Real-Time Operating Systems: Perspective

Summary
Real-time operating systems are designed to deliver precision timing and predictable response, especially in aerospace, automotive and medical equipment applications—where microseconds and nanoseconds matter.

Table of Contents
Technology Basics
Technology Analysis
Business Use
Standards
Selection Guidelines
Recommended Gartner Research
Insight

List Of Tables
Table 1: Markets/Applications for Embedded Systems Technology
Table 2: Common RTOS Standards
Real-Time Operating Systems: Perspective

Technology Basics

Embedded computer systems are at the core of machines we use in our daily lives, from cellular phones to pagers to cars. They can be found within consumer products, like microwave ovens and digital cameras, and within dedicated technical equipment, like industrial measurement systems, medical instruments and car engines.

These hidden systems are very small, and for the most part, they are transparent to us. We are not even aware that they work behind the scenes to perform timely and consistent tasks that make our lives easier. Embedded systems typically comprise both hardware and software. The hardware consists of tiny components, like microcontrollers or microprocessors. The software is usually a real-time operating system (RTOS) that performs dedicated tasks and is designed to control time-dependent applications and components in a consistent and predictable manner. RTOSs form the foundation of an embedded system.

This emphasis on precision timing is what sets RTOSs apart from their general-purpose operating system counterparts (that is, Microsoft Windows XP, Unix, Linux). Although today’s microprocessors allow operating systems to run faster, RTOSs are designed to handle another important element: complexity. Although today’s microprocessors allow general-purpose operating systems to run faster, speed alone cannot guarantee that a time-critical application will meet its deadlines consistently. Sometimes deadlines will be met, and sometimes they will not, and that is because general-purpose operating systems simply are not designed to respond to time-critical events within a predictable time frame.

Not even the fastest processors running general-purpose operating systems are equipped to handle real-time, mission-critical applications. An RTOS must respond in a fully predictable manner to unpredictable external occurrences. If it does not, in many cases it can impart severe and irreversible repercussions. Timing errors, for example, in a car’s anti-lock braking system, or in an airplane’s autopilot system, can threaten or even result in the loss of life.

RTOSs are created to be tough and rigid but, at the same time, very flexible. A highly complex genre of software, an RTOS is based on highly intricate inter-task communications and multitasking technologies. An RTOS’s effectiveness is decided not only by the correctness of a job’s results, but also by the time in which the results are produced. Real-time computing occurs when the system acts in a predictable manner, but also within an exact delivery time.

RTOSs are designed to handle the execution of applications within extremely rigid response times—sometimes involving microseconds and nanoseconds. This timing factor, of course, separates RTOSs from their general-purpose operating system brethren. General-purpose operating systems, like Microsoft Windows, Unix and Linux, can be slow and unresponsive at times. RTOSs absolutely cannot perform sluggishly. They must meet system-timing constraints to ensure predictability and accuracy. They must react in a timely, fully predictable way to unpredictable external conditions as they arrive. They must eliminate risk from extreme load conditions. Embedded operating systems are incorporated within microprocessor-based products and supply the software interface between the embedded application software—which provides device-specific controls—and the underlying hardware.

Another distinctive characteristic of RTOSs is that they must be able to run with minimal resources. Microsoft Windows and Unix often require megabytes of memory and expensive high-speed processors to run effectively. Because they typically run on high-volume devices and systems, RTOSs, conversely, require a minimal amount of memory and on low-cost processors.
Real-Time Operating Systems: Perspective

It is challenging for developers of real-time embedded systems to create systems that respond predictably to demanding outside forces. It is difficult to anticipate such things as varying configurations, memory requirements and module dependencies while at the same time building a reliable system. RTOS developers spend a significant amount of time looking at worst-case scenarios, since real-time systems usually operate unattended, and any breakdown or failure of a real-time system can produce catastrophic results. To ward off such results, the designer must produce a system that is deterministic and ensures worst-case interrupt latency and context-switch times. A deterministic operating system ensures that the worst-case execution time of each of its system calls can be calculated.

