

RTOS uses virtual machine technology to share the CPU with Windows

by Kim Hartman, TenAsys

When applying Windows to time-critical instrumentation applications, a real-time operating system is necessary to satisfy the requirements for accurate and repeatable data acquisition and control. TenAsys introduced an RTOS that shares the CPU with Windows, using virtual machine technology.

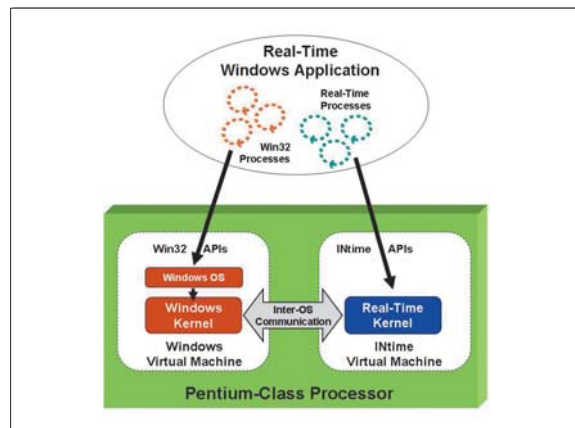


Figure 1. The RTOS and its processes run on the same hardware as the standard Windows kernel, sharing the CPU, memory, and I/O resources.

■ Embedded system designers have for years had to balance the need for a flexible, full-featured human interface or database system with the deterministic requirements of time- or mission-critical system functions. The trade-offs often result in two distinct computing elements being built into one system: one hardware subsystem to run Windows for the GUI, third party applications, and enterprise functions, and a second hardware platform to host a dedicated RTOS that controls the time-critical elements of the application.

Unfortunately, a second control computer adds substantially to the cost of hardware, and the complexity of software development for two different platforms. A “single-computer dual-OS” system, where one compute element hosts both the Windows system and the RTOS, reduces manufacturing cost and complexity, simplifies the coordination of Windows with real-time processes, and enables the use of a single set of development tools, cutting design costs and effort spent on a second set of engineering tools and staff. For this reason real-time Windows solutions that allow two execution environments to share a single CPU platform are becoming increasingly popular.

Some real-time Windows solutions utilize a Windows driver inserted in the Windows kernel, an approach that presents many reliability

and software design challenges. In contrast to this, a true dual-OS solution employs virtual machine technology where the non-deterministic application elements execute on a “Windows virtual machine”, and deterministic software executes on a “real-time virtual machine” (figure 1). To ensure efficient implementation of real-time threads, the virtual machine architecture supports direct access to I/O and memory, a fixed priority scheduling system with priority-inversion protection, and simplified interrupt-handling services. They can then create and deploy sophisticated real-time applications without having to write complex and cumbersome device drivers for access to real-time hardware, simplifying development and debugging of control and data acquisition algorithms.

The virtual machine approach to combining real-time task processing and Windows applications in the same system is quite different from approaches that install a real-time kernel in the form of a Windows device driver or subsystem. The device driver and subsystem models force real-time applications to operate as part of the Windows kernel. Kernel mode code has privileged access to the entire memory space, including the Windows kernel and other device drivers; it lacks address isolation and memory protection. A real-time thread running on such a system can easily overwrite

other processes: both real-time and Windows processes. Because such programming errors are difficult to detect in kernel mode, and can result in spurious but critical failures, achieving reliable operation through this method often requires extensive testing and debugging. Many such errors are not detectable until the system has been deployed in the field. Creating a complex, multi-threaded, real-time application to run inside the Windows kernel is contrary to the notion of building reliable, safe, and dependable real-time applications.

On the other hand, using a virtual machine to add real-time responsiveness to Windows, where real-time applications run in user-mode (ring 3), not kernel-mode (ring 0), enables the RTOS to maintain reliable operation of real-time processes in the event of a Windows crash. Each real-time process runs in a separate 32-bit protected memory segment. By taking full advantage of the virtual addressing hardware built into Intel architecture CPUs, a real-time Windows environment is able to provide the following protection and debugging mechanisms to real-time processes:

- ✓ code, data, stack, and heap are kept in separate, non-contiguous pages
- ✓ code is placed in read-only pages for extra protection
- ✓ paging is utilized for address isolation and pointer overrun protection

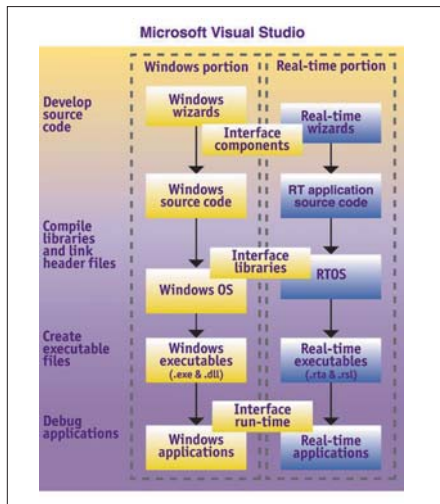


Figure 2. This diagram shows the basic steps in the software development process using Visual Studio.

Every real-time thread can be assigned distinct memory pages for its CODE, DATA, STACK, and HEAP memory regions. These pages are then separated by non-allocated pages of memory so that pointer overruns can be easily and quickly detected. CODE pages are further protected by marking them as read-only. These segments are distinct from those used by Windows and provide address isolation and protection not just between the real-time processes, but also between real-time processes and non-real-time Windows code.

