

# Windows-based systems and the Win32 API

*The recent past has seen Windows-based systems grow in popularity, penetrating deeper and more widely throughout the industrial world. The main driving factors have been lower costs of ownership of Windows-based PCs and lower development costs with the Win32 API, familiar to hundreds of thousands of programmers. More recently, hard real-time extensions to Windows NT have allowed real-time developers to benefit from these cost-reduction trends. The influence of Windows-based systems is likely to continue to increase. This paper describes VenturCom's RTX hard real-time extensions for Windows NT and RTX real-time extensions for Windows CE.*

## EXTENDING WHAT WE ALREADY HAVE

In the dark ages of real-time programming, each system was a solution unto itself. Hardware was so expensive, space was at such a premium, and the problem set thought practical to tackle so small that implementing a "from the ground up" design each time made some sense. Systems were coded entirely in assembly language and were written for a stand-alone environment with no underlying operating system providing support for common facilities. Gradually, the lessons learned from other areas of programming have made their way into the real-time arena, tempered always with the unique requirements that real-time programming (or "real world programming") imposes. So we have come to accept higher level languages when they matured enough to be as efficient and flexible (and malleable) as was necessary. Real-time operating systems have found their place, providing a rich core of common facilities such as task scheduling, memory management, inter-program communication, and synchronization, to name a representative few, to programmers writing for their specific platform, leaving these programmers free to focus on the unique aspects of their particular problem. Now, with the increasing prevalence of Windows systems at all levels of industrial life, and considering the dramatically reduced cost of a Windows-based system, it is unsurprising that developers looked to implement real-time solutions on these platforms.

Windows is many things, some of them all at once, but one thing it is not (at least right after the shrink wrap is removed) is ready for real-time. It has too few priority levels, its scheduling decisions are rather opaque and somewhat non-deterministic, its interrupt activity is not subordinated to thread priorities, and it does not satisfy the response time requirements of a hard real-time system. VenturCom's RTX product, real-time extensions for Windows NT and Windows CE, addresses these problems, allowing a developer to deliver hard real-time performance systems on commodity architecture. RTX extends Windows by providing a Real-Time Application Programming Interface (RTAPI).

Customer code requiring hard real-time response runs in a Real-Time SubSystem (RTSS), sharing the kernel's address space and potential to directly access all I/O hardware. RTSS implements facilities for process and thread scheduling and management, inter-process communication between both RTSS and Windows processes, interrupt and exception handling, object management, clocks and timers, among others. The most important feature of RTSS is that it is a real-time subsystem. It has been designed with the deterministic requirements of hard real-time systems in mind. Its code paths eschew excessive iteration, data structures have been designed to keep lower priority tasks from interfering with higher priority ones, and every design decision has been made with the idea that deterministic real-time response is a paramount requirement. Numerous exercises and tests at VenturCom and our customers sites have proven that a stock Windows system can deliver guaranteed real-time response to an external event in under 50  $\mu$ sec while it is being used to perform other interactive tasks.

## RTX API AND RTX PROGRAMMING

### RTX API

The RTX API provided by VenturCom contains a rich subset of familiar Win32 procedures, real-time procedures, and C runtime facilities. A programmer familiar with the Win32 API from any of its incarnations (e.g. Windows NT, Windows CE, Windows 95, Windows 98, etc.) will recognize its heritage.

At the present time, RTX API is made up of over 200 calls divided among memory management, clocks and timers, synchronization, input/output, and thread management functions, including functions from the standard ANSI C runtime libraries. To give an idea of the breadth of functionality provided, as well as to demonstrate the close (in most cases, identical) correspondence between RTX API routines and standard Win32 routines, we consider a few calls:

RtOpenSemaphore returns a handle of an existing named semaphore object.

```

HANDLE
RtOpenSemaphore(
    DWORD    dwDesiredAccess,
    BOOL     bInheritHandle,
    LPCTSTR  lpName
);

```

The return values and semantics of `RtCreateSemaphore` correspond exactly to the Win32 `OpenSemaphore` call. Note that even though RTX API currently makes no use of the `dwDesiredAccess` and `bInheritHandle` parameters, they are provided both for possible future use and for compatibility with Win32.