Embedded—or hidden—systems and the RTOS software that drives them are at the foundation of a diverse range of important and technical applications. Some of these applications are highly sophisticated and very technical in nature. These include automotive engineering, aeronautics and aerospace systems, data communications and telecommunications equipment, industrial control devices, medical imaging, and instrumentation and scientific systems.

RTOSs are also driving a growing consumer-oriented market. This includes information appliances, such as cellular phones, digital cameras, digital video disks (DVDs), inkjet printers, Internet-enabled handheld PCs, pagers and TV set-top boxes. New and evolving technologies, such as the Internet, miniaturization of components and wireless communications, are helping to extend the reach of embedded systems technology.

Real-Time Operating System Characteristics

A real-time operating system is one that ensures that the response to an event will occur within a fixed time. The integrity of a real-time system depends on two things: the results of a computation and the time at which these results are delivered. To produce results in the spontaneous yet predictable manner in which it does, an RTOS must focus on multitasking and task scheduling. The following looks at several areas that are characteristic of RTOSs.

Inter-Task Communications

A task is a single-threaded, sequential group of instructions with its own space. Generally, tasks run in their own world with no awareness of one another. It is the operating system that must manage and coordinate simultaneous task execution by assessing each task’s priority and assigning processor resources to each task on an as-needed basis.

Within a general-purpose operating system, each process has equal access to the microprocessor. If a very important task needs to run, and a lesser task is already running, then the important task is usually forced to wait. This is called preemptive scheduling. With RTOSs, however, tasks are given priorities. This is so a task assigned to a priority event can immediately take over the microprocessor. Important tasks do not have to wait to run while less relevant processes execute. This is called deterministic scheduling. It refers to the ability to predict when a specific event will occur at its precise moment.

Before an application is run, control blocks, task priorities and task stack space must all be determined. Priority levels must be assessed for every application, driver, task and interrupt handlers. There are other variables, including disk transfer algorithms, main memory residency status, paging and process swapping. The number of concurrent task processes varies with each RTOS.

High Reliability and Availability

Reliability and availability are critical for RTOS environments. While a failure in a general-purpose operating system merely imparts inconvenience and forces a system reboot, a failure in an RTOS can result in the loss of human life. RTOSs must ensure reliability and availability for their applications. They
can insulate most of the work for building high-availability applications from the application designer by automatically providing the means to manage inter-process communication, supervise system resources, and automatically detect and handle errors.

It is important to distinguish between Safety-Critical requirements and High-Availability requirements. Safety-critical requirements demand robust memory protection, guaranteed resource availability and often require governmental or industry certification (for example, FAA DO-178B). High reliability and availability systems might simply cost money if they fail (for example, a telecommunications system). These systems require redundancy and rapid recovery, but no loss of life is threatened.

**Small and Modular Architectures**

RTOS vendors usually offer a modular operating system, one that is centered on a kernel and surrounded by extensions. A modular operating system has a small kernel with only the basics: inter-process communication, memory allocation, process creation and scheduling functions. There is a wide variety of extensions that can be added or subtracted to meet the specific requirements of a device, development tool or real-time application.

RTOSs have such extensions as communications protocol support, device drivers, file systems, graphics and windowing, Input/Output (I/O) managers, TCP/IP and user interface elements. The QNX Neutrino RTOS, for example, can be configured to operate on an Intel x86 CPU with just a few hundred kilobytes of memory. Even when such elements as a file system, TCP/IP stack, windowing system and applications are added, it still only requires about 2MB to 4MB of memory—a fraction of the memory required for a typical desktop system. The modular architecture of an RTOS is well suited for small, low-cost devices that are manufactured in high volumes. This is because these devices must keep cost, function and size to a minimum.

