# Using the Windows NT Operating System for Soft Real-time Control - Separating Fact from Fiction

White Paper


***"Several vendors opted to collect one data point on the Windows NT operating system, present it out of context, and then claim that it was unacceptable for control. These vendors have committed to a path of proprietary extensions that will limit their ability to adopt standard Microsoft technologies."***



Rockwell Automation
Allen-Bradley
Rockwell Software

# Introduction

## *Why We Chose the Microsoft® Windows® NT Operating System*

Rockwell Automation was faced with a critical decision regarding our operating system strategy for open PC-architecture-based soft control. Various alternatives existed in the marketplace including:

- Real-time operating systems (iRMX™, QNX®, OS-9®)
- DOS extenders (ControlWare™, PharLap TNT™, etc.)
- Extenders of the Windows NT operating system (Radisys INtime™, VenturCom™ RTX™, Imagination Systems Hyperkernel™)
- Standard operating systems (Windows 95®, Windows NT®, IBM™ OS/2®, SCO® UNIX®).

Various alternatives had strategic advantages such as:

- Downward scalability
- Real-time performance
- Technology access
- Development productivity
- Third-party compatibility
- Market acceptance

Very early on, market factors drove us to a Win32-based solution. Based on customer feedback, the Microsoft Windows platform was the only operating system with sufficient software vendor support to be truly thought of as open. To our customers, “open” meant that end-users had choices from multiple software vendors. The overwhelming momentum of Microsoft Windows in the automation software market has made it the clear choice for open applications. Another key aspect customers mentioned repeatedly was that software from multiple vendors had to be able to coexist on the platform in a well-behaved manner. The goal of open control became critical to our decision making.

This paper summarizes the reasons we chose the standard version of the Windows NT operating system as our primary soft control platform. It also provides the results of our investigations into the performance and appropriateness of using the Windows NT operating system for control applications. Surprisingly enough, selecting to use the standard Windows NT operating system has become somewhat controversial, in light of the recent product positioning statements being made by our competitors. As soft control emerges in the marketplace, a great deal of information has been spread regarding the performance, design, and reliability of the Windows NT operating system in an attempt to drive end-users to the conclusion that the Windows NT operating system is totally inappropriate for soft control applications.

Though much of the information has some basis in fact, it doesn't have the proper context, leading end-users to erroneous conclusions. Furthermore, equivalent scrutiny of the proposed alternatives to using the Windows NT operating system is lacking. This paper attempts to provide an understanding of the performance results of the Windows NT operating system and provides a discussion of key concern areas in the proposed alternative real-time extension technologies. Based on our testing and understanding of the Windows NT operating system and of alternative real-time extension technologies and further based on our experience with plant floor control requirements, Rockwell Automation has chosen an unmodified Windows NT operating system to provide users with all of the benefits of a truly open system.

# Background

## *Deterministic Control in Hard and Soft Real-time Systems*

The issue of deterministic control and the concepts of *hard* and *soft* real-time systems lie at the heart of the soft control debate. To begin the discussion, consider a typical control system. In general, all control systems can be thought of as receiving input data, performing some form of processing, and then sending output data. The control sequence may be:

- Started by an external event, such as an I/O scanner interrupt
- Scheduled to occur at a fixed time interval, such as once every 20 milliseconds
- Occur in a continuous loop, running as fast as possible

For example, the Allen-Bradley PLC-5 processor supports all three of these mechanisms with:

- Process-Input-Interrupt task (PII)
- Selectable-Timer-Interrupt task (STI)
- Main Control Programs (MCPs)

Control is said to be deterministic if the controller can guarantee that processing occurs within a fixed range of time. For instance, on a given controller, a 20 millisecond periodic task may actually execute in 20-24 milliseconds. For this same task, we might also say that the scan time had a repeatability range of 4 milliseconds. The design is called deterministic if the worst-case performance is known. The controller may also contain a watchdog thread to verify that control has occurred in the proper time interval. If the interval is exceeded it may fault the processor and place the I/O in its fail-safe condition.

The rather esoteric issues of hard and soft real-time arise from the ideas of theoretical vs. pragmatic worst case determinism. On the one hand, hard real-time systems attempt to guarantee worst-case performance by having tightly-controlled designs. The approach with hard real-time systems is to be able to theoretically prove worst-case performance. Furthermore, in hard real-time systems, the burdens of proof rest on the operating system as well as on the control applications and the I/O subsystems.

Even in hard real-time systems, the operating system can only guarantee determinism to its highest-priority real-time application task or interrupt. All other tasks can be pre-empted by this highest priority task, so their determinism depends on both the operating system and the worst-case code path length of the highest priority task (worst-case time to complete programmed operations). This degradation of determinism cascades downward to lower priority tasks. Consequently, the selection of a real-time operating system by no means guarantees real-time system performance. There is no established standard that prevents virtually any existing operating system from referring to itself as real-time.

On the other hand, the term soft real-time system has evolved to describe controllers whose worst-case performance has been observed, not proven. In this case, the complexity of the environment or the use of off-the-shelf components makes it impossible to provide a rigorous design proof. Instead, the system is tested in the laboratory under simulated stress, and the performance is observed over a long period of time. Design confidence results by devising a representative stress test that exceeds real system stress conditions and observing the system for a suitably long time to ensure an accurate analysis. These same types of tests are also required for hard real-time systems for design validation.

In soft real-time systems, the incorporation of components not specifically designed for real-time determinism may limit system performance. However, the goals of high performance and real-time determinism tend to be coherent, so systems that were not designed to be deterministic but to be high performance, often work well as in control systems.

## *Understanding Deterministic Control Requirements*

In practice, no real control system provides absolute repeatability, and fortunately, most real control processes tolerate some level of scan variability. Over the years, Rockwell Automation has developed a great deal of expertise in understanding scan time and scan repeatability requirements for various types of control problems. Though many applications require very tight closure, a broad range of common applications is more tolerant of variations. Two concepts that are useful in understanding scan time requirements are process time constants and machine cycle times.

The theory of process time constants is derived from the discipline of continuous process control, and is generally associated with the proportional integral derivative (PID) algorithm. When modeling a system as a first order process, you can estimate the process time constant with a simple open loop test. Given a step change in the controller’s output, the time it takes for the process variable to achieve 63% of its eventual lined-out response is an estimate of the process’ time constant. Physical processes vary dramatically in their time constants. Where the speed of the high-pressure spool of an industrial gas turbine may have a 420-millisecond time constant response to movements in its fuel valve, the pH of a water treatment settler may have a 2-day time constant response to changes in its feed composition.

A general rule-of-thumb for determining scan time requirements for process control loops is that the maximum scan time should be no more than one-sixth of the process time constant. Consequently, our hypothetical industrial gas turbine should be scanned at least every 70 milliseconds, while our settler may require scanning only once every eight hours. Also, if written in variable time interval form, the PID algorithm is actually quite robust in handling moderate to severe scan overruns. Infrequent overruns of more than 50% of the base scan rate are easily tolerated.