`RtWritePortUchar` writes a one-byte datum directly to an I/O port.

```

VOID
RtWritePortUchar(PULONG PortAddress, ULONG
Value);

```

This is an example of a VenturCom-supplied real-time extension to Win32's semantics.

```

ExitProcess ends a process and all of its threads.
VOID ExitProcess(UINT uExitCode);

```

This corresponds exactly to the Win32 function of the same name. In fact, when an RTX API-compliant program is linked to run in the Win32 environment, the Win32 function is the function that is called, and when the identical object is linked with a different set of libraries to run in the RTSS environment, then it is an RTSS function providing the same functionality, which is called. This ability to develop programs under the Win32 environment and simply to relink to achieve hard real-time performance is one of the major strengths of our approach to providing real-time extensions under Windows.

`sinh` calculates and returns the hyperbolic sine of its argument.

```
double sinh( double x );
```

This is a standard libc function. Unlike DDK programs or other kernel level code, RTSS programs have full access to the FPU, and the RTX subsystem ensures that every thread sees exactly its view of the FPU.

#### *RTSS Access from NT Drivers*

RTX enables NT device drivers and kernel-level components to access RTSS objects - similar to the way Win32 applications do - by providing the RTK (Real-Time Kernel) API. When RTSS is active, a Windows NT driver linked with the RTK API static library can open and create RTSS shared memory, semaphores, mutexes, and other objects, perform wait/release operations on synchronization objects, and so forth. In addition to flexibility, RTK API calls offer lower overhead and latency than Win32 operations on RTSS objects, although, running under NT control as they are, no determinism guarantees can be made.

#### **Developer's Perspective**

Source code portability of RTX API programs between Win32 and RTSS works to a developer's advantage. Typically, one would start development and initial debug of an RTX application in the Win32 subsystem using standard tools, like Microsoft Developer Studio, then re-link the program for RTSS, and run it in the real-time environment. Although RTSS programs are readily compatible with source-level debuggers like Microsoft Windbg or Numega Softice, using the Developer Studio in Win32 is vastly easier. The differences in behavior between Win32 and RTSS will typically be limited to scheduling, timing, priority settings, and, of course, performance.

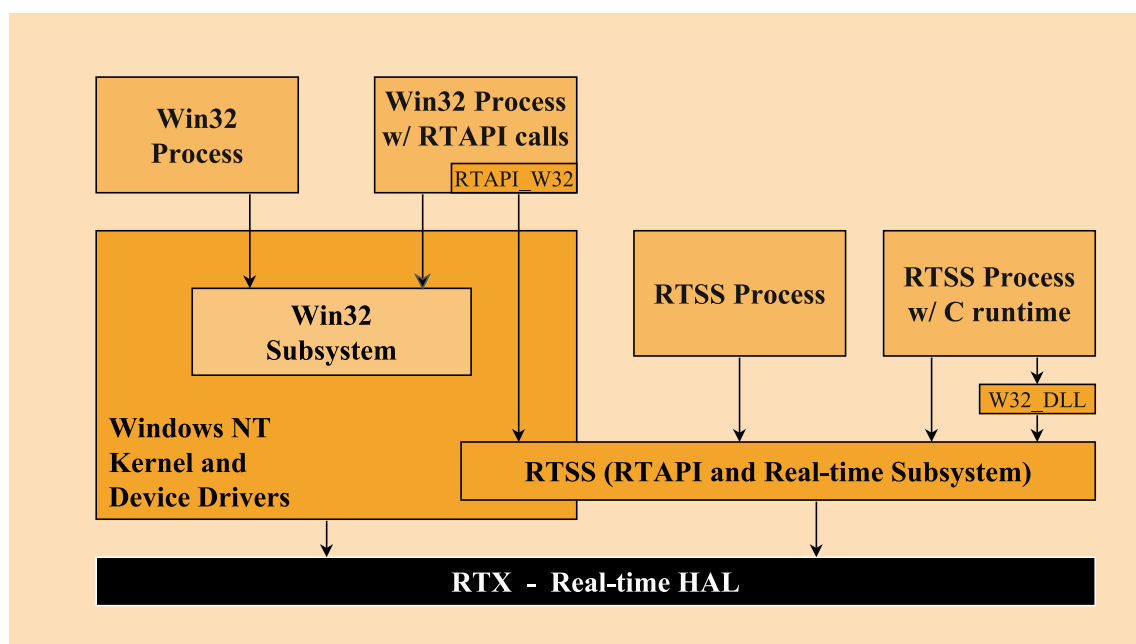


Figure 1. RTX Architecture

When the program is moved to RTSS, a developer may want to use RTSS DLLs. RTX supplies one such library: W32\_dll.rtss, providing additional Win32 calls, used to support a subset of C runtime library calls layered on top of Win32.