A common thread of most real-time systems is that they usually require a very small memory footprint—sometimes as little as 2KB to 10KB. A smaller footprint enables the costs of memory to be decreased and, in some cases, allows a more compact physical design and reduced power consumption. RTOSs tend to minimize RAM in favor of read-only memory (ROM). Some even use flash memory. ROM is less expensive than RAM and does not need to boot from a hard drive. ROM is also denser, which means it consumes less space on a microprocessor. Kernels occupy only a small fraction of space, with vendors attempting to use RAM for the essentials.

As embedded designs become more complex, with more features and more concurrent operations, the size of a device’s software frequently exceeds the practical limits of the “ROM-based” deployment model. Subsequently, most of the designs we see today store code in flash memory and then load it into RAM for execution when the system boots. Therefore, an RTOS needs to provide high-performance, fault-tolerant flash file systems.

Such minimal memory requirements are needed in kernels for low-cost, dedicated application environments, such as handheld devices and portable phones. In some instances, a small footprint enables a complete application to fit into on-chip memory, without the need to employ additional external devices. RTOSs that are embedded into microprocessor-based devices can accommodate a wide span of applications, ranging from cell phones to airplane ailerons to micro-controllers running nuclear power plants.

**Portability**

Obviously, the more target processors that an RTOS supports, the better. With its VxWorks RTOS, Wind River Systems supports more than 20 processors. Although there are many others, most RTOS vendors
Real-Time Operating Systems: Perspective

Support ARM, IBM and Motorola PowerPCs, Intel x86, MIPS, Motorola 68000 and ColdFire, and StrongARM chips. Some others, like QNX Neutrino, also target Hitachi SuperH and Intel Xscale.

RTOS Essentials

In order for an operating system to be considered real-time, it must provide the following characteristics: dependability, predictability, simultaneity and timeliness. More specifically, RTOSs must provide/support the following features:

- Multithreaded and preemptible capabilities.
- Thread prioritization levels (256 levels are supported by many RTOSs).
- Predictable thread synchronization.
- Priority-based preemptive task scheduling.
- Instant response to external events. Hard real-time systems demand response within one microsecond, while soft real-time systems might demand only 10 millisecond response.
- Fast I/O.
- Minimal periods of disabled interrupts.
- Usually a small kernel with minimal memory requirements; however, some markets do not require this, such as communications infrastructure equipment.
- Usually a development environment that enables programmers to build in their own unique operating system capabilities.

The ABCs of Real-Time Operating Systems

- Deadline—A specific moment or predetermined time unit in which a specific processing action must be completed.
- Determinism—A measure of how long an RTOS can delay before recognizing an interrupt. The closer an RTOS can get to achieving a fully deterministic process, the better the entire system performs.
- Hard Real-Time—Real-time computing with strictly specified, absolutely precise timing requirements. The exactness of the timing can come down to microseconds. Anti-lock breaking and military systems are examples of hard real-time applications.
- First In, First Out (FIFO) Scheduling—A scheduling algorithm in which a task continues to execute until it voluntarily relinquishes control or is preempted by a higher-priority thread.
- Interrupt—An internal or outside event that can launch a predetermined action from the processor.
- Interrupt Latency—Unit of time between a hardware interrupt occurs and is acted on.
- Memory Protection—Use of the processor’s Memory Management Unit (MMU) to provide multiple address spaces, enabling programs to be protected from each other, and protecting the kernel from application programs.
- Microkernel—A small kernel that implements a collection of core services. This set usually includes inter-process communication, interrupt handling and a task-scheduling architecture. Some RTOSs
Real-Time Operating Systems: Perspective

use a microkernel architecture; others use a monolithic kernel similar to that used in general-purpose OSs like Windows and Unix.

- Modularity—An RTOS is typically composed of a kernel that is surrounded by optional extensions. A developer can customize an RTOS by adding or subtracting extensions (for example, communications protocols, graphics and windowing managers and so on) to suit a specific application.

