

Getting Real with NT

Approaches to Real-Time Windows NT

This article presents the Radisys approach to extend Windows NT to support Real-Time applications. After presenting the seven possible means to extend Windows NT, the article explains the solution chosen by Radisys. It is based on modifying the HAL and taking advantages of the hardware tasks offered by the Intel processors.

Developers of industrial, real-time, and embedded computing systems have for years faced an excessively fragmented operating systems market with a large number of commercial real-time operating systems (RTOSs), kernels, executives, general purpose OSs, and many proprietary solutions. While the low end, or "deeply embedded" market will probably continue to be served by a variety of specialized solutions, Windows NT has captured the attention of developers of higher-end products. INtime™ from RadiSys delivers hard real-time operation for Windows NT, extending the usage of Windows NT to embedded applications such as machine tools, industrial control, medical and communications equipment that require mission-critical performance and determinism. These applications can take full advantage of Windows NT's standard user interface, network capabilities, development tools, and off-the-shelf software and still deliver rock solid performance of critical real-time tasks. INtime brings together field-proven real-time technology and robust, seamless integration with Windows NT to deliver hard real-time capabilities for Windows NT.

INtime from Radisys enables the usage of Windows NT in real-time applications that demand a level of determinism, reliability, and performance which are typically beyond the scope of general purpose operating systems. With its fully pre-emptive, multitasking kernel, Microsoft characterizes Windows NT as being "soft" real-time, that is, capable of satisfying most response requirements, on average, within a given set of constraints. Another way of saying this is that you can expect Windows NT to miss some of your scheduling deadlines. Whether or not this is acceptable depends on your application, of course. The "non-real-timeness" of Windows NT, rooted in the fundamental design of the OS, is the result of various kernel policies and mechanisms. Although these policies and mechanisms were put in place for good reason, namely to optimize average system performance, they spell trouble for the real-time or embedded systems programmer. See (1) for a complete discussion of the limitations of Windows NT as a real-time operating system.

Notwithstanding these limitations, there is still tremendous promise for Windows NT in real-time applications. There are seven fundamental approaches that attempt to bring together, with varying degrees of success, the advantages of Windows NT, and the demands of real-time and embedded systems development:

- restrict Windows NT to soft real-time applications,
- create and enforce a finely tuned, highly constrained environment for the application,
- provide a Win32 API wrapper around an RTOS,
- couple a real-time operating system with Windows NT,
- modify the NT kernel,
- modify the Hardware Abstraction Layer, and
- compliment the standard NT kernel with a real-time kernel.

RESTRICT WINDOWS NT TO SOFT REAL-TIME APPLICATIONS

If your application can handle occasional "hiccups" or delays, you may be able to use standard Windows NT. Although the actual window of predictability is up for debate, your application should be able to handle timing variations in the 1-20 millisecond range with the realization that there are no guarantees. The cost of missing deadlines should be relatively low, and not result in a system failure or unacceptable performance degradation.

CREATE A FINELY TUNED, HIGHLY CONSTRAINED ENVIRONMENT

By paying careful attention to the system load, interaction with other processes (via the network or locally), and in effect "closing" the system, you can limit the amount of spurious or unpredictable behavior. You may also need to write most of your application in NT's kernel mode, with the majority of your code in device drivers. An NT expert who knows what's going on under the hood and knows where the hidden dangers lie, may be able to construct a finely tuned sys-

There are seven fundamental approaches that attempt to bring together the advantages of Windows NT, and the demands of real-time and embedded systems development.

tem which meets your requirements, for now. However, developing any substantial applications with such restrictions neutralizes many of the benefits of an OS such as Windows NT and will result in code which is very difficult to support and maintain.

PROVIDE A WIN32 API WRAPPER AROUND AN RTOS

This approach, which provides the ability to cross compile Win32 applications to run on a RTOS, doesn't actually make use of Windows NT. Standard Windows NT applications cannot be utilized with this approach, and you are limited in your options for future expandability. Also, since the target system is not Windows NT, you are forced to use specialized tools for compilation and debugging.

COUPLE A REAL-TIME OPERATING SYSTEMS WITH WINDOWS NT

In most cases, this means running Windows NT and a real-time operating system (RTOS) on separate systems. For this approach, the Windows NT system is used only for the operator interface and other non-real-time functions. The dedicated RTOS system is used for the actual real-time control. This scenario requires you to learn and maintain two separate development environments and also increases the cost of the total system by requiring at least two computers. Running the two OSs on the same system eliminates the extra hardware costs, but still requires two separate development environments.

MODIFY THE WINDOWS NT KERNEL

Because Microsoft does not license the source code to the Windows NT kernel to third parties, this is an option which is only available to Microsoft. Because of their focus on the broader OS market, indications are that these types of modifications will not be coming from them.