An additional consideration in PID control is the need to minimize dead time and dead time variation introduced in the measured value and output values manipulated by the PID algorithm. Extremely high-speed applications require specialized, tightly coupled analog modules, typically accessed directly through an I/O backplane.

Machine cycle time rules provide similar guidelines for the proper application of discrete control applications. With discrete control, the scan time determines the degree to which the input and the output events are synchronized. A general rule of thumb is that scan time variations should not exceed 5% of the machine's cycle time. For many machines this may require fast, dedicated controllers, however, many machines have less severe requirements. Additional complexity may be involved when large interposing relays, solenoid valves, or other electromechanical devices introduce delays in the control action. These requirements vary significantly with application type.

### *General PLC Performance Capabilities*

Another useful way to understand control system requirements is to look at the capabilities of modern controllers. Programmable Logic Controllers (PLCs) have been in use for more than two decades, continually evolving to incorporate the rapid improvements in microprocessor technology. Today PLC processors typically can support scan times in the 5-10 millisecond range, with scan repeatability ranges of 1-4 milliseconds (depending on the PLC model) for selectable-timer-interrupt style tasks. Dedicated PLC processors can provide even faster scan times for special applications. Certain modern PLC processors also support "interrupt" tasks, such PII of the PLC-5 processor, typically that enables a program to detect and act on an event within 0.5 to 2.0 milliseconds.

Performance characteristics of PLC processors depend on architectural and application factors. The controller's interrupt activity for servicing system timers and communication interfaces impacts scan repeatability. Other intermittent events that impact repeatability include:

- Processing serial input
- Handling keypad entry
- Performing error handling for communications
- Servicing network requests
- Performing controller housekeeping

The design of the PLC application code can also impact scan repeatability. PLC processors that support conditional branching, looping and subroutine calls will have variable code path lengths that lead to natural variations in scan times. Typical scan times for large real-world PLC applications range from 15-100 milliseconds. Less time critical MCP-style tasks often tolerate scan time variations of 25-50 milliseconds. For many PLC applications and simple General Motion Control (GMC) applications, these performance variations do not present a problem.

For higher-speed applications, STI-style tasks and/or small dedicated PLC processors are more prudent selections. These applications usually contain relatively simple programs and require tight coupling to their I/O hardware. In addition, there are certain applications that either require better performance than can be provided by general purpose PLC processors. Three such application areas are:

- advanced applications of General Motion Control (GMC)
- many applications of Computer Numeric Control (CNC)
- high-speed coordinated drive control

In addition to the controller, the I/O modules, adapters, racks, and cabling are key components of all control systems that impact scan times and repeatability. Many I/O systems can introduce scan time and repeatability issues that exceed those seen in the controllers. For example, a 10Mbps Ethernet™-based I/O system that uses a master/slave protocol to scan networked I/O adapters will typically have 8-10 millisecond round-trip scan times. One source of scan time variation in this system would be recovering from a transmission error detected by a cyclic redundancy check (CRC) for packet integrity. Even if the request succeeds in one retry, it introduces 8-10 milliseconds of variability in the I/O processing scan.

## System Integrity and Fail-safe I/O Operations

Beyond deterministic performance, three other critical issues impact the safe use of any control system:

- System integrity
- Rapid fault detection
- Guaranteed fail-safe I/O operations

During normal operations, control scanning occurs with some typical pattern of variation. However, it is absolutely critical that the I/O enters the selected fail-safe output state should any of the components in a control system fail. Since there are many potential failure modes for each of the control system's components, this is a very non-trivial exercise.

Key failure modes of a controller include:

- halting (by executing corrupt or defective code)
- power supply failure
- receiving a non-maskable interrupt (NMI) due to a memory parity error
- binding in an infinite loop (by executing corrupt or defective code)
- generating an unexpected processor exception (by executing corrupt or defective code)

To improve system integrity, PLC processors are designed to do extensive system integrity checks continuously and during their housekeeping cycles so they don't continue to run after a component failure. Processor exception-handling traps many of these failure modes. Typically, PLC systems generate a critical fault (i.e., the red light fault of the PLC-5 processor) in response to these types of exceptions.

Additionally, the I/O interface scanners must be designed to reliably detect when the PLC processor fails and place their outputs in the fail-safe condition. Controller failure detection is usually performed using a watchdog timer, which is located in the I/O interface scanner module, that must be periodically reset by the controller to maintain I/O operations. This is not enough. Because the I/O interface scanner module may also fail the I/O link adapters must also have watchdog timers for the I/O interface scanner.

Since the failure modes of the control processor include immediate halting, the I/O subsystem cannot depend on any action from the PLC processor to help it determine when to go fail-safe. All microprocessors can potentially clear/disable interrupts and halt if they execute corrupt code in a privileged mode. Since stack corruption prior to a return instruction can vector the processor anywhere within the kernel, misaligned code may be executed and the processor may simply halt.

## Summary

This section of the paper has provided some background and context information for understanding control system requirements and for assessing the performance, design, and reliability of the Windows NT operating system. When we later discuss the performance results we achieved with the Windows NT operating system, we will be able to discuss them within the context of actual control system requirements and to compare them with actual PLC capabilities as a benchmark for deterministic control. In addition, we covered some key issues regarding system integrity and fail-safe I/O operations that are every bit as important in assessing the Windows NT operating system for "soft" real-time performance.

In the next section, we present the test results of the Windows NT operating system in detail and general conclusions about the appropriateness of a Windows NT operating system for soft control applications.

# Assessing the Windows NT Operating System

## *Dispatch Algorithms of the Windows NT Operating System*

Before discussing the soft real-time performance tests of the Windows NT operating system, we need to review its basic dispatching algorithms. Essentially, the Windows NT operating system dispatches three types of objects:

- **Interrupt service routines (ISRs)** ISRs are driver routines primarily dispatched in response to device interrupts.
- **Deferred procedure calls (DPCs)** DPCs are driver routines queued by ISRs to perform less time-critical processing.
- **Dispatched threads of execution (threads)** Threads are the primary execution unit supported by the scheduler in the Windows NT operating system. The kernel, device drivers, and processes can own threads.

