gauntlet. It's going to be Linux from here on out.

RH Could we talk more about what MontaVista means to the Linux community? Maybe you could give me some differentiating points between MontaVista and Red Hat Linux, as an example.

JR Red Hat has a very successful Linux business. I'm not super-familiar with its business models because it's in a different space, but it's a public company and you can look at its numbers. Its fundamental play is in the enterprise server market. It has a big battle on its hands competing head-on with Microsoft and is actually doing pretty well. I hope it does well on the enterprise side.

It has great partners like IBM. It's completely different from the embedded business. Red Hat's public numbers show that virtually all of its revenue comes from the enterprise side. That's where it's putting its focus; that's where its investment goes; and to the extent to which it did have an embedded business, it's declining—and the company knows that, and that's fine. That's a good business decision on Red Hat's part, much like us not being in the enterprise space.

MontaVista is completely focused on the embedded space. The biggest fundamental difference has to do with processors. If you're in the enterprise space, guess what? It's all based on the Intel architecture, so go at it. If you're in the embedded space, it's Intel architecture and everything else, too. So if you look at the MontaVista product line, you will see that it's incredibly rich across all the architectures I mentioned before. If you look at the Red Hat product line, you will see it's completely focused on x86.

So the normal market forces and business forces are aligning the companies quite nicely and in quite complementary ways. Red Hat will take the advanced server into the enterprise space. MontaVista will take Linux as a carrier-grade OS into telecommunications. The Professional Edition, which is our core product, is generally available across many processors and suitable for a wide range of applications.

In January, we announced a Consumer Electronics Edition that is focused on battery-operated consumer devices such as cellphones, PDAs, and other consumer devices such as set-top boxes and PVRs. We thus have a very broad embedded product line from the high-end, carrier-grade systems, down through the consumer device space. For everything in between, which we call A to Z embedded, we provide the standard—the Professional Edition.

In that sense, Linux deploys beautifully from the IT

and enterprise space down to communications servers and, eventually, God only knows what kind of embedded systems such as cellphones, PDAs, and that sort of thing. This is quite a remarkable coverage from one core OS. No other system does that.

RH Traditionally, Linux and the kernel itself have been developed on the Intel x86 platform. It's well known that the early kernel model was very tied to the x86 memory management structures that evolved. Yet aren't a lot of the platforms for embedded Linux today non-x86 architectures?

JR Yes, there's a very interesting quote from Linus [Torvalds, creator of Linux] 12 years ago. He said, "Look, [Linux] is just for the x86. I have no intention of other platforms because I'm just goofing off," and, of course, that's completely not the case today. Now, Linus's work is by no means the only core platform supported—Linus's personal work happens to be typically done on the x86. But now, given SGI, MontaVista, Linus's and everybody else's work, Linux is a completely cross-platform and cross-architecture system. For example, Linux now runs on the Tensilica eXtensa architecture, which is very unique. We see no technical limits with respect to dependence on x86 at all. We have, like I said, eight major architectures, 26 variants, all sorts of different caching arrangements, MMU, TLB structures—all that sort of stuff.

Let's put it this way: If it were a big job to rip out x86 stuff and put in all these other CPU-specific things, we wouldn't be doing what we're doing today. We release an image of Linux and its utilities across all those architectures at the same time, exactly at the same revision level—something no one has ever done in the world before, by the way. And we've done it in a very straightforward, highly automated way. If there were really weird "broken" parts of Linux, it would make that impossible. So we've demonstrated beyond a doubt that Linux is completely portable and architecture-independent.

RH In the early to mid-'90s, there was an explosion of telecommunications providers, and there seemed to be a strong dominance at that time of BSD Unix in their products. The IP stack was fast and robust. Why hasn't that taken hold for the embedded telecommunications sector now rather than Linux?

JR It's interesting. To the extent to which there were gaps, I think those have been closed; and, if you'll notice, things like IPv6 end up showing up in Linux first. So it's

a natural selection in the market. There's nothing wrong with BSD at all. It's another great project. But if you just look at IBM on down, for whatever reason—maybe because of Linux, it's hard to say—the momentum and money and everything else are centered around Linux, so the investment goes there.

Underlying all of this is a fundamental economic situation. Software is not getting any cheaper, and wherever the center of gravity is, as there is with Linux, middleware pieces show up, drivers get written, and so on. It just dramatically changes the economics of using that system versus say a BSD or anything else.

And if you're not dead mainstream, which Linux happens to be, it is more costly to do whatever you're going to do. BSD or another OS may be the right decision for someone, but it's not going to be as cheap as it is to go down the Linux path, just because of the critical mass.