- Portable Operating System Interface—(POSIX) is a set of standards developed by the Institute of Electrical and Electronics Engineers (IEEE) that cover most operating system services. POSIX 1003.1b (real-time), 1003.1c (threads) and 1003.d (more real-time) are concerned with RTOSs.

- Predictability—Refers to the concept that the precise deadline and the task to be performed are known in advance.

- Preemptive Scheduling—Ability of a system to defer a low-priority task to a high-priority task.

- Priority—A predetermined order of process execution.

- Priority inheritance—An RTOS mechanism that helps prevent priority inversion, a condition in which a low-priority thread consumes CPU cycles at a higher priority than it should.

- Round-Robin Scheduling—The scheduling of tasks at equal priority levels; they are handled in order of arrival.

- Soft Real-Time—Real-time computing that is designed to perform consistently and predictably, but offers leeway in time constraints. Soft real-time requires that deadlines be met, but those deadlines are “softer” (longer periods of time) than the deadlines of hard real-time systems. Credit card readers and point-of-sale systems are examples of soft real-time systems.

- Sporadic Scheduling—A scheduling algorithm that allows a thread to service periodic events without jeopardizing the hard deadlines of other threads or processes in the system.

- Task—A specific function of the system.

The Difference Between Hard and Soft RTOSs

**Hard RTOSs**

A real-time operating system is generally considered either “hard” or “soft.” A “hard” classification refers to requirements in which a task has a hard, firm deadline of less than a few microseconds. There is no wiggle room. If the task is not completed by the determined deadline, it has failed. If a single task has failed, the entire system has failed. It is the duty of the hard RTOS to shut down the action—if a critical processing event does not occur or does not occur fast enough—so that the availability or reliability of the entire system is not compromised.

For example, let us say that a mechanical device is configured to pick up an object from a moving conveyer belt. Since the object is in motion, the mechanical device has a very tight window of opportunity to correctly lift up the object. If the device’s timing is late, even though the location is correct, the object will have passed by. Conversely, if the device’s timing is too early, the object will not yet be there. Either way, the system has failed.

Another example of a real-time application is an aircraft’s wiring control system. If any single activity in this control system does not operate exactly on time and in the predicted manner, the aircraft can no longer be controlled. Subsequently, this means loss of the aircraft and perhaps loss of human life.
Real-Time Operating Systems: Perspective

Basically, a failure in a hard real-time system can lead to loss of life, major equipment damage or serious financial loss. For these reasons, very tight constraints and controls are required and found in many real-time environments, such as car engines, medical equipment monitoring and space navigation equipment.

Those developing hard real-time systems must concentrate on the behavioral attributes of specific system tasks and their execution environments. For task attributes, system developers must define timeliness parameters, such as arrival times, deadlines, resource usage, suspension times and worst-case execution times. Deadline requirements must be converted into thread priorities, so a real-time operating system must be multithreaded and preemptible. For execution attributes, developers must specify caching, interrupt priorities, queuing, resource interactions and system-loading factors.

Soft RTOSs

A soft real-time system must complete processing and respond to real-world events in a predictable manner every time. They are required to perform quickly and consistently, but there is a less-demanding deadline for response than is found in a hard real-time system. Although it is best to be as precise as possible, it is acceptable if some deadlines are missed. Soft RTOS systems are somewhat forgiving and will let some noncritical mishaps occur. Basically, lateness in soft real-time environments does not result in some type of complete system failure or catastrophe.

Technology Analysis

Embedded system technology plays an important behind-the-scenes role in the devices that touch us on a frequent basis. Software developers have used real-time operating systems since the early 1980s, when most real-time operating systems were proprietary. While many of these legacy real-time systems still exist, there is a growing trend toward the use of commercial RTOSs.

There are numerous reasons for the shift from proprietary to commercial systems. These include increased application complexity, faster time-to-market demands and more standardized requirements. There are many functional commercial RTOSs on the market. As embedded applications have grown more sophisticated, so have the RTOSs that drive them.

