expresslogic

The Top 10 Pitfalls of Embedded Open Source Software

While free to use, Open Source Software may pose risks for your product and company's well being

Introduction

Open Source Software (OSS) has been a boon to the computer industry, both at the enterprise level and within the embedded world. The contributions of the OSS community cannot be ignored, with software from Development Tools to Operating Systems to Applications. Generally speaking, this software is free to use under some sort of license such as the Gnu Public License (GPL). OSS is very commonly used in desktop and enterprise environments, and also within the embedded and IoT areas. It is the use of OSS in the embedded and IoT areas that requires some unique cautions for the manufacturers of devices, and these cautions will be the subject of this paper.

In particular, we will focus on a certain kind of Embedded Open Source Software – a Real-Time Operating System (RTOS). An RTOS is used in many embedded/IoT products, including home automation products, wearable devices, medical instruments, and industrial control systems. In these types of products, an RTOS supports application software written by the product developer by enabling multitasking, communication, memory management, and other functionality that benefits the application and makes it easier to develop complex products. When selecting RTOS software for a new project, developers can choose among three general alternatives:

- Develop it in-house (DIY)
- License a Commercial RTOS
- License an OSS RTOS

We will not explore the first option, DIY, which is very rarely employed by today's product developers. DIY generally is limited to relatively simple use cases or legacy systems where migration might require significant effort. While DIY might work fine in some cases, it is more likely that developers will choose between the other 2 alternatives. This paper will focus on the issues surrounding the use of an OSS RTOS in an embedded/IoT commercial product. Many of these issues will not be relevant for non-commercial applications including research, training, prototyping, hobby work, or the like, but the use of an OSS RTOS in a commercial product does introduce these issues.

Here are what we believe to be the Top 10 Pitfalls of using an Open Source RTOS in Commercial Products:

1. Reliability

The reliability of an RTOS is critical to the reliability of the end product in which it is embedded. If the RTOS fails, or behaves unexpectedly, the product itself likely will also fail or behave unexpectedly. This is one reason why RTOSes used in safety-related systems must be certified by regulatory agencies. OSS RTOSes generally are not certified, and therefore are not suitable for use in safety-related systems. But developers of any product should be concerned about the use of an RTOS that is not safety-certified, because while an RTOS malfunction might not lead to injury or death for the user of such a product, an RTOS malfunction will likely lead to that product's failure to perform as intended. At the very least, it likely would lead to customer dissatisfaction, poor sales, and perhaps even a product recall. Safety Certification is a way for any product manufacturer to be confident that the RTOS has been thoroughly reviewed, tested, and proven to perform as intended.

In fact, one popular OSS RTOS - FreeRTOS – had to be re-written to be safe, and is licensed as a commercial variant. "*This variant, SafeRTOS, has the same functional model as the FreeRTOS kernel, but has been rebuilt for safety*." [underline added] (see: http://www.embedded-computing.com/dev-tools-and-os/the-future-of-freertos). What does that imply about the safety and reliability of the OSS FreeRTOS kernel?

Another way to be assured that an RTOS is reliable is to look at where it is being used, and how successful the products are that currently use it. If an RTOS is used in low volume products, or just for research or evaluation, it does not inspire confidence that the RTOS is reliable. It might be, but it is not evidenced by such limited use. On the other hand, an RTOS that is used widely, in many products that are very successful – even market leaders – must be reliable. Aggregate RTOS deployments are a strong indicator of RTOS reliability, ease of use, and overall success.

Generally, for their unproven reliability, and for the reasons we discuss below, OSS RTOSes are not used in high-volume, successful products. No OSS RTOSes even claim deployments of 1 billion units. If they are used in any commercial products, the developers must supplement the RTOS with added debugging and application effort to assure proper operation. Indeed, with enough debugging, an OSS RTOS can be used. But is such extra time, effort, and expense really the best way to make a reliable product, when a proven reliable RTOS is available instead?

2. Security

OSS security is a challenge. Open source code is freely available for anyone to examine and devise a means of subverting. Its popularity is its own undoing, and if it's used in a successful commercial product, the motivation for hackers is heightened, whether for profit from extortion or maximum disruption as a form of cyber terrorism. Also, OSS components might contain security vulnerabilities that could be exploited to threaten any product in which they are used.