I can tell you, having been in a proprietary OS business, it's the most amazing situation. If you look at the amount of time that we had to spend at Ready Systems on defining everything—every interface, every page in the manual, every API, and every bit of every library—versus how we spend our time at MontaVista, the efficiencies are wildly in our favor. Economically, you can't be on the wrong side of that.

RH Could you talk about what it's like to program Linux as an embedded system versus programming on VRTX. Talk about the applications and how you interact with the system.

JR First of all, just from a programmer's standpoint, as successful as VRTX was and obviously as successful as Wind River's VxWorks is, if you go to any bookstore, you aren't going to find a VRTX book and you're not going to find a VxWorks book, but you're going to find a ton of Linux books.

The good news is that if Linux will work for the application, will meet its requirements, then a lot of folks already know how to program the system, certainly from an application standpoint and, for that matter, device

drivers too. It's extremely well known and well understood.

You could argue, at least from a software engineer's standpoint, there's a high probability they already know how to use the system completely because it's Linux and they have looked at the sources and know how the thing ticks. In many cases the engineers used Linux or Unix during their education.

One of the things that used to drive us crazy in the RTOS business was that customers would take ungodly amounts of time just to decide which RTOS they were going to use. They would actually have to do studies because RTOSs were unknown entities. So they would go get VRTX and pick it apart a little bit. The same thing with

VxWorks or anything else. They wanted to make sure they knew what they were getting, sometimes down to the last bits and bytes.

That's no longer necessary on the Linux side. Folks know in detail exactly what Linux is. There's a whole knowledge base that comes along with Linux, and it greatly accelerates the process. If you'd been nothing but a VxWorks programmer and never touched Unix—which is

unusual, but nevertheless, if you could find someone like that—then obviously you would have a learning curve for Linux. For the vast majority, however, it's the other way around. Linux is well known, and these other systems are obscure by comparison.

The vast majority of our customers build their own hardware so they are porting or doing what we now call an LSP [Linux Support Package], not a BSP [Board Support Package]. They are implementing the adaptation of Linux onto their hardware, a process that is well documented and well understood.

You also have all the Linux device drivers to look at. So it's the same process of integrating the low-level code for device drivers and then synchronization at the application level.

That's all the same, and arguably a Linux application is a more well-defined entity. It's a classic C or classic C++ program, and it exists in a very well-defined programming environment—i.e., the Unix/Linux or POSIX multi-programming model. You have to remember that with

VRTX and VxWorks, that whole area of what a program is and how it links in the namespaces was not real well defined. It kind of grew up by itself, ad hoc.

The definitions of what an application is, its interfaces, IPC mechanisms, and all that stuff, are far richer and more standardized in the case of Linux than they are for what we used to do with VRTX or VxWorks.

But thank God for Intel and everybody else. The underlying processor architectures and performance have grown up so much that, in fact, these applications can support the richer environment that Linux represents. Embedded microprocessors have “grown up” to support an OS such as Linux.

There’s one other thing to keep in mind. A little light went on in my head back in my Ready Systems days. As you got into the RISC processors with caching, you started to have to pay attention to the MMU even if you didn’t use it. When there was a fault, you had to reload pages even though you weren’t doing any mapping. To get VRTX to run on a RISC processor, you had to pay attention at some level to the MMU just as much as Unix or Linux would have to, at least as far as handling faults and making sure the pages were marked correctly for caching and so on.

So you started to see that when something went wrong, you took a jump and you had to handle it. The overhead with respect to the processor was identical whether you were running Linux or running VRTX in terms of what the processor had to do to get the memory stuff right in both cases, but without the benefit of the memory protection you get with Linux. You were also losing the classic determinacy, depending on where you were in memory and whether a page was currently mapped in the TLB or not. There are now variances in execution time depending on the memory layout. Not one customer in the old days was complaining about the lack of microsecond determinacy.

So a little light went on in my head. Although we started out measuring microseconds, very few, if any, applications really, really had those strict requirements because they weren’t going to be able to meet them on highly cached processors anyway. That’s when it became clear to me that the gap between RTOSs and Linux was narrowing because of processor architecture, because of performance, because of customer requirements, and because of the reality about what it took to actually run on those processors.

I thought the time was now right—and this was in 1999 when I started MontaVista—for the industry, in a sense, to absorb what Linux represented. In fact, we’ve

been completely correct in that assumption. We’ve done very well, and we’ve lost next to no business because of determinism issues.

RH If I can try to restate, what you’re basically pointing out is that as the embedded market grew up to use bigger, faster processors that outstripped the old memory hierarchy, you thus lost the realtime capability anyway. So Linux made as much sense as anything else?