In years past, a real-time kernel only had to supply some type of inter-tasking relationship structure and rudimentary I/O mechanisms. In fact, most of the functionality was delivered by the application. Things have changed, however. Now, the RTOS is responsible for providing much of the functionality, including such elements as POSIX and network file systems, graphical user interface (GUI) application programming interfaces (APIs) and networking protocols. Today's commercial real-time kernels now include many features found in traditional Unix operating systems, with of course extensive scheduling and memory management capabilities. With their support for graphical user interfaces, inter-processing capabilities and widespread connectivity, today's real-time operating systems now include more functionality than ever.

Software from traditional RTOS vendors—like LynuxWorks (formerly Lynx Real-Time Systems), Accelerated Technology (acquired by Mentor Graphics in March 2002), Green Hills Software, Microware Systems (acquired by RadiSys Corp. in August 2001), QNX Software Systems and Wind River Systems—is used in such diverse markets as automotive, aerospace/defense, consumer appliances, digital imaging, industrial measurement control and Internet infrastructure. Even Microsoft, the industry leader in Intel x86-based operating systems and application software, has established a growing presence within the embedded market with Windows CE and the newer CE.NET.

Within recent years, there has been a shakeup of the commercial RTOS market, involving a variety of acquisitions. First came Wind River’s US$1 billion acquisition of Integrated Systems Inc. (ISI) in 2000. ISI
was one of Wind River’s leading competitors in the real-time operating systems market and a specialist with more than 15 years’ experience. Wind River Systems continued its quest to diversify with the acquisition of Eonic Systems’ Virtuoso in April 2001. Virtuoso, which is designed for the digital signal processing (DSP) applications, is used in such devices as digital cameras and fax machines. Wind River has renamed this product VSPWorks.

Even non-RTOS specialists got into the act, in hopes of bolstering their portfolios with embedded systems technology. In August 2001, RadiSys Corp., a provider of Internet and communications systems, acquired Microware, another real-time operating systems expert and its popular OS-9 RTOS. In March 2002, Mentor Graphics, an electronic hardware and graphics software design vendor, acquired Accelerated Technology and its Nucleus Plus real-time software.

A Look at the RTOS Market

Like most computer-related industries, the embedded software market stagnated in 2002. Big customers of RTOS software and development tools, like consumer electronics, data communications and telecommunications, were heavily impacted by the economic downfall.

Embedded systems are used most frequently in the data and telecommunications, consumer electronics, industrial controls and military/aerospace electronics industries. The technology is additionally used in computers/peripherals, medical electronics equipment and automotive electronics.

The RTOS market is dominated by Wind River Systems, with a handful of other vendors comprising the rest of the share. According to a May 2003 RTOS market study issued by Gartner Dataquest, Wind River Systems (VxWorks, VSPWorks, pSOS) leads with 39.6 percent of the market, followed by a fragmentation of others, including Accelerated Technology’s VRTX and Nucleus, Green Hill Software’s INTEGRITY and ThreadX, QNX Software Systems’ Neutrino, Enea OSE and others). Microsoft Windows CE.NET and Linux also carry a market presence.

Wind River is attempting to add value to its already popular offerings by packaging solutions stacks for specific industries. In November 2002, Wind River announced Wind River Platforms, a series of market-specific integrated embedded platforms. The Platforms provide an approach to embedded product development with industry-specific operating systems, tools, networking, connectivity and device management, reference hardware, and support services. Some of these Platforms are geared for the following areas: consumer devices, industrial devices, network equipment, server appliances, and the DO-178B standard for airborne systems and equipment.

Other vendors are pursuing similar strategies. QNX, for instance, provides technology bundles customized for developers of in-vehicle telematics devices. This technology is used in cars and integrates GPS satellite tracking and wireless communications for remote diagnostics and roadside assistance. QNX’s packages include industry-specific applications, drivers, and source code, as well as board support bundles for telematics hardware.