For example, mbed TLS is an Open Source product that supports ARM architectures, and is part of ARM's mbedOS. Mbed TLS was formerly known as PolarSSL, an open source product, before it was included into mbedOS by ARM. PolarSSL is an example of an OSS security component with publicly known vulnerabilities:
http://www.securityweek.com/high-severity-vulnerability-found-polarssl-library
https://www.cvedetails.com/product/32568/ARM-Mbed-Tls.html?vendor_id=15698
https://tls.mbed.org/security

Commercial software has a much better chance of avoiding such vulnerabilities, and thus defending against nefarious attacks. Commercial software companies employ qualified, managed, in-house software developers, and use no external software of unknown pedigree (ie: SOUP – see #9a below). This way, the supplier has full control over every line of code in its products, and can much more easily prevent any unintentional, or even intentional introduction of code that would aid a hacker. With an OSS RTOS, there is little known about the authors of the code.

Commercial communication software, including TCP/IP stacks, can be secured by the absence of unintended entry points (a closed system). This makes it impossible for remote takeover or denial of service attack. Furthermore, data security can be assured with IPsec, SSL, TLS, and DTLS protocols which use powerful encryption to prevent unauthorized use of data traffic across networks, including the Internet.

With OSS, there is so much unknown about the code. Is it SOUP that is innocently contributed, but possibly with flaws? Is it contributed by an ill-intentioned party seeking to sabotage a product? The risks are real, and the only recourse for OSS is to rely on community review to catch such code before it makes it into a deployed release. This is a risk, and for many companies, it's too big of a risk to take.

3. Independence

A true OSS RTOS is community-based, and not beholden to any one organization. OSS RTOSes are sometimes supported and sub-licensed by commercial organizations, making them pseudo-commercial offerings. Other OSS RTOSes might be controlled, or "stewarded," by a commercial organization, providing added value for users of that organization's commercial products. It's not clear whether such stewardship includes modification of the OSS RTOS itself, but it seems likely that such is the case. If so, the OSS RTOS takes on a pseudo-commercial aspect, and support must come from the stewarding corporation. As such, it now becomes owned or at least supported by one organization, making it no longer independent and community driven. For example, mbedOS (mentioned above) is available only for ARM processors, so its use is an effective lock-in for ARM. This limits options for future use on a different microprocessor.

Lack of independence creates concern over the use of the RTOS in an environment that is not compatible with the stewarding organization. As another example, now that Amazon AWS is the steward of FreeRTOS, there might be valid concern over using FreeRTOS with Google Cloud or another IoT cloud service.

4. Performance

Depending on the application, the speed with which RTOS services can be performed can make a difference in a product's performance and reliability. Faster services mean lower RTOS overhead, more CPU time available for the application, and a greater margin of safety in demanding environments (eg: rapid interrupts requiring frequent context switching).

The performance of any RTOS can be measured and quantified. Express Logic's "Thread-Metric" benchmark suite (described here: https://rtos.com/wp-content/uploads/2017/10/MeasuringRTOSPerformance.pdf, and downloadable here: https://rtos.com/download/2262) measures the performance of any RTOS in terms of its execution time for each of the basic RTOS services: Interrupt Processing, Preemption, Context Switching, Semaphores, Mutexes, Message Passing, and Memory Management.

By actually measuring the performance of an RTOS – OSS or commercial – developers are better informed regarding that RTOS's ability to meet product performance goals. Most commercial RTOSes publish performance numbers for basic services, or make those numbers available for evaluation. However, developers must be careful to compare apples to apples. Different manufacturers might actually measure different functionality, but refer to it by the same name, which makes a direct comparison misleading. By using the same benchmark code for measurement, this problem is avoided and a true comparison can be made.

Likewise, RTOS code size should be considered. Smaller code size enables use of lower cost microprocessors, less memory, and leaves more room for application code. Code size is simple to measure, and is not as prone to variation as published performance. Still, there are some variables to consider, namely compiler options, the compiler itself, and dependencies. It's best to do a careful size measurement for a specific development environment, using the tools that will be used for actual product development to measure all alternatives.