MODIFY THE HARDWARE ABSTRACTION LAYER (HAL)

The HAL is a layer of code between the NT executive and the hardware platform that hides hardware-dependent details such as I/O interfaces and interrupt controllers. Microsoft routinely grants HAL source code licenses to hardware vendors who need to do special adaptations for their hardware to run Windows NT. Microsoft has also granted HAL source code licenses to various companies, including Radisys, for products that attempt to extend Windows NT with real-time capabilities. Performing extensive modifications to the HAL, such as manipulating the clock or rewriting the way in which interrupts are processed, represents an unprecedented, unproven use of the HAL, creates a non-standard environment, and may pose serious maintainability challenges.

Even more importantly, because the "non-real-time-ness" of Windows NT is rooted in basic NT kernel mechanisms, modifying the HAL can only result in slightly improved soft real-time performance. As long as the standard Windows NT executive is used to schedule and process threads and interrupts, hard real-time determinism is not possible.

COMPLIMENT THE STANDARD NT KERNEL WITH A REAL-TIME KERNEL

Any solution which claims to bring hard real-time performance to Windows NT must provide an alternate kernel to handle real-time task scheduling and execution, running in conjunction with the standard Windows NT kernel. Introducing such a kernel into the Windows NT environment, however, may actually decrease system reliability unless the kernel is at least as reliable as the Windows NT kernel. It is critical, therefore, that the real-time kernel be proven in real-life applications, with extensive testing which can only come through repeated usage over time. Other important considerations are memory and address space protection, as well as the ability to survive catastrophic system failures (NT "blue screen" crashes). Finally, clean integration with the Windows NT environment, for example leveraging the same development tools and APIs where possible, is critical for the general usability of the solution.

There are at least two alternatives for coupling a real-time kernel with the Windows NT kernel: a) place the kernel inside of an NT interrupt service routine or device driver, or b) place the kernel outside of NT's address space in a separate hardware task.

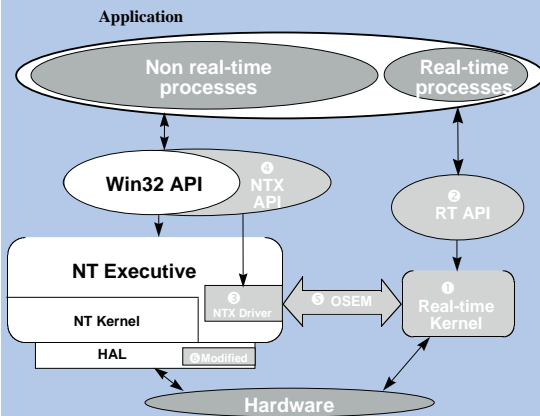
Real-time kernel inside an NT interrupt service routine (ISR) or device driver

At first glance, this is the most straight-forward, and easiest to implement approach. However, with such an approach, the user is forced to develop real-time applications in the NT kernel mode (as opposed to user mode, the "normal" development mode). In the NT kernel mode, code has privileged access to the entire memory space, including the NT kernel and other device drivers, with no address isolation or memory protection offered. Thus, a real-time thread could easily overwrite the address space of another process, including other real-time processes. Because these types of programming errors are typically extremely difficult to detect and result in spurious but critical failures, achieving reliable operation often requires extensive testing and debugging, with many errors not detected until after a system has been deployed in the field. Writing a complex, multi-threaded real-time application in this privileged mode is contrary to the programming model espoused by Windows NT.

Equally as serious is the challenge of maintaining reli-

INTIME™ PROVIDES WINDOWS NT WITH REAL-TIME PERFORMANCE AND CAPABILITIES

INtime is installed on a system running a standard version of Windows NT. Once installed, Windows NT and INtime communicate as shown:



INtime consists of:

- 1. Real-time kernel:** The real-time kernel provides deterministic scheduling and execution of real-time threads. INtime real-time interrupts and active INtime threads immediately pre-empt the execution of any NT threads and disable all non-real-time interrupts.
The real-time kernel is based on the iRMX real-time operating system, originally developed by Intel in the early 1980s. Over the years, the core technology of this real-time operating system has continually advanced, resulting in a highly responsive, highly reliable, field-proven real-time kernel.
- 2. RT API:** Real-time threads access the capabilities of the real-time kernel via a real-time application programming interface (RT API), modelled after the Win32 API. By utilizing the RT API, developers can achieve real-time determinism within a familiar Windows NT environment.
- 3. NTX Driver:** The NTX driver is a Windows NT device driver that provides centralized support for the OSEM. The NTX driver facilitates communications between real-time kernel threads and NT threads.
- 4. NTX API:** The NTX API extends the Win32 API with a small set of functions that allow non-real-time threads to communicate and synchronize with real-time threads.
- 5. Patented OS encapsulation mechanism (OSEM):** The OSEM manages the simultaneous operation and integrity of the NT kernel and the real-time kernel, and provides memory protection and address isolation between processes for added reliability and robustness.
- 6. Modified Windows NT Hardware Abstraction Layer (HAL):** INtime includes a special version of the Windows NT HAL that improves the overall reliability and robustness of the system.

