Choosing the Right System Software for Medical Devices

By Stephen Olsen, Product Management, Wind River

WHEN IT MATTERS, IT RUNS ON WIND RIVER

EXECUTIVE SUMMARY

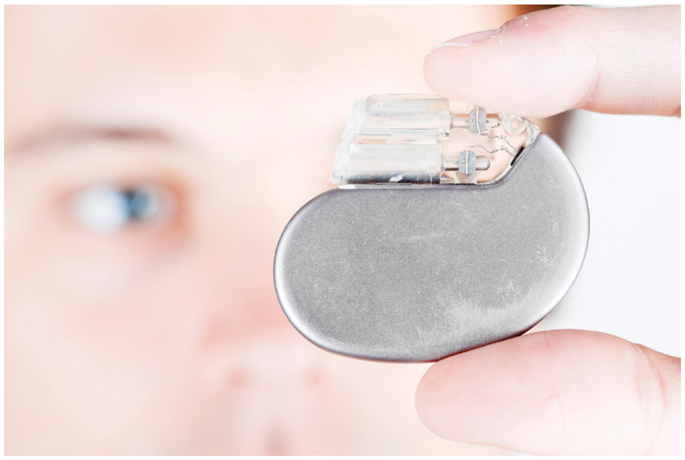
Finding the right software for the design of medical devices is critical, as there is little room for error. Developers have many choices to make, from whether to use a "roll-your-own" solution or a commercial off-the-shelf product to whether to employ a real-time operating system or a general purpose operating system such as Linux or Android. There is no optimal solution for all devices, but understanding the parameters of the application and its interaction with the target hardware will help developers narrow their search. This article explores the breadth of considerations that are essential in making the best choice.

TABLE OF CONTENTS

STARTING POINT: DESIGN CONSIDERATIONS

Several years ago, I had Lasik eye surgery. While the procedure took less than one minute per eye, I found myself wondering about the software that was used to control the laser. For my procedure I had a choice between a manually controlled laser or an automatic tracking laser. I weighed the pros and cons of each scenario, and eventually I opted for the more complex automatic tracking laser. But was it the right choice? What would have happened had I chosen the laser with manual tracking? My sight is better than 20/20 now without glasses, but the surgery got me thinking about the system software that goes into such medical devices.

This example underscores the point that any discussion about software for medical devices should begin with one overarching consideration: Medical devices directly impact human lives. That means every decision that involves the specifications of the device also impacts human lives. That said, modern medical devices come in all shapes and sizes—from very large, elaborate radiation machines to small, implanted pacemakers that are battery operated—and they vary greatly in the criticality of their intended functions. Therefore the first specific consideration in selecting the right system software should be the intended use case.



How will the device be used? Will it be used primarily by a healthcare provider, by the patient at home, or both? Will it collect and store data? Will there be special service modes that allow access only to a trained technician? Will it be wall powered, battery powered, or both? Are there charging cycles? Is the device usable when it's charging?

The answers to these use-case questions will inform and guide the system software evaluation and decision criteria. For example, when systems are small and implantable, the need for real time is usually negligible and the software associated with these devices may be relegated to monitoring a small set of sensors and administering some small amount of therapy. In a maintenance or data collection mode, the device may receive a radio signal from outside the body, and this could be done with a low-power microcontroller that can be built with a simple state machine.

On the other end of the spectrum, systems that employ a general purpose operating system, such as Linux or Android, are typically not concerned with memory and physical size constraints. Imaging

systems that are administering a signal into the body and producing an image from that signal employ a large signal-processing algorithm that needs to be processed in near real time—for example, an ultrasound that is imaging a mother's womb. This is near real-time because the response can be one second delayed from the actual signal, but it must closely correlate to the scan, as the operator will use this feedback to guide the device to give parents peace of mind.

The majority of portable electronic medical devices fall somewhere in between these two examples. These systems are somewhat memory-constrained, battery-operated, and connected devices that may have real-time constraints as well. The combination of such attributes almost necessitates a real-time operating system over a general-purpose operating system.

TRENDS IMPACTING MEDICAL DEVICE DESIGN

In addition to the intended use case, it is important to consider some of the meta-trends in the medical industry. One such trend is that an increasingly broad range of professionals is using a variety of devices to treat their patients. As medical devices become more ubiquitous, with both professionals and non-professionals using them—such as in-home care or nursing home situations—the question is not only whether these devices are capable of doing their job but, even more importantly, whether anyone with a basic understanding can operate such a device. Is the device easy to use in an emergency situation when seconds matter? What happens if the device fails?

When you combine this trend with the skyrocketing costs of healthcare, it's clear the onus is on the embedded software developer to design a complete solution that goes beyond basic device functionality. There's a mandate to minimize cost per capita of each person's healthcare by looking at ways to streamline diagnoses, add innovative ways to administer preventive care, and in the process find new technologies to minimize the costs of these devices as they become more pervasive in our society.