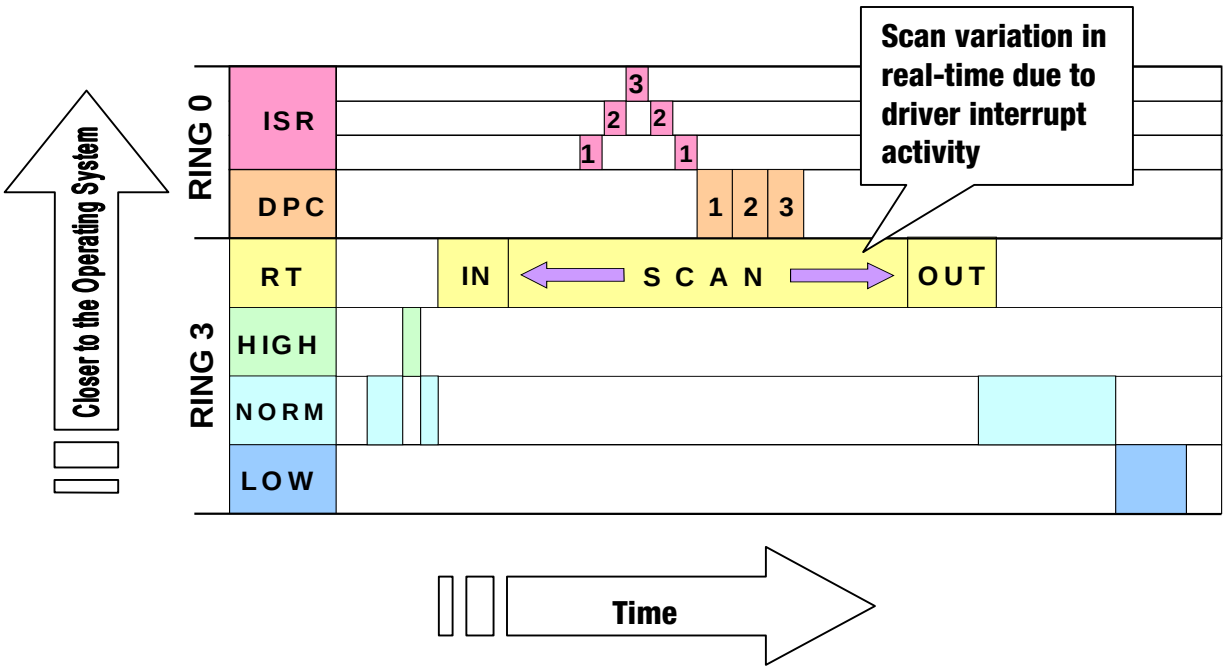


**Figure 1: Simplified view of dispatch algorithms in the Windows NT operating system. Running Soft Control in the Real-time (RT) Class prevents other applications from severely affecting determinism. However, lower priority applications do affect determinism when they cause Ring 0 device drivers to perform operations such as reading and writing to disk.**

### *Interrupt Service Routines*

An ISR has an associated interrupt priority level called the IRQL and is related to hardware interrupt request levels. The Windows NT operating system uses 32 interrupt levels, numbered 0-31. The operating system's Hardware Abstraction Layer (HAL) usually maps the 16 PC hardware interrupt request levels to IRQLs 12 through 27 (see Table 1). When hardware interrupt requests occur, the virtual programmable interrupt controller (VPIC) located in the HAL delays their dispatch if the IRQL assigned to the interrupt is less than the IRQL of the currently executing object.

When interrupts are dispatched, the IRQL is raised to the value associated with the interrupt. However, an ISR can raise and lower the IRQL while the ISR runs. When the ISR returns, the IRQL is restored to its original value. If interrupts are pending when the IRQL is lowered beneath the ISR's assigned IRQL level, then the highest priority pending interrupt is dispatched immediately.

ISRs can prevent pre-emption during their execution by raising the IRQL to a high level (KeRaiseIrql), and later lowering it (KeLowerIrql). Though it is considered a violation of the device driver rules of the Windows NT operating system, any privileged (ring 0) routine can also disable interrupts (the x86 CLI instruction) and perform time critical operations rather than raising and lowering the IRQL. Interrupt processing resumes when the task re-enables the interrupts (the x86 STI instruction). This is considered an acceptable practice for extremely short code path lengths if there is no other solution available to the driver writer (e.g., dealing with problem hardware, which requires very tight timing, and IRQL manipulation is ineffective.)

**Table 1: Level Assignments of the Windows NT Operating System**

| Interrupt Level (IRQL) | Assignment |
|---|---|
| Level 31 | Hardware error interrupt (NM) |
| Level 30 | Power failure interrupt |
| Level 29 | Inter-processor doorbell |
| Level 28 | Clock interrupt |
| Levels 12-27 | Mapped to PC hardware IRQLs 0-15 |
| Levels 4-11 | Unassigned |
| Level 3 | Software debugger interrupt |
| Levels 0-2 | System synchronization soft interrupts |

### *Deferred Procedure Calls*

While the ISR is running, it may post a request for a DPC to run later on its behalf. Drivers may also ask for deferred procedure calls to be queued using kernel or I/O timers. The kernel API supports timer resolutions in 0.10 microsecond increments, however, this only guarantees that the DPC gets queued to execute, and actual kernel timer services are not guaranteed to function at that high a resolution.

Microsoft recommends that drivers should be designed to have as short an ISR as possible, and they should be able to tolerate significant delays in the dispatch of their DPCs. Keeping ISRs as short as possible and tolerating delays before DPC processing are the two key factors in making drivers able to be used in the same machine without interfering with each other. Drivers that mask interrupt processing for prolonged periods of time can interfere with the operation of other drivers.

The DPC is normally added to a first-in, first-out (FIFO) queue of pending DPCs. If DPCs are present in the queue, they are dispatched when the ISR activity has ceased. The DPCs are not prioritized and execute to completion in their queued order. In multiprocessor systems, multiple DPCs can be active on different processors at the same time. Also, in multiprocessor systems, the DPC can become immediately active upon queuing if the other processor is not at an elevated IRQL level.

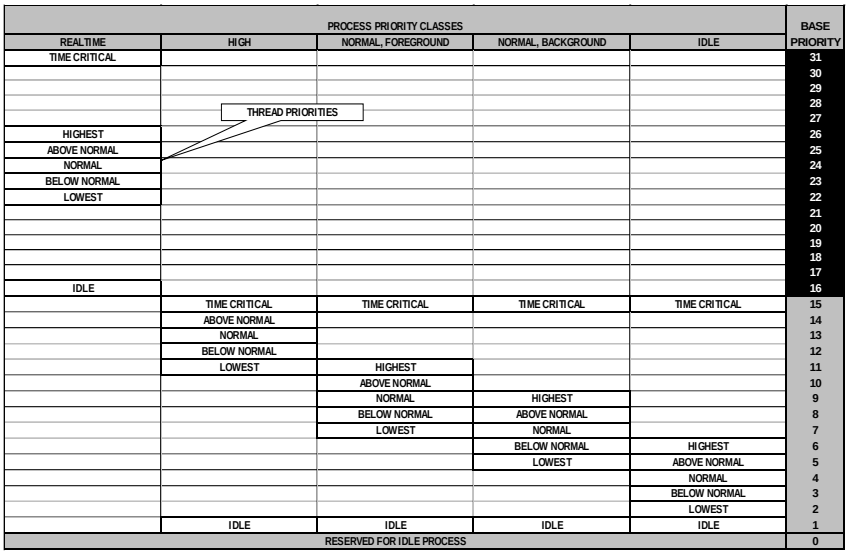


**Figure 2: Process Classes and Thread Priorities of the Windows NT Operating System**

Threads are dispatched on a processor when there are no ISRs dispatched and no DPCs pending. Internally, the Windows NT operating system maintains 32 thread priority levels, numbered 0-31, and dispatches the highest priority thread. The Windows NT operating system has different policies for treating threads with priorities between 0 and 15 than it has for threads with priorities between 16 and 31. The higher priority threads have static priorities; they run until they block, yield, or complete. If multiple threads are of the same priority, they run in a round-robin fashion. In Win32 APIs, these priorities are exposed as the Real-time Priority Class.