able operation of the real-time kernel in the event of an NT blue screen crash. By definition, when Windows NT crashes, something catastrophic has occurred such that Windows NT itself cannot recover. The integrity of all of Windows NT is in question, including interrupt handling, the operation of all device drivers, and HAL services. Continued operation of the real-time kernel which is encapsulated by Windows NT will be unreliable at best, and most likely lead to the crash of the real-time processes.

Real-time kernel outside of NT's address space, in a separate hardware task

Radisy's has pioneered an approach to real-time Windows NT, called INtime™, which provides robust, seamless integration with Windows NT. INtime™ utilizes standard Intel-architecture support for hardware multitasking to maintain proper address space isolation and protection between non-real-time Windows NT processes, and real-time processes. Radisy's patent-pending OS encapsulation mechanism (OSEM) is responsible for the simultaneous operation of Windows NT and the INtime™ real-time kernel on the same CPU.

In a standard Windows NT configuration, the bulk of the OS runs in the confines of a single hardware task. Additional hardware tasks are normally only defined to handle catastrophic software-induced failures such as stack faults and double faults where a safe, known environment is required from which to handle the failure. INtime™ transparently creates a hardware task for the real-time kernel and manages the switching and execution of the standard Windows NT hardware task, and the real-time hardware task. This approach guarantees the integrity of both the Windows NT kernel and the real-time kernel, and enables the successful operation of real-time processes even in the event of a total Windows NT failure (a "blue screen" crash).

The OSEM encapsulates the entire Windows NT priority spectrum in the lowest INtime, real-time priority level. This ensures that real-time threads and interrupts will always have priority over Windows NT threads and that the end system will operate deter-

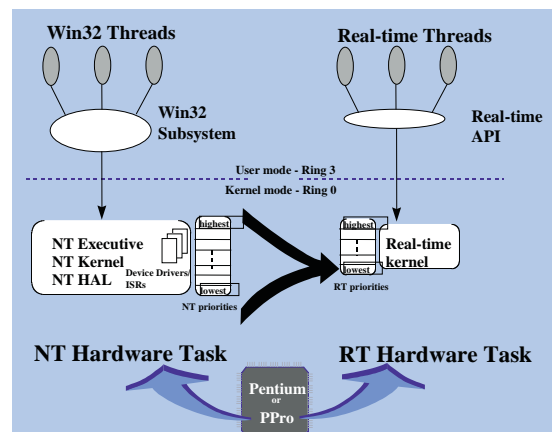


Figure 1. INtime OS Encapsulation Mechanism (OSEM)

ministically, regardless of Windows NT activity.

INtime provides a variety of memory protection mechanisms, relieving the developer of writing code in the Windows NT kernel-mode space. The result is improved reliability and robustness, as well as simplified programming and debugging. For each real-time process created on top of the INtime kernel, a separate 32-bit segment is automatically created. This segment is separate and distinct from those used by Windows NT and provides additional address isolation and protection not only between complex real-time processes, but between real-time processes and non-real-time Windows NT code. This memory protection is provided automatically to the INtime developer, using standard Windows NT flat memory model compilers (such as Microsoft Visual C/C++)..

Finally, the OSEM, by providing a clean, well defined interface, minimizes interaction with Windows NT to a few key areas, resulting in improved product reliability and simplifying compatibility between Windows NT releases.

The INtime OSEM provides:

- Transparent task creation and switching
- Memory protection:
- Real-time responsiveness

- Robust, seamless interface to Windows NT

When looking to Windows NT as a potential platform for industrial, real-time, or embedded systems, it's critical to assess what the application will demand in terms of reliability and determinism, both now, and in the future. Making an informed decision requires understanding some fundamental mechanisms of Windows NT, and how they affect the utility of the OS as a real-time operating system, and the basic approaches to solving this problem. The good news is that with products such as INtime™ from Radisys, developers will be able to leverage Windows NT and still achieve the reliability, robustness, and determinism which their applications demand. ■

Dave Kresta is the Marketing Manager for RadiSys' Embedded Software Operation. He has been involved in the design and development of software products for over 10 years. He holds a B.S in Computer Engineering from the University of Michigan and an MBA from Portland State University.

REFERENCES

- (1) "Windows NT as Real-Time OS?", Martin Timmerman and Jean-Christophe Monfret.