JR Absolutely. You could argue, using the pure software argument, that many applications are realtime and need an RTOS. Therefore, there’s no room for Linux. But you can’t implement realtime with the underlying processors of today. They are heavily cached and non-deterministic. You couldn’t possibly use them. Guess what? These processors are used all over the place.

The fact that you can use RISC processors, as they are now very widely used—PowerPC does extremely well on these applications, for example, as does MIPS—already tells you that the transition has been made, quite independently of the software side.

RH How much is Linux’s success and its capability based on the whole gcc [GNU C Compiler] environment and related tools?

JR They’re a matched set: Linux, gcc, and glibc. That whole environment is completely locked at the hip to Linux, and Linux to it. Whatever performance or functionality gaps there were in gcc—if there were any—have largely been filled.

For virtually all of our customers—and we have something on the order of 500 now—it’s a matched set between Linux and gcc, glibc, and g++. For that matter—and my marketing people would probably kill me for saying this—the development environment this provides is simplistic, but it gets the job done. The aggregate benefit to engineers, especially if they have Linux on the host side and target side, is that so much works at such a high level that in the end, these may not be fancy tools but they are incredibly productive ones, and the environment overall is very, very productive.

So right alongside of Linux is gcc, and they’re a matched set.

RH What about Java? It’s obviously had phenomenal growth, possibly even greater than Linux in terms of its compressed timeframe. It had an initial focus in Web ap-

plications, Web servers, and B2B. But it has also been trying to make a big push into the embedded and personal device space. How do you see that going?

JR We actually have a Java product from IBM we work with, add value to, integrate with our Linux, and resell. There's an old Jerry Sanders [founder of AMD] quote back in the early AMD days that "the semiconductor business was a lot like growing asparagus"—that is, it took a couple of years for the payoff. You need to plant the stuff and the first couple of years you don't have much of a harvest. The same is true of Linux. We've been in business for four years, and I think our growth is accelerating. This is a conservative business in some ways; people try you out and then take you to the next step. I think Java, at least from our standpoint, is doing the same thing. We've done some business on Java—not as much as we like, but we're seeing the demand for Java in an increasing number of applications.

It's still early, but this year Java will be a lot better business for us than it was last year. Now that's combining two thoughts: the overall notion of Java and embedded systems, and how we're doing specifically. There are other folks that I think are doing pretty well with embedded Java, of course running on Linux, and we hope to be one of them.

RH A final point to ask is that since you also worked in the traditional embedded market, what mix is there in your customers' current designs? Are people doing more applications on top of your embedded kernel, apart from device drivers that they will always have to write? People used to add whole components into the kernel itself.

JR That's a good point. We saw this, and it always worried me a bit. We'd go visit a VRTX customer, and they'd say, "Hey, look what we did." They had made VRTX into a poor-man's Unix and then had their applications on top of that. There were pieces missing. The alternative was to do it all yourself. So we were a good starting point to simplify it to just adding those missing pieces.

Arguably, Linux is the culmination of this trend. In other words, we're just going to be programming at a much higher level, and we'll just take the services that Linux has and not have to worry about adding them our-

selves because they're all there and we can afford to do this in such a rich environment. Maybe Linux makes this a little easier to do. It's not uncommon to have 10 guys associated with the platform—i.e., doing the heavy lifting and getting the platform stuff running. And then there are another 100 guys behind them who are doing pure application work. That's becoming a little more formalized and a little clearer in people's heads.

We're adjusting our products to have the core Linux for those core developers. Then the things we're doing on the tool side are for the application developers, and they're different folks. The application guys are obviously much farther away from the hardware. In fact, they can

use Linux workstations as simulators. It's the same thing. That's the charm of building embedded systems on Linux.

So maybe some things are a little clearer because you have such a nice, clean separation between the kernel and the application space.

But I have to say at another level—and this

is a positive comment—that it's all the same. I've got any number of folks who are old Ready Systems customers who are now MontaVista customers, who have been doing business with me for almost 20 years. Basically, all we're doing is moving development to a much higher level, but the fundamental drill, although shorter in time, is the same. Customers can just do it a whole lot more efficiently when they're in this next-generation environment, and that's what we've always tried to do: get folks to be able to build their systems quicker.

It's a credit to us in the old RTOS business that the technology we started in '81 has essentially lasted two decades as being the core way things were done. It's only just now that there's a transition to the next generation of Linux.

But 20 years for the same technology, given how much processor technology has changed over that time, says that software is inherently a slower-moving phenomenon. Now I think that 20 years is pretty much up for RTOS technology. And that's why we are where we are with Linux for the next 20 years. I figure if I do this for another 20 years, it will be time to retire, and that's not too bad. Forty years, two systems. Q

© 2003 ACM 1542-7730/03/0400 \$5.00