The policy for threads with priorities between 0 and 15 is to dispatch the highest- priority thread, but rather than simply letting the thread run until it blocks, yields, or completes, it also will round-robin time-slice multitask with other threads at its priority using a 10 millisecond quantum. In addition, the Windows NT operating system applies several unique heuristic rules that automatically boost and decay thread priorities in this priority range. This range is exposed in Win32 application programming interfaces (APIs) as normal priority class, high priority class and idle priority class (see Figure 2). Desktop applications normally operate in normal priority class.

### *Privilege Levels*

ISRs and DPCs always execute at a privileged level (ring 0). Threads may run at a privileged level or at the application level (ring 3). Any code executing at ring 0 can execute privileged instructions, which on the x86 include CLI, STI, and HLT. Code running at ring 3 can try to run privileged instructions, but it will generate a privileged instruction violation exception. When this happens, the exception handler of the Windows NT operating system either stops the thread or performs the instruction on its behalf.

### *Symmetrical Multiprocessing*

When the Windows NT operating system is running with multiple processors, it has a symmetrical dispatch policy. The Windows NT operating system verifies that the *N* highest-priority objects are running at all times (where *N* is the number of processors). If a thread can be run and multiple processors are equally available, the Windows NT operating system attempts to schedule the thread on the same processor that it ran on last, to improve processor cache performance. Some early SMP HALs only dispatched ISRs on one processor, to simplify bus access synchronization. Most current system HALs support dual ISR dispatch. Dual-processor systems have become fairly common, and quad- processor systems are now available. Further increases in SMP processing power are expected in 1998.

### *Summary*

To summarize, the Windows NT operating system dispatches three types of objects: ISRs, DPCs and threads. All ISRs are higher priority than all DPCs. DPCs execute in first-in-first-out (FIFO) order. Threads are present in four classes and are scheduled by priority. The highest priority class does not have dynamic priority boosting and threads run until they block, yield or complete. The three lower priority classes use a dynamic priority-boosting algorithm developed to enhance desktop operations, and also have a 10-millisecond quantum that is used for round-robin, time-slice multitasking.

### Performance of the Windows NT Operating System

For the purposes of soft real-time, the performance of ISRs and the performance of real-time priority class threads running at the highest available level, priority 31, are of the most interest to us. Interrupt service routines offer the best soft real-time performance achievable with the standard Windows NT operating system, but at the same time they restrict the maximum code path length of the control algorithms so other device drivers' ISRs operate correctly. In addition, I/O scanner adapters cannot be accessed at this level using a standard driver of the Windows NT operating system, so special I/O access must be provided when performing control in an ISR.

Real-time priority class threads running at priority 31 provide the highest level of determinism achievable with the Windows NT operating system, without interfering with hardware operations. In addition, these threads can access device drivers using standard mechanisms. I/O scanners can be written using standard device driver models of the Windows NT operating system. This increases the level of standardization of the software and greatly reduces the risk of introducing hardware dependent problems into the system that may invalidate Microsoft and OEM (Original Equipment Manufacturer) platform testing.

In addition to setting the thread's priority to the highest level, we must also force all of the thread's code and data to load into physical memory, and then we must lock it in place, to guarantee that the thread is not delayed by page faulting. The Windows NT operating system provides APIs to help accomplish this. Another advantage of being a standard process running threads at the highest priority level is that only the code and data associated with the control thread needs to be page- locked into memory. Less critical routines that are used to access the controller for programming and supervisory access can be allowed to page fault as usual without interfering with the operation of the control threads.

### *Interrupt Service Routine Performance*

Minimum execution times for interrupt service routines on a 200MHz Pentium® Pro system were found to be on the order of 5 microseconds. Assuming that the routine averaged 6 clocks per instruction, this translates to roughly 165 instructions, and is consistent with code path-length information provided by Microsoft. Since this execution time depends on the code path length, we would expect it to scale inversely with clock speed for a given chip architecture. However, this represents only the minimum interrupt latency.

For low-priority hardware interrupts, maximum interrupt latencies on a 200MHz Pentium Pro system were observed to be on the order of 75-80 microseconds, which translates to 450-480 microseconds on a 33Mhz 486. At 6 clocks per instruction, these timings would represent maximum path lengths of ISRs on the order of 2,500-2,700 instructions. The occurrence of these longer path lengths may have been due to preemption by multiple higher priority interrupts.

If interrupt service routines were to be used for soft control, restricting the control code path length to 2,500 instructions would likely ensure trouble-free operation, however this would represent a fairly limited application capacity. Most likely, the fast Pentium Pro peripherals could tolerate interrupt-dispatch latencies commonly seen on slower 486 equipment, so path lengths as long as 15,000 instructions would probably be acceptable. However, in general, interrupt service routines should be reserved for only the most time-critical portion of a control application.

The deterministic control performance of interrupt service routines, in terms of repeatability, is on the order of 75-80 microseconds on 200Mhz Pentium Pro systems. In order to perform predictably, a soft control application running in the context of an ISR must restrict its path length (conservatively) to 2,500 instructions, or (optimistically) to 15,000 instructions to prevent it from interfering with the normal operation of other devices. On symmetrical multiprocessing systems, ISRs can be active for longer periods of time since the other processors can service the rest of the system while the ISR is dispatched.

### *Real-time Thread Performance*

While interrupt service routines only need to worry about the performance of other interrupt service routines, real-time priority class threads have to worry about pre-emption from ISRs and their DPCs. The performance of the DPCs is the major cause of variation for real-time threads. Our testing suggests that the disk driver, the network driver and, to a lesser extent, the video driver, are the major sources of long DPCs. These drivers may create significantly sized buffer copies and perform other activities that require substantial code path length.

We tested the performance of real-time threads on a variety of platforms. The tests indicated that the repeatability of periodic scan cycles was affected by the presence of different networking and disk driver devices. With a set of well-performing peripherals, performance scaled inversely with processor speed, and was dramatically improved in symmetrical multiprocessing (SMP) systems.