RTX documentation specifies which RTSS calls are deterministic. A typical application would use non-deterministic calls during its setup phase, then proceed with deterministic execution, finally invoking non-deterministic calls again when terminating.

## RTX Design Basics

The salient elements of RTX (see Figure 1) are:

- Real-time HAL that delivers NT/RTSS interrupt isolation, fast timers, and NT shutdown management.
- The RTX\_RTSS driver, providing the real-time scheduler, object manager, Win32-RTSS IPC, and other RTX facilities.
- Rtpapi\_w32.dll, providing soft real-time support for Win32 processes and Win32-RTSS IPC.

The real-time HAL is fully compatible with most common PC platforms. The driver is written by the DDK rules, so RTX requires no changes to the NT kernel, executive, or device drivers. Thus, VenturCom's lack of access to NT kernel source code turns to strength, as RTX needs no or minimal changes when new versions of NT become available.

## User Interrupt Management

RTX supports interrupt management, with the same API for Win32 and RTSS programs. Application threads attach to interrupts and process them in full thread context: RTX creates an interrupt thread (IST) to process each interrupt attached by an application. Thread-context interrupt processing has a number of advantages over traditional Interrupt Service Request routines (ISRs), while adding relatively trivial overhead:

- Interrupts can be processed in the order of handler thread priorities, rather than hardware priorities.
- Thread context imposes no correctness limitations on the interfaces used during interrupt handling, eliminating one of the major causes of crashes in real-time systems - improper calls from ISRs. ISTs, although requiring careful coding for performance, are much less crash-prone than ISRs.
- Thread context greatly eases debugging, compared to ISRs

## Real-Time HAL

### Interrupt Isolation

The primary function of the HAL is interrupt isolation between NT and RTSS, described below in the Performance section.

### Protection from NT Crashes

In addition to interrupt management and fast-timer services, the real-time HAL also provides NT shutdown

management. An RTSS application can attach an NT shutdown handler for cases where NT performs an orderly shutdown, or crashes, producing a so-called Blue (Stop) screen. An orderly shutdown allows RTSS to continue unimpaired and resumes when all RTSS shutdown handlers return. In a blue-screen shutdown, RTSS shutdown handlers run with certain limitations, unable to call NT services (e.g., for memory allocation) or to handle faults. In practice, it means that a shutdown handler should clean up and reset any hardware state, possibly alert an operator, or switch to a hot spare.

### Fast Timer Support

On all PC platforms real-time HAL provides clock resolution of 1µsec or better, and timer period of 100µsec or better. When RTSS is not used, there are no timing differences between a real-time HAL and a regular HAL systems.

## RTX Performance

### Sources of Interrupt Latency

One of the most important characteristics of an RTOS is the interrupt response latency. The most significant software-related cause is interrupt masking by the NT kernel and device drivers via NT IRQL (Interrupt Request Level) calls, which re-program the PIC (Programmable Interrupt Controller) chip of a PC to mask interrupts below a certain level. Such masking can last for several milliseconds. There is also processor-level interrupt masking which disables all interrupt on the CPU. Processor-level masking is only done by the Windows NT kernel code (e.g., interrupt processing or thread scheduling) and some special NT drivers for periods of up to 50µsec. Finally, interrupt-processing overhead also contributes to interrupt latencies.

### RTSS Latency Reduction Techniques

The real-time HAL entirely eliminates IRQL-related latencies by re-programming the PIC. When RTSS is running, all NT interrupts are masked; when NT is running, RTSS interrupts can not be masked by NT. RTX also dynamically eliminates some unnecessary processor-level interrupt masking by the NT kernel and drivers via the RTX Dynamic Hook driver, which

### TYPICAL LATENCIES FOR A 266MHZ PENTIUM II, µSEC

Interrupt thread	9 - 38
Timer interrupt	10 - 43
Context switch (yield)	3.5
Context switch (semaphore)	5.5
Thread priority change	3
Thread yield	3
Acquire semaphore, uncontested	1.5
Acquire semaphore	5
Win32-to-RTSS semaphore call	50

