Open Source vs. Commercial Software

Tom Barrett, CEO, Quadros Systems, Inc.

Open Source. The very term conjures up images of software that is free for the taking. Just download it off the web, compile it and your problems are solved.

Before going further, let me make it clear from the beginning that I am the CEO of a company, Quadros Systems, Inc., that develops commercial real-time operating systems (RTOS) and related products. We come into contact with open source RTOS products quite frequently and have learned a few things about open source software that may be worthy of consideration. At one time, I was even involved with a company (now extinct) that was focused on an open source operating system. Consequently, I have a perspective on open source and commercial products that I hope will prove informational and beneficial.

Above all, I am a realist. I do not take the position that all open source code is bad or should be avoided. To the contrary, some of it can be very useful to embedded system developers. However, open source software is *not* the universal solution for embedded systems development that some of its adherents make it out to be. I believe that there are distinct advantages for using commercially available proprietary software. Let's look at a few.

1. Code size. Open source OSes tend to be large, difficult to scale down and do not give the optimum results that a commercially available RTOS can yield, either in performance or footprint.

Why are the open source products so large? Because open source, while it is a community populated by some very talented software engineers, offers no single champion for underlying principles such as "small" and "efficient." Open source, by its nature, is expansive because it encompasses the market, design and implementation concepts of its developers. The result is quite often a large footprint on the target system. And low cost of memory does not necessarily forgive oversized code. Memory is a recurring cost to an embedded system, so it makes practical sense, from both the engineering standpoint and the economics of the embedded system, to keep memory usage to a minimum. When a product sells millions of units, an increased bill-of-materials cost to accommodate the additional memory of an open source product, will result in higher fixed product costs that may impinge on pricing flexibility of the product.

Commercial products, especially RTOS products, can vary in size from a few kilobytes to a few hundred kilobytes. However, most of them fall into a bracket that is in the sub-100 kilobyte range. And many are much less than that. To give some clarity to the size difference between the open source products and commercial RTOS product, I determined a few years ago that the Linux 2.4 scheduler alone was larger than my entire RTOS in its worst-case configuration. Did that make Linux a bad choice for the customers? Not necessarily. The point is that code size can make a big difference to a product's cost and end-user pricing.

2. One size doesn't fit all. The embedded systems market is not some monolithic market with a fixed set of needs. If it were true, there would be no need for the many commercial software

products that target that large and growing market. The major thrust of popular open source software, such as embedded Linux, does not meet the individualized needs of the fragmented and highly specialized embedded systems market. Proprietary products have focused specifically on providing optimized offerings. In the embedded system market, there are DSP applications and there are control applications. And there are applications where both DSP and control are required. A DSP-centric (or other type of dataflow application) works well with one type of RTOS but not so well with one used for control applications, and vice versa. There are commercial RTOS products that will satisfy all of those needs. Open source products are not typically capable of spanning such a range of application requirements.

And then there is the issue of deterministic design and meeting real-time requirements. Commercially available RTOS products generally tout themselves as being capable of use in hard real-time environments. While it is true that not all embedded systems need to be able to perform to hard real-time requirements, they can still benefit from a commercial RTOS because of the code size, the scheduling options, the efficiency of the code and their reliability. Open source operating system choices certainly exist for these non-real-time applications but their code sizes will be much larger.

3. Tested, tested and tested. When it comes to the operating system in an embedded application, the OS must be the most trusted piece of software in the system. Open source software may have reliable roots, but the constant, community-updated nature of open source software can lead to unproven code. That unproven nature is not conducive to building trust. Proprietary RTOS software, proven in use in billions of devices, must, by its very nature, be reliable, robust and very trustworthy. A commercial RTOS company will not last long shipping buggy software. When it comes to open source or commercial operating systems and the area of trust, on which one are you willing to bet your company’s reputation?