During disk and network stress tests on systems with well-performing peripherals, we observed roughly 16-millisecond variability on 100Mhz single-processor Pentium systems, dropping to roughly 8-milliseconds on 200Mhz single- processor Pentium Pro systems. These same test results improved dramatically for dual-processor systems. Even 133Mhz dual-processor systems displayed variabilities as low as 3-4 milliseconds with 200Mhz dual processor systems displayed variability as low 2-3 milliseconds. The availability of a second processor seemed to improve performance dramatically. We would expect this result to be even more dramatic on quad-processor systems even though ISR behavior may be slightly worse due to inter-processor synchronization issues.

### *Need for Additional Hardware Performance Certification*

During our testing, we did identify some network cards that behaved poorly, with observable delays of 30-40 milliseconds during network stress testing. Video drivers displayed poor behavior on single processor systems when configured for very high screen and color resolutions. These observed variations were likely to be device-driver dependent, driver-configuration dependent, as well as device dependent. This indicates that a more stringent hardware and driver certification program is needed, beyond Microsoft and its OEMs' basic testing, if the standard Windows NT operating system is to be used for soft control. As a result, Rockwell Automation is developing standard tests to further certify hardware performance when using the Windows NT operating system for soft control applications.

### *System Performance in Relation to Control Requirements*

The timing of the interrupt service routines was more than adequate for very high-speed control applications. However, there are limits to how large a control application can be run in that context without becoming a badly-behaved device driver. Any ISR-based control code requires special interfaces to the I/O scanner cards, since it can't use the standard device driver access model of the Windows NT operating system.

Real-time control threads on well-performing systems demonstrated variations of up to 8 milliseconds on 200Mhz single-processor Pentium Pro systems, and 16 milliseconds on 100Mhz single-processor Pentium systems. These variabilities are slightly below par relative to modern PLC performance but should be acceptable for a wide variety of basic control applications. Regulatory control with process time constants of greater than 250 milliseconds and interlock logic without sub-second timing criticalities should not be a problem for these systems. Real-time control threads performed dramatically better on symmetrical multiprocessing (SMP) systems. These systems displayed scan variability that was fairly good, even by PLC standards.

### *Summary*

The deterministic control performance of the Windows NT operating system was found to be more than adequate for many control applications. The interrupt service routines on fast single processor systems and real-time threads on most dual-processor systems performed as well as most modern PLC processors. Real-time threads on single-processor systems performed just beyond the lower ranges of modern PLC performance, but should be adequate for less time-critical process and discrete control applications.

Some video displays, peripherals, and device drivers behaved poorly relative to other systems. The application of these poorly behaved systems can lead to unacceptable control performance. Despite extensive platform certification testing performed by Microsoft and its OEMs, Rockwell Automation felt that a higher certification standard needed to be set to verify that equipment would perform adequately for soft control, regardless of operating system selection. As a result, we are further developing our system stress tests to help evaluate equipment for control applicability. At this time we intend to provide a limited list of highly tested hardware.

## Design of the Windows NT Operating System

When we first started to evaluate the Windows NT operating system for soft control applications, we were concerned about several key design areas in addition to the determinism of the operating system. Foremost in our minds was safety, so system integrity features were key including rapid failure detection and proper kernel exception handling. In addition, we wanted to make sure standard PC peripherals used in conjunction with I/O scanner and industrial network adapters could be mixed and matched with minimal risk of interoperability problems. Other areas of concern included the efficiency of thread synchronization primitives, and the ease in which we could avoid potential priority inversion problems associated with synchronization objects.

In this section, we will discuss our investigations into these design areas, and we will also discuss some other key strengths of the design of the Windows NT operating system that have not been previously available in other control systems but are very desirable from a soft control perspective. We start with a discussion of unexpected kernel exception handling and the blue screen.

### Understanding the Blue Screen of the Windows NT Operating System

There seems to be a great deal of misunderstanding regarding the stability of the Windows NT operating system and the meaning of the "blue screen." Before we discuss the blue screen in detail, we need to review the topic of unexpected kernel exceptions and their causes.

When executing, the Pentium processor generates several self-diagnostic fault and abort exceptions. When an exception is thrown, execution can be restarted if an exception handler exists that considers it acceptable to restart the fault. However, execution stops on all abort exceptions. The most relevant fault and abort exceptions that are likely to be encountered in a control system are due to either a hardware failure or a software defect. The most common exceptions are summarized in the following table:

| | | |
|---|---|---|
| 0 | Divide Error. DIV of IDIV divide by zero | fault |
| 2 | NMI exception. Hardware error | abort |
| 5 | Bounds Check Array access. Bounds check failed | fault |
| 6 | Invalid Opcode. Misaligned or corrupt code | fault |
| 8 | Double Fault Exception occurred while dispatching an exception | fault |
| 10 | Invalid TSS. An external interrupt attempted a task switch via an invalid TSS | fault |
| 12 | Stack Exception. Stack segment overflow or underflow | fault |
| 13 | General Protection. Attempt to access memory that is protected or invalid | fault |
| 14 | Page Fault. Attempt to access a virtual page that is not loaded | fault |
| 16 | Floating Point Error. Floating point calculation exception | fault |

In response to any fault exceptions at ring 0, the exception dispatcher of the Windows NT operating system makes special checks to determine whether the processor was operating on the interrupt stack and whether the IRQL level was at or above the level for DPC processing. If these two conditions are true, either system code or a device driver was executing, so the exception is considered fatal, and the system halts all threads, bug checks, and then halts or restarts. The bug check can be set up to:

- do nothing
- generate a blue screen syndrome dump
- create a memory dump file
- enter the kernel debugger

Reviewing the above list, you can see that all these exceptions were caused by either a hardware failure (memory parity error for example), or some privileged code corrupting kernel memory or executing a critical bug with unknown consequences. The errant code may not have excepted until after it corrupted writable parts of ring 0 memory (including the control and status registers of I/O and control devices). What is known for certain is that code has failed. As a result, the only safe thing to do is fault the processor and put the I/O in a fail-safe condition as soon as possible.

In both the control and the data processing world, continuing after an unexpected kernel exception, though typically uneventful, is potentially unsafe. This is true for not only Windows NT operating systems, but for other operating systems, real-time operating systems, and controller executives. By halting the processor immediately, and letting the I/O scanner watchdog time out, the Windows NT operating system responds appropriately to the kernel exception. The blue screen of the Windows NT operating system is simply the equivalent of the PLC-5 red-light fault.