Table 1. RTX 4.2 Performance Results

Ad Ventur Com

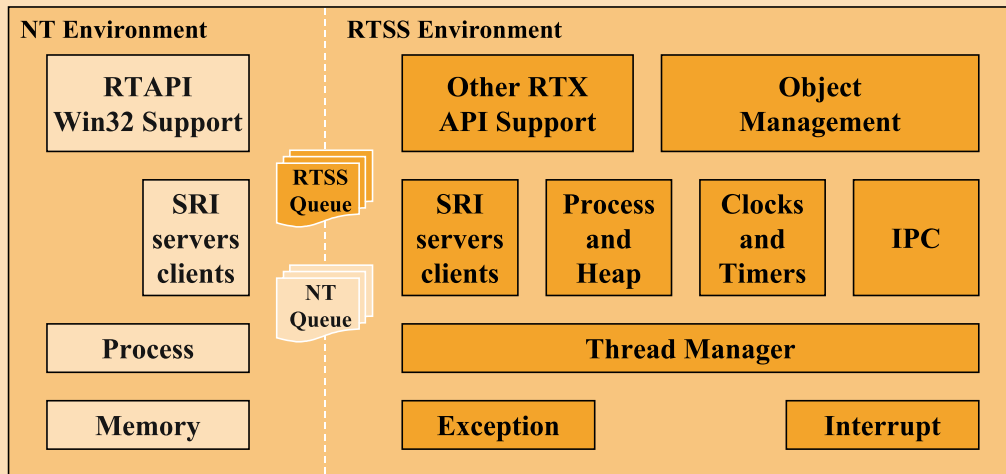


Figure 2. RTSS Detailed Architecture

replaces the Intel x86 cli/sti instruction pairs by jumps to lock/unlock routines which use IRQL calls instead. Finally, the real-time HAL has been modified to avoid long-latency processor-level interrupt masking.

#### Determinism and Platform Factors

The result of this design is deterministic response, with the worst-case interrupt latency of 40-100µsec on a typical PC platform. The hardware platform matters, as some video cards and DMA devices may take over the PCI bus for unusually long intervals, impacting performance. RTX tools help qualify a platform's response time, and pinpoint highest-latency drivers and peripherals.

#### Performance Metrics

Table 1 presents selected performance numbers for the latest release of RTX. See the OMAC Real-time Evaluation Report ([http://www.vci.com/prod\\_serv/nt/rtx/arc\\_test\\_results.html](http://www.vci.com/prod_serv/nt/rtx/arc_test_results.html)), a joint study by GM, Hughes, and ATR for a thorough evaluation of RTX performance.

## RTSS - THE REAL-TIME ENVIRONMENT

### RTSS Design Basics

The major modules of RTSS (see Figure 2) include the SRI, object manager, thread manager, process and heap management, RTX API support, exception handling and others. The thread manager implements pre-emptive scheduling with 128 priority levels. An RTSS thread runs ahead of any NT threads, until it suspends, or a higher-priority RTSS thread becomes eligible to run. RTSS threads and kernel code are fully preemptive.

### Win32-RTSS IPC

RTX Win32 processes can access real-time objects, which live in the RTSS environment (e.g., mutexes, semaphores, and shared memory). Wait requests for RTSS synchronization objects are queued in thread priority order. Win32-RTSS IPC allows tightly integrated

applications where hard real-time processes run in the more resource-intensive RTSS environment, and the rest of the application runs in the Win32 subsystem.

### Service Request Interrupt (SRI) Design

An important architectural feature of RTX is its lockless interrupt-driven interface between NT and RTSS. This clean architectural separation allows easier port of RTSS to various environments (e.g., Windows CE, or a multiprocessor system), while ensuring a fast and robust implementation. The NT side of the RTX driver and the RTSS environment communicate by inserting commands into one of the two buffer queues (one in each direction) and initiating a Service Request Interrupt (SRI) to request service by the other side. A request is executed by a server thread and a reply message is posted in the other buffer. A typical NT-to-RTSS request is an IPC operation like WaitForSingleObject or a Release operation on a RTSS object. A typical RTSS-to-NT operation is a memory allocation or a file I/O request.

### Structured Exception Handling

RTSS compatibility with Win32 includes Structured Exception Handling (SEH), providing the SEH API (RaiseException, SetUnhandledExceptionFilter, et al), as well as try/except and try/finally semantics. Thus, C++ applications and middleware using throw/catch exception handling are fully RTSS-compatible.

The RTSS implementation of SEH is designed for real-time: a thread not taking an exception is not adversely affected, since all processing of software exception is done in application mode with interrupts enabled. A hardware exception causes an interrupt, but all subsequent processing is also in application mode. Contrast this with Win32, where a software exception generates an interrupt, then doctors the trap frame in kernel mode, or edits the trap frame with interrupts disabled.