4. Development time. It is quite common today for a developer to deliver multiple end-user products in a single year. As these product life spans decrease, the number of product development cycles tends to increase. The pressure to meet the market window for the new product becomes intense, as any embedded system software engineer will state. Developers get the greatest return when they spend their development time adding value based on their domain-specific knowledge. In most cases, that excludes operating systems and the integration of device drivers, communications stacks and middleware. The developer simply wants to know how to use those packages. Any amount of time spent integrating or learning the inner workings and hidden mechanisms of those packages in order to use them add to the development costs.

Open source products, such as one might find in a complex embedded system, today typically need integration, which usually falls on the shoulders of the system developer. That process adds time to the project and increases risk, two things that every development engineer I’ve ever encountered tries assiduously to avoid.

Many commercial RTOS companies, my own included, can deliver the fundamental value of an RTOS, protocol stacks and middleware, all of which are integrated, ported and running on a target platform with a binding to a specific compiler. That means rapid development with minimal risk. The difference is quite stark.

5. Who to turn to when there is a problem. If you download an open source software package from the web for free, there is a reasonable probability that it is going to require some modification to work on your hardware and in the application itself. So, where do you go for correction and support? You can try a threaded discussion with the open source community and hope someone can figure out your problem. Or perhaps you acquired the product from a company that provides the open source product and sells support for it. That's a fairly common model. It gives you a place to go if trouble or special needs arise. But the reality is that the cost of that support can often exceed the price of a license of a commercial product.

With the commercial product, there is a manufacturer or distributor who can provide the needed assistance. More than likely, the product license included some amount of technical support or additional support is available at a modest fee. Whatever the case, there is a specific phone number or email address or even a person's name that can handle the matter. You don't have to do much more than pick up the phone or write a quick email to connect with that support. It's usually that simple.

6. License issues. Open source products typically make use of the GPL, LGPL and other forms of open source licenses. These licenses can impose severe restrictions on what can be done with the open source code and how your application code makes us of it. Some of those restrictions can even lead to your being legally obligated to expose your intellectual property, such as specific application code, to anyone who wants it, even to competitors. So to protect your IP, you need to hire an IP lawyer to make sure your application's trade secrets stay on the side of the license that protects your interests. But wait a minute, paying lawyers instead of development engineers, can easily make development product costs that much higher.

With commercial software products, there are as many licenses as there are products but they generally fall into two basic forms – the "shrinkwrap" license and the negotiated license. In the former, the licensor grants the licensee a right to use the licensed product in a specific manner. That manner may place limits on the number of projects (products) the licensee can develop, or the number of units produced, or a specific processor, etc, etc. The list could go on far beyond this article's limits. Fortunately, there is usually no need to get a lawyer involved as the shrinkwrap license is normally short and written in terms that are easy for a layperson to understand. This form of license typically does not require a signature, as the license is deemed to be accepted by the licensee when the packaging seal is broken, usually shrinkwrap, hence the name of the form. Thus, the shrinkwrap license saves time and money.

The other basic license form, the negotiated license, often requires the services of a lawyer as it usually deals with very specific needs of both parties. There are special terms and conditions to negotiate and specific legal language to ensure that all parties interests are covered. The process to reach an agreement can take weeks or months. And then the negotiated license will almost certainly require several layers of review and approval before both the developer and the open source vendor sign it. It can be a long and expensive process.

7. Religion vs. Practicality. All too often the concept of open source has become almost a "religion", especially in the operating system area, where much of the dogma stems from opposition to a certain omnipresent operating system. While many have given eloquent voice to

that opposition, blind adherence to that dogma is a poor substitute for making the right, informed choice between an open source and a commercially available product that benefits both the embedded system developer and the project.

8. Does “free” mean no cost? Open source is touted as free, but there are obvious and hidden costs to its use all along the way that can easily exceed the price of a fully licensed, proven and supported commercially available product. I have already listed some of those costs above--HW cost, support cost, legal costs, schedule costs, etc. Be sure you understand the real cost before you start your development.

Summary

Open source certainly is not free, and it may not be the best software for your system. You need to make the software decision that is best for your application and company situation. As you evaluate your options I hope these comments will help you come to the right decision.