The Linux Factor

Linux, the freely available, open source Unix-like operating system, has made an impact on embedded software developers. The operating system has matured over the past several years—both in terms of functionality and market presence. Linux is emerging as the fastest-growing server operating system. Now, many embedded systems developers are investigating Linux as an alternative to commercial RTOSs.

The open source model offers interesting benefits to embedded system developers. Linux vendors believe that they have the best operating system for devices with limited memory, like cell phones and
Real-Time Operating Systems: Perspective

PDAs. Developers who select Linux are liberated from the burden of runtime royalties. Many find Linux to be a low-cost, flexible development option.

There are additional benefits. Due to the modular nature of Linux, developers have the freedom to change the software to fit their own real-time needs. Another advantage is that Linux supports a wide range of processors, including all of the leading embedded processors, like Alpha, ARM, IA-64, MIPS, PA-RISC, PowerPC, and SPARC. Further, Linux complies with POSIX for its basic API. This makes sharing applications between Linux and other POSIX-compatible operating systems relatively simple.

Linux can be found at the heart of embedded systems from HP and Samsung in palmtop computers, Celestial and Nettel in their business routers, Axis in its cameras, Nokia and TiVo in their set-top boxes, and Ericsson and Nokia in their telephones.

Despite its success in the server operating systems market, Linux is experiencing some potholes on the road to recognition in the embedded industry. Red Hat Software, the world’s dominant Linux distributor, has put a hold on its embedded Linux technologies after poor sales. Another well-known Linux embedded vendor, Lineo, has also experienced a decrease in revenue. SCO’s lawsuit against IBM for allegedly lifting key software code from its copyrighted Unix and dropping the code into Linux, has cast a pall on the entire Linux industry, and it certainly will not help the progress of Linux in the embedded systems market.

The Need for Linux Standards

Linux is maturing as an operating system, but there are still some incompatibilities between its different versions. Applications do not generally run the same way on every Linux operating system.

The embedded Linux market has a unifying, standards-setting body. The Embedded Linux Consortium (ELC) (www.embedded-linux.org) is a nonprofit, independent association whose objective is the advancement and standardization of Linux throughout the embedded computing market. Its aim is to make Linux a leading choice for embedded system developers. In February 2003, the ELC released the ELC Platform Specification (ELCPS), designed to build industry support for a single Linux development standard for embedded products. The specification provides an open platform for embedded operating systems. The ELCPS is designed to improve interoperability among competing Linux distributions by standardizing the application programming interface (API) layer of embedded applications. The ELCPS builds on established Linux and Unix standards such as the Linux Standard Base (LSB) 1.2, IEEE POSIX 1003.1-2001, and the Single Unix Specification Version 3.

According to the ELC, the interfaces defined in the ELCPS let companies develop and revise applications without being tied to a single OS vendor or version. Such independence can lead to reduced total system costs and time to market. Aligent Technologies, IBM, Matsushita (best known for its Panasonic brand), Red Hat, Samsung, Sharp and Sony are participants in the ELC.

There are incompatibility issues on another level—adding real-time extensions to enhance performance. Linux RTOS providers use their own model and APIs for their real-time extensions. At this point, there is no standardized, de facto method to add real-time capabilities to Linux.

Business Use

In the aerospace, industrial manufacturing, medical equipment and telecommunications industries—areas where milliseconds and nanoseconds matter—the use of commercially available real-time software is used frequently. In high-volume areas that traditionally do not require real-time response, such as office equipment, real-time software is increasingly used as the devices are expected to run faster and do more. To speed up development time and decrease costs, commercially available RTOSs are often used in the place of in-house products.
The following provides an overview of markets where RTOSs are used most frequently and the types of applications that are fueled by RTOS software.