Cost Containment	• Capitation • Insurance Approval • Outpatient Care
Quality of Care	• Improved Diagnoses • Preventive Care • Better Technology
Aging Population	• More Medical Needs • Frequent Monitoring • Self-Diagnosis
Privacy	• HIPAA • Electronic Medical Records

Figure 1: Key healthcare trends and issues

Further, the average lifespan of both men and women is increasing in many countries. With more people living longer, diseases associated with aging are far more prevalent. We need to find innovative and cost-effective ways to monitor, diagnose, and treat these maladies. Embedded medical devices need to safely communicate information from device to device, device to network, or even device to a server in the cloud. To go even further would be to utilize fluid computing by allowing an application to execute where it is best applied, either in the cloud or on the edge, depending on the use case. All the while, we must ensure that communications and data that flow between the device, the edge, and the cloud are safe and kept in compliance with the Health Insurance Portability and Accountability Act of 1996 (HIPAA).

SYSTEM REQUIREMENTS FOR MEDICAL DEVICES: AN EXAMPLE

Looking across the landscape of medical devices in use today, there is a wide range of applications—but regardless of size, shape, and use case, all medical devices share the same need for system reliability, ease of use, and fault tolerance.

Let's take a look at the system requirements of one device as an example: the wall-mounted defibrillator. Defibrillators are the quiet sentinels seen in hallways and rooms wherever there are large numbers of people. These devices are involved in both life-altering and lifesaving situations. They are often used by innocent bystanders who are not familiar with how the device works—and in many cases people are faced with learning to use it quickly because every second matters. The basic system requirements of a defibrillator include:

1. **A long-term shelf life:** It may be wall powered or completely battery operated, but it must work within a moment's notice to monitor a patient's vitals and, if needed, deliver the necessary treatment.
2. **An easy-to-use human–machine interface (HMI):** The HMI must be so simple that anyone who can read or hear the language can use it.
3. **Secure and stable communications:** Communications are essential to enable the defibrillator to self-diagnose.
4. **A multi-CPU design:** The design should help ensure efficient operating performance along with taking advantage of power management opportunities.

Now let's look at these requirements in a little more detail:

Long-Term Shelf Life

For medical devices that can actually save a person's life, the need for long-term shelf life is critical. If it is battery operated, the device will need to last its entire useful life without being recharged. This means the system software must be designed to minimize the use of the battery. One way to do this is to use two processors: a smaller processor, a type of "sanity check" processor, is used to keep the system alive, and a larger processor handles all the real-time events as they happen.

The smaller processor can be used to wake up the device once a day or once a week to do some evaluation of the hardware and ensure its readiness. It should also be able to report back to the administrator so that those monitoring the device know it's functioning normally or, in the event of a failure, can arrange a service call to fix a perceived malfunction.

Once the device is activated, it must wake up the main CPU, the "event processor." Among the many responsibilities of this core is to be capable of delivering an intuitive HMI. This core is also responsible for monitoring the patient's vitals and delivering the electrical shock the patient might need—all this while minimizing the use of power for the device's full lifecycle.

Easy-to-Understand HMI

In the event a patient needs the defibrillator, the device wakes up and instructs the first responder on how to use the device. This should be done both audibly, via a speaker, as well as visually, via some type of graphical user interface (GUI). It is important not to overcomplicate the GUI, as studies have shown that a simple, intuitive interface with minimal options will be used by a layman more quickly than an interface that is more complicated.

A clear and easy-to-use GUI involves instructing the user to correctly attach the pads on the patient. Once they are placed on the patient, the device must quickly determine whether the patient needs a shock. If so, then it charges the capacitors to the correct dosage and instructs the user to back off the patient so that they can be shocked without shocking any bystanders, again in easy-to-understand instructions.

Secure and Stable Communications

After the episode, the unit must phone home to give the data to the administrator who can relay the pertinent data in a HIPAA compliant way to the patient's doctor for further review. Then it must give instructions to the user on how to restore the system to its proper place so it is ready for the next event.

Multi-CPU System Design

The use of two cores is recommended, one to monitor the health of the device on a daily or weekly basis and the other to be capable of higher power, higher functionality to drive the interface, assessment of the patient data once the probes are attached, and shocking the patient.

In this case, the system must meet some communication needs, which involves the use of a GSM (Global System for Mobile communications) protocol stack, and during the event a much higher capacity CPU, to determine if a shock is warranted. Because of these needs, the use of a simple "round-robin" scheduler is exceeded and the use of either a real-time operating system or a general purpose operating system is warranted.

The operating system decision depends on several factors. First, are there fast boot and low power requirements? In this device every second counts. Minimizing the amount of software to run would warrant a real-time operating system, which is typically many times smaller than a general-purpose operating system. A smaller footprint means smaller RAM requirements and hence smaller power use needs. A smaller footprint also means less code to run. Since time is of the essence, a real-time operating system is warranted. If a medical device was used in a non-emergency situation, a general-purpose operating system would be acceptable and the availability of middleware could be an advantage.