## RTX FOR WINDOWS CE

VenturCom's Real-time Extension (RTX) 4.2 for

Windows CE extends the capabilities of Windows CE and enables developers to create real-time Windows CE applications that are source code-compatible with Windows NT RTSS and Win32 subsystems. Real-time applications can be written for a full spectrum of devices from enterprise servers running Windows NT to soft programmable logic controllers running Windows CE.

RTX 4.2 for Windows CE includes the RTX API library, a console library for fast prototyping of benchmarks and compatibility with Windows NT, high-speed clocks/timers driver, SRTM benchmarking utility, and RTX sample drivers. Key features that the RTX API adds to Windows CE include:

- Semaphores
- High resolution clocks and timers
- Flexible interrupt support
- Contiguous memory allocation
- PCI bus Plug 'n Play support
- Console I/O using printf

To facilitate porting and developing benchmarks for Windows CE, a console library is provided. This library supports the printf API and allows programs that provide a main to be linked and run under Windows CE.

## OUTLOOK

### **Multiprocessor RTX**

In the near future VenturCom intends to provide a multiprocessor version of RTX, initially supporting a dedicated-processor model, where RTSS "owns" one processor, with NT running on the rest of the processors. Such a configuration is especially attractive because:

- It improves RTSS response, as PIC re-programming and other NT-preemption work becomes unnecessary on the RTSS-only processor.
- NT performance is, likewise, improved, as it suffers no preemption by RTSS interrupts.
- Cache access performance is improved, since NT and RTSS are no longer competing in the same arena for access to memory.
- We avoid preempting NT in an arbitrary context, where it may be holding kernel spinlocks, possibly affecting all NT-running processors.
- Implementation is relatively straightforward.

More symmetrical configurations, with RTSS and NT sharing several processors, may be appropriate for some customer application. The basic RTX design is compatible with such systems, and VenturCom may implement them in the future.

### **RTSS on Windows CE**

Recently VenturCom has begun the porting of the RTSS subsystem to Windows CE, achieving latencies

comparable to those of the NT implementation. An RTX application will deliver hard real-time performance on RTSS Windows CE, while still being source-level portable to NT Win32, and NT RTSS. This allows RTX customers unparalleled portability and scalability.

## RTX AS A PLATFORM

RTX provides a solid platform for development and is an integral part of many NT-based real-time products, some of which are listed on the VenturCom Web page. Among RTX-based applications and development efforts - primarily focused on industrial automation, control software, and signal processing - are: RCS+ systems from Advanced Technology & Research Corporation, OPC (OLE for Process Control) industrial automation software from FactorySoft; Paradym-31 - a soft-logic product from Intellution/Wizdom; a real-time Java-based control engine developed by Object Automation, laser shape-cutting systems by Cleveland Motion Controls, and numerous others.

### **RTX Futures**

VenturCom will be using our real-time experience to further enhance the development environment available to real-time programmers. We work closely with our real-time partners to provide the appropriate feature set for their applications. RTX is an evolving product. The most up to date information concerning RTX is available at <http://www.vci.com/>. With Microsoft's commitment to enhance Windows CE to provide the hard real-time capabilities previously only available via extensions such as RTX, our goal shifts from one of providing the necessary facilities to one of standardizing an easy to use interface. Without promising anything, it seems safe to say that future releases will extend the RTX API to encompass more of the Win32 programming model. Additionally, VenturCom will be providing real-time distributed control and data exchange services and distributed data flow based on the publisher/consumer model, and a range of device drivers via our DCX offering (more information is available at [http://www.vci.com/prod\\_serv/dcx/overview.html](http://www.vci.com/prod_serv/dcx/overview.html)).

---

*Chris Jones, a graduate of the University of Massachusetts (Amherst) with a BS in Mathematics, has more than 20 years of experience in systems programming on a variety of platforms ranging through Data General minicomputers, Honeywell Multics mainframes, Symbolics Lisp Machines, and Kendall Square Research and BBN multiprocessor systems. He hopes Windows systems will better withstand his focus on them. Chris works as a principal engineer at VenturCom Corp.*

*Mike Cherepov, who has BS and MS Computer Science degrees from Illinois Institute of Technology, has been programming professionally for over 15 years. For most of that time he has been writing operating systems for platforms ranging from DSP chips to large-scale 64-bit multiprocessors, before discovering the paramountcy of the x86 architecture and Win32 API. Mike works as a principal engineer at VenturCom Corp.*