Most commercial RTOSes have significant performance advantages over OSS - as much as 4x to 8x faster. Similarly with regard to size, commercial cloud connectivity can be provided in less than 30KB, while OSS components can take over 200KB. Many OSS components and products are actually written in C++, but the intended target is a sensor device that requires a small footprint. The use of C++ not only has an adverse impact on the amount of FLASH usage, but it has an even bigger impact on RAM and complexity in general. OSS might not be a good choice for small Cortex-M or other microcontroller-based targets.

5. Lack of Advanced Features

OSS RTOSes tend to be simple, designed to perform basic RTOS services that enable an embedded or IoT device to function. Functionality like Preemptive Scheduling is commonly provided, as well as Interrupt Servicing, Semaphores, Message Passing, and the like. Commercial RTOSes often provide additional features, based on more advanced technology, in part to make these RTOSes more valuable than the competition. Such added value features include Preemption-Threshold™ Scheduling, Integrated Real-Time Trace, Downloadable Application Modules, Memory Protection, Event Chaining, and the like. These advanced features provide additional tools for application developers to employ that make their software operate faster, and make development and debugging easier as well. The result is a more efficient,

higher-performance embedded/IoT product that gets to market faster, and helps the product be more successful throughout its life cycle.

6. Middleware

Most OSS RTOSes consist of just a kernel. While commercial RTOSes might also be described as a kernel, the difference is that commercial RTOSes are surrounded with additional commercial middleware. That middleware might include:

- An embedded File System
- TCP/IP networking stack
- USB Host/Device support
- Graphics framework
- IoT Cloud Service interfaces.

While each of these middleware components might be available for use with an OSS RTOS, they are not always integrated, nor supported by a single responsible organization. That integration, which we refer to as the "Integration Gap," must be bridged by someone – usually the end product developer. That results in additional project time, cost, and risk of error. This is one of the reasons that commercial RTOSes fare better in enabling their users to achieve on-or-ahead-of schedule project completion. This success of commercial RTOSes is evidenced by developer survey data from Embedded Market Forecasters of Natick, Massachusetts (see http://www.newelectronics.co.uk/electronics-technology/an-industrial-grade-rtos-could-get-products-to-market-more-quickly/155137/). A commercial RTOS with middleware generally is pre-integrated and supported by a single entity. That entity has a strong stake in making all components work harmoniously, efficiently, and without conflict in an embedded/IoT device. This minimizes surprises and additional integration work during development, making a timely project completion more likely.

Some of the OSS solutions for the IoT have operated on the premise that the most important thing an application does is talk to the cloud. Accordingly, their emphasis has been cloud connectivity, with little regard for the overall RTOS demands of an IoT device. In reality, an IoT product typically has much more functionality, and cloud connectivity is just one small part of it.

7. Support

Perhaps one of the most significant concerns regarding the use of an OSS RTOS in a commercial product is the issue of support. OSS RTOSes do not have professional, commercial support. They do have a support community that may provide suggestions and guidance regarding questions a developer might have, and those suggestions might be very helpful. Or not. If not, there really isn't anywhere to go for help. Because it is free to download and use, OSS RTOSes cannot finance a support staff that is available for developers. Any support is by virtue of the community's time and effort to help. That's not particularly reliable, and for a commercial development project, it might not be particularly responsive.

A commercial RTOS, on the other hand, generally is available with full commercial support from the authors and maintainers of the product, for a fee. That fee pays for the support staff, and makes it possible for that staff to be responsive to developers' needs. In fact, most such support is contractually bound to be responsive. More than that, the support organization is selfishly

motivated to help developers to be successful, as that leads to continued use of the commercial RTOS and continued support revenue. Commercial support is accountable to the user – something that OSS RTOS community support cannot offer.

There also may be a long term risk associated with OSS. The OSS community generally doesn't guarantee any backward compatibility (in fact FreeRTOS has changed its published APIs in the past), and can change anything based on the community's needs, including the API, licensing terms, etc. Commercial RTOS providers guarantee full backwards compatibility in the API, and the licensing terms are fixed in contract form that cannot be changed unilaterally.

For some application uses, perhaps many such uses, this level of community support is adequate. However, for commercial product development, with competitive time-to-market and internal schedule pressures, an accountable, expert, motivated, and responsive support team is essential.