Some extension technology vendors of the Windows NT operating system claim that they can partition memory in such a way as to guarantee that their real-time executive safely continues to run after an unexpected kernel exception occurs. We feel that this is a disadvantageous design and a potentially unsafe thing to do in response to a exception, for the following reasons:

- The system is just as likely (if not much more likely, see next item) to generate an unexpected kernel exception in the context of the real-time executive.
- To properly assure fail-safe operation, the real-time executive should halt immediately, just as the Windows NT operating system should halt.
- No shutdown handler can be safely called from this state, and even if one was, it could not be relied on to recover from corruption, interrupt stack overflow or from a NMI. Consequently, the I/O scanner adapter must be relied on to guarantee a watchdog time out and set the I/O to its fail-safe state.
- Having multiple system failure modes simply increases system complexity and makes system integrity testing much more complex.

The likelihood of encountering operating system and device driver bugs is higher in systems that are either more complex or have undergone less testing. Though the Windows NT operating system is more complex than the proposed real-time executives it has undergone extensive testing. The lowest-volume, least-tested components of a soft control system, which are the most likely components to be the cause of kernel exceptions, are actually the I/O adapter scanner drivers. In these partitioned systems, these drivers are part of the real-time executive and it is likely that the real-time executive will encounter unexpected kernel exceptions than the Windows NT operating system.

When using premium systems from the highest quality hardware OEMs (Compaq®, DEC®, Dell®, Hewlitt-Packard®, IBM®, Rockwell Automation, etc.), Windows NT operating systems have proven to be extremely reliable. The kernels and drivers endorsed by these key OEMs are virtually bug free. Actual case studies show that reliability problems correspond to its use with immature components and device drivers. For example, having to modify your system's BIOS in order to run the Windows NT operating system could be an indication that the manufacturer has not performed adequate testing.

It is our opinion that a conservative upgrader of Windows NT operating systems, using a premium OEM platform, will find a Windows NT operating system more reliable than a real-time executive running in conjunction with a Windows NT operating system on the same box. The use of rigorously tested, high-quality hardware is an obvious requirement for any PC-based control system.

Regardless of memory partitioning, kernel objects within the Windows NT operating system always have access to most of the physical memory used to memory map dual-ported RAM into the system, and they also have access to the hardware I/O ports. The presence of a kernel exception indicates that code has executed that either contained bugs or encountered an unanticipated state due to a hardware failure. The code that contained bugs did not necessarily except immediately, and it may have corrupted the states of devices or system control structures in ways. This would make it unsafe to continue even for the partitioned real-time executive.

### *Interoperability/Ease of Tuning vs. Real-time Performance Features*

The need for easy and trouble-free integration of third-party hardware is another key design issue that is critical to the successful implementation of open control. At the same time, it would be nice if the platform could support some more exotic means of achieving better real-time performance. As we have already discussed, the Windows NT operating system defines a set of standard driver rules that were selected to verify interoperability and performance. Drivers are encouraged to have as short an interrupt service routine (ISR) as possible, and to be able to tolerate fairly long delays in Deferred Procedure Call (DPC) processing.

If a driver obeys these rules, and the many other rules associated with plug and play (PnP) configuration, a driver most likely coexist well with other drivers conforming to these rules. This is a primary feature of "open" control: trouble- free mixing and matching of hardware options. Unfortunately, this trouble-free operation comes at a price when pushing real-time performance to its limits.

Critics of the real-time performance of the Windows NT operating system have repeatedly raised several issues in which they have asked for many real-time design extensions. Most of these extensions would cause severe tradeoffs with other goals of the Windows NT operating system including ease of tuning, and trouble-free driver operation.

The following list summarizes some of the criticisms that have been voiced and the changes that have been recommended:

***Threads are always lower priority than ISRs and DPCs*** Many of the APIs of the Windows NT operating system are simply not allowed to be called at elevated IRQLs since they cannot use kernel objects for synchronization. If application threads were allowed to operate at an elevated IRQL, they would have to follow the same rules as ISRs and DPCs. Since the only source of time-critical events is system interrupts and timers, the same functionality could be accomplished in an ISR or DPC. Though this might be a more convenient programming model than ISRs and timer DPCs, it really wouldn't improve performance significantly.

***DPCs are processed in a FIFO queue without prioritization*** The problem here is who would assign the priorities, and the second problem is that high- priority, recurring DPCs could badly delay low-priority DPCs without even knowing what their purpose was. To make such a capability useful, end-users would have to be able to adjust these priorities and ease of tuning would be lost. How is a driver to know its priority relative to a set of unknown peer drivers? The Windows NT operating system chose a FIFO to make sure each driver has access to the queue.

***DPCs are always lower priority than interrupts*** DPCs were created to complete the processing of an interrupt or to be timer routines. If an ISR needs to perform its DPC action at an elevated IRQL, it should have done it in the ISR. Placing it in a DPC that blocks interrupts is no different than extending the ISR.

The Windows NT operating system assumes DPC timer routines are less time-critical than interrupts. It might be useful to have a timer routine that had an associated IRQL level other than that of DPCs to allow extremely deterministic polling drivers. However, from the perspective of the Windows NT operating system, timer routines are usually used to timeout I/O requests and to poll devices to see if operations are complete. It might be desirable to be able to specify a timer interrupt routine and associate an IRQL with it. However, not having this feature is certainly not a showstopper.

***WaitForObject is FIFO rather than prioritized-FIFO, and synchronization objects do not support automatic priority boosting to prevent priority inversion*** These features would make sure that high-priority threads are not impeded by lower-priority ones when using simple synchronization objects, such as mutual exclusion semaphores (Mutex Objects), and critical sections.

If WaitForObject was prioritized, high-priority threads could skip forward in line and minimize the wait times for accessing synchronization objects. In addition, if the threads, which owned synchronization objects, were automatically boosted to the priority of their highest waiting thread, the priority inversion for simple synchronization objects would not be a problem.

However, these concepts do not make sense in the context of the dynamic priority boosting heuristics of the Windows NT operating system. Since synchronization objects are used independently of certain threads' priority classes, this would introduce confusion and complexity into the thread dispatcher. Furthermore, these features would introduce a significant amount of overhead into the operations of the dispatcher. Since even real-time thread priorities are modifiable at run-time, supporting these concepts would introduce a great deal of complexity into the process services APIs.

Alternatively, applications can easily work around these problems by creating slightly more complex synchronization objects, built off of the primitives, at the cost of a moderate hit in lock performance. Encapsulation classes can be (and have been) created that provide priority inversion protection and prioritized waiting. Still, developers have to be careful of hidden locks, which might be used to provide synchronization from within runtime libraries.

### *System Scalability and Portability*

One area where design of the Windows NT operating system is a perfect fit for soft control is the area of system scalability and portability. A Windows NT operating system supports increases in processor power from low-end Pentium systems up to the Pentium Pro and to dual- and quad-multiprocessor systems. Pentium II processors, with clock speeds ranging from 333MHz to 450MHz are on the near-term horizon. CPU performance can already be scaled up by a factor of roughly 8, without any changes in software design. The Pentium II should double that high-end over the next year or two. The symmetrical multiprocessing (SMP) design takes advantage of application threads and provides excellent performance improvements. In addition, Windows NT operating systems support multiple CPU architectures. Ring 3 application code is highly portable between x86 and Alpha. Drivers require slightly more attention, but even drivers in the Windows NT operating system can be written in C and have a high degree of CPU portability.