ADDITIONAL SYSTEM SOFTWARE CONSIDERATIONS

Beyond the four key considerations discussed above, medical device designers should take into account a broad range of additional system software attributes, including:

Modularity: Medical devices need to adapt to changing needs in the network, so the operating system must be built on a modular, upgradeable, future-proof architecture that separates the core kernel from middleware, protocols, applications, and other packages. It should provide a stable core so that middleware, new protocols, and other packages can be added or upgraded without changing the core. This modularity will also help manufacturers of medical devices better differentiate their products and maintain them competitively over longer periods of time.

Scalability: With the proliferation of medical devices and classes of devices—ranging from small form factor, single-application devices to large-scale, complex, multi-application systems—the scalability of the system software is of utmost importance. A single RTOS that can scale to meet the unique memory footprint, functionality, and processing power requirements of multiple product classes can help manufacturers of embedded systems increase the return on their operating system investment, cut development costs by leveraging the economies of scope, and reduce time-to-market.

Security: Medical device system software needs to support security features not only to protect against malware and unwanted or rogue applications but also to deliver secure data storage and transmission and tamper-proof designs. OS-level support for these features is critical, since adding them at the user or application level is ineffective, expensive, and risky. And, since security threats are always changing, the system software needs to support the secure upgrade, download, and authentication of applications to help keep devices secure going forward.

Safety: Clearly, safety is paramount for medical devices because they could endanger life and malfunctions could cause injury or death—but not all medical device applications are equally life critical. When evaluating system software, look for features and capabilities that allow multiple applications with different levels of criticality to run on the same hardware platform.

Connectivity: Medical devices are increasingly connected to public networks for a wide range of applications. This means the system software may need to support a wide range of communications standards and protocols such as CAN, Bluetooth, Continua, IEEE 802.15.4, Wi-Fi, and Ethernet—and deliver high-performance networking capabilities out of the box. In addition to these capabilities, look for system software that can help retrofit existing devices with the required connectivity options so they can be brought online without reworking the core of their embedded software.

Rich user interface: With customer experience and the user interface becoming key differentiating features for medical devices, powerful yet easy-to-use human–machine interaction capabilities are becoming a must for system software, including quality 2-D and 3-D graphics engines, support for multiple monitors and touch screens, and rich graphic design tools.

COMMERCIAL VS. OPEN SOURCE: DECISION CRITERIA

The availability and use of both commercial and open source development options are pervasive among medical device manufacturers. Each has unique advantages and trade-offs, but the choice typically comes down to the completeness and sophistication of commercial offerings versus the low cost and ubiquity of open source software.

Commercial offerings are now available specifically for the creation of medical devices. For example, VxWorks® Plus delivers all of the rich feature set of VxWorks real-time operating system (RTOS), with an additional collection of advanced middleware and protocols for security, safety, networking, connectivity, device manageability, user interface (UI), and graphics that customers need to create the most demanding devices for the Internet of Things (IoT). This includes features that help meet the requirements of medical device manufacturers (for up to Class III medical equipment). Additionally, it includes a compliance package to facilitate approvals from the U.S. Food and Drug Administration (FDA) and other regulatory agencies worldwide.

Open source software options, such as the Linux operating system, are also popular for a number of good reasons:

- Distributions are free and can be modified and redistributed under the GNU General Public License (GPL).
- Thousands of developers have adopted Linux, making it easier to find developers who use it frequently and know it intimately.
- Linux runs on virtually any processor and is supported by virtually all major hardware manufacturers.
- The maturity of Linux has made it a practical choice in medical device development.
- Linux is feature-rich in tools, management, security, and graphics—important for medical device screens that require clarity and readability—and has a large ecosystem of board and software providers.

For all its advantages, however, using open source software such as Linux in a medical device also poses a number of challenges. For example, medical device manufacturers must follow several FDA guidance documents, and the medical device software standard IEC 62304 is now recognized or required in most jurisdictions. In addition, medical devices marketed in the United States are regulated by the Center for Device and Radiological Health (CDRH), a branch of the FDA. The FDA makes it clear that the burden of ensuring safe and reliable performance does not end with product launch. When evaluating operating systems, planning for bug fixes and security updates for the entire lifecycle of the product is recommended.

CONCLUSION

It is incumbent on no organization, vendor, or individual to tout any particular system software product or approach as superior to all others for medical device makers. Needs and requirements vary greatly, as do features, functions, and capabilities. It is important, however, to evaluate the full range of considerations before making the selection. After all, human lives are at stake in the creation and use of medical devices.

One day we may find ourselves at the mercy of some stranger trying to use a defibrillator that we designed. Do you trust your own design to work every time—especially when it's needed at the most critical time? If we design with this question in mind, the devices we create will work each and every time. When you've improved the quality of life for everyone who comes into contact with the medical device you designed—that's a job well done.