8. Cost

It's often noted that OSS RTOSes, and OSS in general, is "free." That is true, in most cases, for a license to use the code itself. But that is not the only cost of using OSS. As described above, community support is not satisfactory for a commercial product development project. Hence, some manufacturers will provide in-house support, essentially DIY support. With access to the source code of the OSS, it is indeed possible to put together a staff of support engineers who are experts in the OSS software, and who can be available to support its use in-house. However, such a staff requires training, and must be paid for their time spent supporting users. In some organizations, this overhead is absorbed, but it is still a cost, and likely still budgeted project by project.

Another cost of "free" OSS is integration, as we have discussed above. The Integration Gap requires personnel time, which translates to project cost, time and risk. These costs must be recognized, and ultimately draw on the project budget.

In the following section, we will discuss an additional cost – the cost of legal concerns. These include IPR infringement, and the obligation to freely share proprietary code with the community. These costs can be substantial, and ignoring them can be catastrophic for a commercial enterprise.

9. Legal Concerns

There are multiple legal concerns related to the use of an OSS RTOS in a commercial product. Here are three that are commonly encountered:

a. SOUP

Software of Unknown Pedigree, or SOUP, is a risk. Using SOUP, from a community download, is risky since even one line of code might infringe on an author's Intellectual Property Rights (Copyright or Patent). Even the claim of such infringement might involve substantial legal costs, time, and a resulting ruling to

change the product code to avoid using the infringed code. At the very least, a license fee might be the easiest way forward.

Commercial RTOS software has no such risk. Most commercial RTOSes are licensed with full indemnification against any IPR infringement claims and determination. The licensee (you, the user) has no financial risk – it's all borne by the commercial RTOS licensor. This is a tremendously significant point that must be considered before using SOUP in a commercial product. The more successful the product is, the more lucrative an effort can be to claim that it infringes someone's IPR.

b. Requirement to Disclose Proprietary Code

Some OSS licenses, GPL for example, mandate that users of the OSS must contribute back to the community any software they write that is combined with, or linked with (in some cases) the OSS itself. This means that proprietary application code, the software added value that makes a commercial product competitive, might have to be shared with the public – including your competitors! That's an uncomfortable obligation to undertake, and a good reason not to use OSS in a commercial product. If you do, be advised to carefully review the license agreement to see what you are permitted to do with the licensed software, and what you are obligated to do in return.

c. Product Liability

Laws are being enacted to strengthen product liability responsibility, such as the European Commission's Product Liability Directive. In the USA, the FDA has ruled that medical device manufacturers' CEO's may be liable for product malfunction. Whether or not such laws are in place, the producer of a product should follow "best practices" in the creation of their product in order to maximize safety and defend against claims of negligence on their part. OSS presents a huge problem in this area, because it is hard to argue that using free software off the Internet is best practice.

10. No commercial pressure to make open source better.

OSS is not commercial. That means it is not revenue-generating, while it does require development and maintenance by the community. Commercial software is developed to generate revenue. That is its primary purpose, and often is essential to the developing company's prosperity, or even its survival. The competitive pressure faced by commercial software developers provides selfish motivation to succeed, and that pushes them to make their products better and better over time.

There is no such pressure for OSS. Yes, there are intrepid community developers who truly care about making the OSS better, and there is a mechanism for the community to request new features and to fix certain bugs over others. But in no case is a community member's job on the line, nor is any organization's survival at stake. The motivation is fundamentally different, and significantly stronger for a commercial organization.

Commercial software companies always have competition. If there is no competition, there is no market. In order to survive in a competitive environment, commercial software developers invest in engineering and other staff to work together to find customer needs and to satisfy them in the best way possible. Those companies that do not succeed in this business fail to survive, and end up closing their doors or being absorbed into other companies at low value. Those who survive and control their own destiny invest significant funds toward product improvement and expansion, which they expect to generate revenue in return. This fundamental business dynamic operates to the benefit of the consumer, guaranteeing him or her access to the best products from the best companies that have survived the longest.

Summary

Before using an Open Source RTOS or other software in a commercial product, consider the issues discussed here. It may be that you'll continue to use OSS, if these concerns do not apply to your particular situation. For others, perhaps they do apply. Careful consideration of these issues might convince those companies contemplating use of an OSS RTOS to re-consider and opt instead for a commercial RTOS. They might very well find that a commercial RTOS gives them more predictable cost, lower risk, and more readily available expert, responsive support.