### *Standard Drivers and Ease of Administration*

Another key design advantage of using a Windows NT operating system is the ability to directly leverage standard drivers and new plug and play standards. In addition to I/O scanners and industrial networks, soft controllers need to interact with TCP/IP networks, serial ports, SCSI ports, and possibly in the near future, USB and FireWire devices. Threads have full access to all the standard Microsoft drivers. Inter-driver access, class driver and mini-port architectures of the Windows NT operating system simplify the support of many device types, but only if you are using a standard Windows NT operating system.

In addition, Microsoft is making device management even simpler with the new extensions being added to the Windows NT 5.0 operating system as it leverages Microsoft's second generation of plug and play. Microsoft's automatic device driver configuration system and the rules for well-behaved drivers provide an excellent framework for maintainable and interoperable open control.

### *Summary*

Handling of unexpected exceptions by the Windows NT operating system, typified by the blue screen display, was found to be acceptable for soft control applications. The handler unified many system failure modes and immediately stopped control thread processing, so that the I/O scanner watchdogs could quickly detect system failure and place the I/O in its fail-safe state.

Though the Windows NT operating system precluded some advanced techniques for improving real-time performance in its device drivers, most of these techniques would have introduced the need for critical user tuning and administration, vastly reducing the ease of integrating hardware and software from different vendors. The design of the Windows NT operating system provided high levels of performance, while maintaining strong rules for device interoperability.

The advanced features of the Windows NT operating system provide outstanding scalability, portability, and ease of administration. The Windows NT operating system also provides key drivers for networking and communications and makes them available to applications and drivers. Planned advances in the plug and play architecture will make setup and hardware configuration simpler than ever.

## Windows NT Reliability and Fault Tolerance

### Target Markets

Though Windows NT operating systems are not directly targeted for soft control applications, it is targeted for mission-critical server roles, and high performance workstation roles. Consequently, Windows NT operating systems have been designed for robustness, reliability and performance. The operating system is designed to resist attacks of naive or malicious users and to provide the highest degree of isolation between individual applications. In addition, Microsoft has developed unprecedented test programs to help assure the quality of hardware components and systems.

### Microsoft's Testing and Certification Processes

During its development, the Windows NT operating system undergoes regression and stress tests every night on over 200 different common platforms. The product has extensive internal alpha and beta-test programs, and has developed an outstanding reputation for quality among PC operating systems. The Windows NT operating system has an installed base of roughly 5,000,000 systems through 1997, with roughly 800,000 running Windows NT Advanced Server (NTAS). NTAS alone outpaces the combined sales of all flavors of UNIX systems.

In addition to internal alpha and beta testing, Microsoft manages its platform certification testing with its Windows Hardware Quality Labs (WHQL). The WHQL provides hardware vendors with published specifications for the different categories of system hardware. WHQL also provides the hardware OEMs with standardized automated test suites for testing and validating their hardware for entry into Windows hardware logo programs. These tests serve as the basis for a state-of-the-art, self-certification program.

Self-testing OEMs like Rockwell Automation complete automated tests and submit the logs back to WHQL, where the logs are reviewed, a test report is generated, and upon successful validation, the product is added to the appropriate Hardware Compatibility List (HCL). The OEM must provide Microsoft a representative system, along with its test logs. Microsoft reserves the right to randomly spot test the hardware, and will also test the hardware in the event of repeated incidence reports entered by end-users in the Microsoft Knowledge Base. Windows OEMs whose products fail audits have their self-testing privilege revoked, and are moved to a probationary status.

Through this extensive program of internal and external testing, Microsoft has been able to provide one of the highest quality software platforms in the history of the industry with an wide market for compatible hardware. Because of the outstanding effort Microsoft and its OEMs put into hardware compatibility testing, we must be very careful to apply Windows NT operating system so that it does not invalidate the extensive testing that the standard systems receive.

### *Fault Tolerant Technologies: RAID and Wolfpack*

The mission-critical markets that Microsoft targets with the Windows NT operating system also demand fault tolerance in addition to extensive testing for reliability. The Windows NT operating system supports many advanced features that are applicable for soft control. The two most critical of these technologies are RAID (Redundant Array of Independent Disks) file system support, and *Wolfpack* clustering software for file and database server software.

The standard Windows NT operating system supports RAID Level 0 (Data Striping Array without Parity, DSA), Level 1 (Mirrored Disk Array, MDA) and Level 5 (Parallel Disk Array with distributed parity, PDA). With MDAs, all hard disk access is applied redundantly to a second disk drive. If one fails, the system continues without fault. Mirrored disk systems are designed to allow removal and insertion under power (RIUP) of disk systems. When failed drives are replaced, they automatically resynchronize their image with the remaining drive without interrupting service.