### Table 1: Markets/Applications for Embedded Systems Technology

<table>
<thead>
<tr>
<th>Markets</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace</td>
<td>Aircraft wiring control systems, flight simulation, satellite control and tracking systems, space guidance and navigation</td>
</tr>
<tr>
<td>Automotive</td>
<td>Antilocking braking, antiskid testing, engine control systems, in-vehicle telematics</td>
</tr>
<tr>
<td>Consumer Devices</td>
<td>Audio and video equipment, car navigation systems, DVD players, set-top boxes</td>
</tr>
<tr>
<td>Data Networking</td>
<td>Access points, base stations, Ethernet switches, remote access servers, routers, switches, cable modem technology</td>
</tr>
<tr>
<td>Industrial</td>
<td>Process control mechanisms, motion controllers, nuclear power plants, robotics, test and measurement systems</td>
</tr>
<tr>
<td>Medical</td>
<td>Anesthesia monitors, blood-component collection systems, hemodialysis devices, hospital bed-side monitors, MRI and PET scanners</td>
</tr>
<tr>
<td>Peripherals</td>
<td>I/O control, RAID storage, X terminals</td>
</tr>
<tr>
<td>Small Office/Home Office (SOHO)</td>
<td>Browser-enabled cell phones, copiers, digital cameras, fax machines, printers</td>
</tr>
<tr>
<td>Telecommunications</td>
<td>Telephone switching systems, cellular phones, optical transport</td>
</tr>
</tbody>
</table>

### Standards

Many traditional RTOSs are proprietary, conforming only to their own manufacturers’ standards. However, an increasing number of commercial RTOSs conform to the following globally accepted IEEE POSIX standards: 1003.1 (device API, file system, process, operating system), 1003.1b (real-time), 1003.1c (threads), and 1003.2 (utilities). Some real-time operating systems support POSIX 1003.1d, which deals with real-time extensions (for example, interrupt handlers). POSIX 1003.1b was developed for real-time operating systems services and includes interface definitions for the following areas: asynchronous I/O, clocks, memory management, message queues, queued signals, scheduling, semaphores, task management and timers.

Developers that adhere to standards make their code portable to other operating systems that support the standard. By adhering to standards, RTOS vendors provide their customers with the flexibility to port open source applications and to add new features with minimal development cost.

There is a great deal of research occurring at universities, private institutes and centers, and special interest groups which feeds into the IEEE standards bodies. Researchers at these institutions pursue artificial intelligence, design, specification, programming languages, multimedia, scheduling, operating systems and networking technologies. As noted earlier in this report, organizations like The Open Group and the Embedded Linux Consortium are working to advance the standardization of embedded Linux technology. RTOSs are based on a variety of application development, I/O, POSIX, and networking standards. The following highlights some of the most widely supported RTOS standards.

### Table 2: Common RTOS Standards

<table>
<thead>
<tr>
<th>Category</th>
<th>Standard Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>ANSI C</td>
</tr>
<tr>
<td>Development</td>
<td>C++</td>
</tr>
<tr>
<td></td>
<td>POSIX 1003.1, 1b, 1c, 1003.2</td>
</tr>
<tr>
<td></td>
<td>Ultra C/C++ compiler</td>
</tr>
</tbody>
</table>
**Real-Time Operating Systems: Perspective**

**Table 2: Common RTOS Standards**

<table>
<thead>
<tr>
<th>Category</th>
<th>Standard Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>Bluetooth</td>
</tr>
<tr>
<td></td>
<td>ISO 9660 CD-ROM file system</td>
</tr>
<tr>
<td></td>
<td>PCMCIA</td>
</tr>
<tr>
<td></td>
<td>POSIX asynchronous I/O and directory handling</td>
</tr>
<tr>
<td></td>
<td>SCSI</td>
</tr>
<tr>
<td></td>
<td>USB</td>
</tr>
<tr>
<td>POSIX</td>
<td>POSIX 1003.1b (real-time)</td>
</tr>
<tr>
<td></td>
<td>POSIX 1003.1c (threads)</td>
</tr>
<tr>
<td></td>
<td>POSIX 1003.d (real-time extensions)</td>
</tr>
<tr>
<td>Networking</td>
<td>Berkeley sockets</td>
</tr>
<tr>
<td></td>
<td>BOOTP, DNS, DHCP, TFTP</td>
</tr>
<tr>
<td></td>
<td>FTP, rlogin, rsh, telnet</td>
</tr>
<tr>
<td></td>
<td>IPSec, IPv6</td>
</tr>
<tr>
<td></td>
<td>PPP, PPPoE, SLIP</td>
</tr>
<tr>
<td></td>
<td>TCP/IP networking</td>
</tr>
</tbody>
</table>