The net gains from the virtual machine-based single-computer-dual-OS approach are elimination of redundant computer and communication hardware, faster communication and coordination between the real-time and Windows application parts, improved reliability and robustness, and simplified programming and debugging. Typically, applications with cycle times of 1 ms or slower are served quite well by

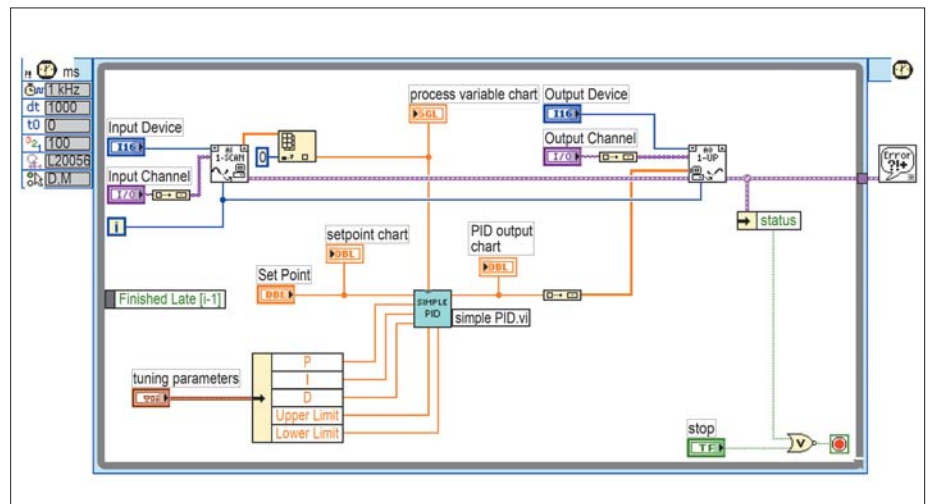


Figure 3. PID loop implemented with LabVIEW VI programming language.

this uni-processor/multi-OS arrangement and have been deployed on the current crop of desktop and industrial motherboard platforms (Pentium 4 class processors running at 1...3 GHz). However, some applications demand faster cycle times. For these applications the solution is to use one of the new dual-core processor chips. Dual-core processors easily support two operating systems at higher performance levels by dedicating one CPU to the RTOS. The CPU instruction cycles of the core dedicated to the RTOS are available 100% of the time to the real-time applications. The CPU cycles of the remaining core becomes the exclusive property of the Windows virtual machine. Contention for key CPU resources such as pipelines, cache, and the FPU are avoided. Coordination between the two processor cores is accomplished by using built-in inter-processor communication mechanisms, eliminating context switches. In this scenario, real-time interrupt latencies are reduced by an order of magnitude, from 10...30 μ s down to 1...3 μ s. Loop cycle times in

the 50...200 μ s range are able to operate with high precision and accuracy. The result is an order of magnitude improvement in the quality and bandwidth of control algorithms that can be deployed on a real-time Windows platform.

Dual-core real-time/Windows implementations can also benefit from the use of a single development environment. For Windows-based applications, the development tool of choice is Microsoft Visual Studio, a multi-language development environment that can be used to build and maintain a broad range of applications. Both Visual Studio 2003 and 2005 provide hooks for third-party plug-ins, providing a means to extend this IDE. By utilizing these hooks, third-party development tool providers can enhance the Visual Studio IDE to include features for tasks that go beyond the development of standard Windows applications, such as the creation of real-time embedded Windows applications, uniting the two parts of a Windows real-time solution under a single IDE. On

such a united Windows/RTOS platform, a Visual Studio real-time debugger plug-in can capture and identify real-time process faults resulting from divide by zero errors, bad pointer accesses, page faults, stack faults, and other CPU exceptions. Typically, only the operating system kernel and low-level device drivers are allowed to exist in kernel-mode. Standard Windows applications always execute in user-mode (ring 3), not kernel-mode (ring 0) which is reserved for the kernel. This prevents them from making modifications (deliberate or unplanned) to the code and data structures belonging to other applications or the kernel. For the very same reason, real-time Windows applications need to execute in user-mode, which also enables them to take advantage of the full range of features present in the Visual Studio debugger.

There is no performance penalty for executing real-time applications in the more reliable and protected user mode operating level. Software executes at exactly the same speed in user mode as it does in kernel mode. The only difference between these two operating levels has to do with the execution of special privileged instructions, fault (trap) handling, and memory protection; the very features that allow for a Visual Studio real-time debugger plug-in. A Visual Studio real-time debugger plug-in can be used to provide full access to breakpoints, source-level single-stepping, and watch variables for real-time Windows applications. The plug-in does not prevent one's ability to debug standard Windows applications; it adds the ability to work with applications in an alternate run-time environment.

Figure 2 shows the basic steps in the software development process using Visual Studio. The "interface" boxes represent components that must be added to the development system and run-time to support the creation and execution of real-time Windows applications within the IDE. The diagram shows two parallel paths of development, from editing code through debugging, both encapsulated within Visual Studio. With proper integration, these two paths are nearly indistinguishable from one another, differentiated only by the differences in the two environments in which the application parts execute.

The goal is to make the mechanics of the development process identical, in order to optimize the use of the developer's knowledge and skills associated with a single development tool. Project wizards, which can be integrated into the Visual Studio development environment, enable developers to quickly build real-time applications. The wizards for the real-time Windows path can be designed to guide one through the many design decisions and then automatically generate usable code upon which to build the real-time parts of the application. An example of a fully-featured real-time oper-

ating system that fulfils the requirements stated above is INtime, from TenAsys. INtime operates by creating a virtual machine environment to separate the real-time and Windows components of an application, which can share a single-core processor or a dual-core processor. Applications written for the INtime RTOS execute with guaranteed determinism as fully-protected user-mode processes in parallel with

the Microsoft Windows operating system on standard PC hardware. Because real-time application code executes in user-mode the INtime environment is immune to application faults that crash kernel-mode driver solutions. INtime 3.0 comes with all the support necessary to develop real-time software subsystems using Visual Studio on single-core and dual-core processors. ■