**Selection Guidelines**

Depending on the vendor’s market focus, real-time operating systems vary greatly in terms of functionality. In general, consider the benefits and limitations of proprietary vs. commercial software, processor support, the market experience of the vendor and maturity of the product, and whether the RTOS’ features meet your needs. In all, look for a highly reliable and flexible platform. And one that is fully supported by credible vendor.

Consider the supporting technology, including development environment quality, embedded system networking (for example, TCP/IP), graphics, and Microsoft and Java support. Determine whether proprietary vendors are moving toward commercial kernels, such as Windows, Unix, and Linux. If the requirement is a high-end system, consider 32- and 64-bit microprocessor technologies powering high-end embedded systems for controllers, network computers, routers, switches and servers.

Take a serious look at the requirements and whether the vendors address the particular niche market. Consider the long-term expense and the time it will take to develop applications on one system vs. another. Look at the vendors’ support facilities and whether the software is robust and scalable enough. Also consider how willing the vendor is to accommodate any special requirements. The following list provides several other considerations:

- **Application Development**—Most real-time operating systems offer varying types of development capabilities. Some support links to third-party programming tools, while others offer comprehensive environments with integrated ANSI C and C++, Java compliance, browsers, compilers, editors and utility libraries. Some vendors have added Java extensions to their systems to allow them to run applets. Make sure to consider RTOS products that offer standard, open APIs, and not ones that lock you in.

- **Memory Management**—To operate effectively, a real-time system must provide flexible and dynamic memory management features. It should offer such addressing and control features as dynamic memory allocation, pageable memory control, shared memory options, and virtual memory addressing. It should support the memory management unit (MMU) of a microprocessor.
Real-Time Operating Systems: Perspective

- Network Connectivity—RTOSs should offer support for a full range of networking standards and protocols, including File Transfer Protocol (FTP client and server), IPsec, IPv6, NFS (client and server), network login security, and TCP/IP (client and server).

- Processor Support—Look for an RTOS that is highly portable, since new and more advanced processors are arriving at a rapid pace. Check to see if the source code needs to be drastically altered to move from platform to platform, or if it needs to be recoded to handle a variety of device drivers.

- Real-time I/O—A real-time operating system should be particularly sound in the area of I/O. It should support asynchronous and synchronous I/O, contiguous disk files, direct I/O from user space, POSIX I/O and directory handling, and SCSI devices.

- Scalability—Investigate the scalability of the RTOS. Does it support symmetric multiprocessing (SMP), and how will it limit growing needs?

- Task Scheduling and Management—An RTOS would not be considered such without multitasking and preempting capabilities. Kernels should allow the developer to specify priority scheduling with an unlimited number of tasks, round-robin scheduling, deterministic context switching, inter-task communications, priority inheritance, and efficient interrupt handling. It should also allow application tasks to be distributed among available processors.

Recommended Gartner Research

2002 Worldwide Embedded Software Tools Market Share, SWTA-WW-MS-0137

Insight

Embedded applications are growing more complex and pervasive. An increasing number of applications are demanding real-time common characteristics like adaptability, availability, consistency, predictability and reliability—all critical features for RTOS environments. The commercial RTOS market has been stunted over the past year by the poor economy. The major vendors are fighting back with stronger products, improved licensing, and investigating new markets.