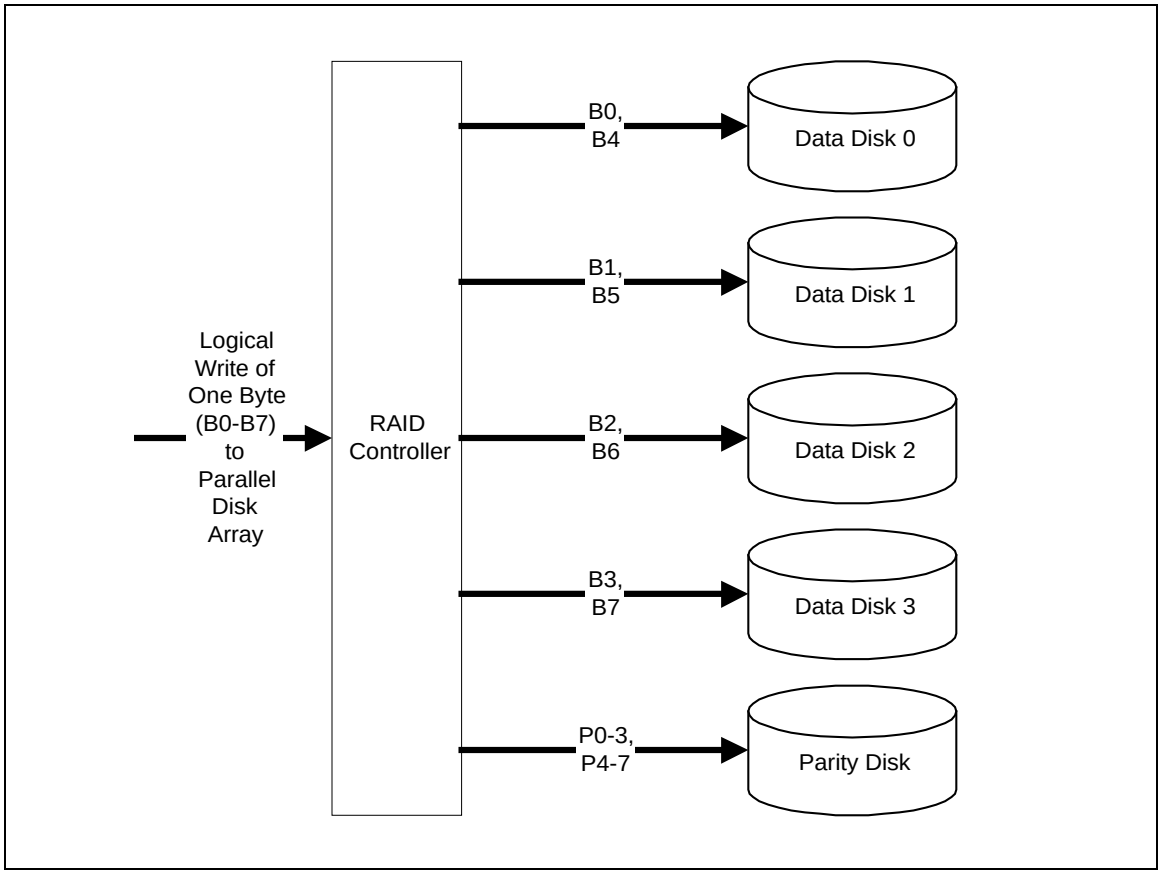


**Figure 3: RAID Level 3 Theory of Operations (similar to RAID Level 5)**

RAID Level 3 PDAs transfer data in parallel across the array's N data disks, and use a redundant check disk to store parity bits for each N bits of data stored in the array. Parity data can be combined with the data on the other drives to recover the state of any single failed disk. RAID Level 5 support of the Windows NT operating system uses a slightly more complex scheme than the one shown for RAID Level 3. PDAs provide both fault tolerance and enhanced disk burst rate performance. For example, a seven-drive PDA provides six times the data burst rate of a single drive. PDAs also support automatic resynchronization when the drives are replaced. In addition, advanced RAID systems support drive spindle synchronization, providing the best performance for large sequential transfers.

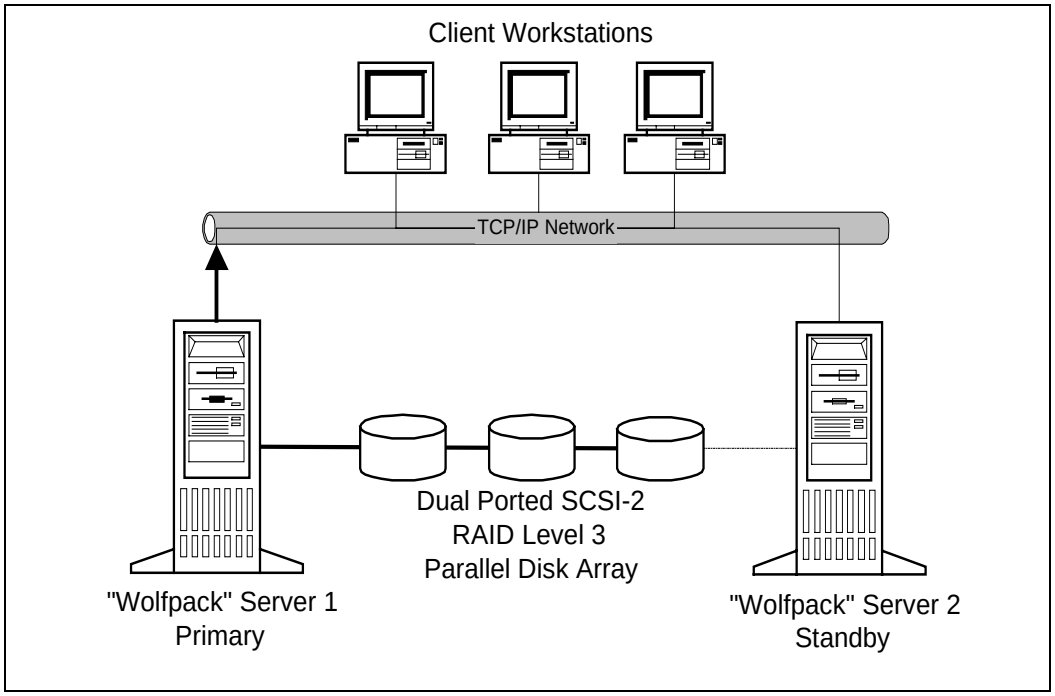


**Figure 4: Microsoft Wolfpack System Block Diagram**

In addition to RAID support, Microsoft is developing fault tolerant file- and database-server capability in its *Wolfpack* clustering project. Wolfpack phase I provides the ability for two servers, who share access to a dual-ported RAID drive system, to serve as a fault tolerant file server and/or SQL Server database server. The servers share an IP address, so they appear as one system to the network and transparently provide redundancy to client systems. Wolfpack also provides a system API for building fault tolerant applications on the platform. Future phases of clustering will support larger general purpose and fault tolerant clusters.

### *Summary*

Microsoft targets the Windows NT operating system for both mission-critical and high-performance markets. Through Microsoft's Windows Hardware Quality Laboratory (WHQL), Microsoft runs an extensive OEM hardware testing and certification program that produces outstanding reliability, and provides an unbelievably wide range of compatible hardware options. Windows NT operating systems have built-in support for high performance, fault tolerant disk arrays, supporting RAID Levels 0, 1, and 5. Wolfpack clustering software is entering beta-test, and will provide Windows NT operating systems with redundant file and database server support.

# Key Areas of Concern with Extension Approaches

The previous sections of this paper discussed general background information regarding deterministic control requirements and provided a detailed assessment of the performance, design, reliability and fault tolerance of Windows NT operating systems. This section discusses some of the key areas of concern that we had when we reviewed alternative approaches to standard Windows NT operating systems. These alternatives included providing real-time extensions to the Hardware Abstraction Layer (HAL) and subsequently combining a Windows NT operating system with a real-time kernel.

Despite some confusion in their positioning, all of the extension approaches we reviewed, including VenturCom, Imagination Systems and Radisys, were very similar. All included modifications to the HAL, and introduced a proprietary real-time kernel for scheduling real-time activities. Over time, these approaches have influenced one another and have become even more similar. Though they all claimed that their kernels were derivations of their previous products, all were, in fact, new code bases. From the perspective of real-time extensions, their approaches have converged.

### *General Schematic of Real-time Extensions for VenturCom and Imagination Systems*

The most significant difference between the approaches was that Radisys INtime provided a memory protection scheme between its real-time kernel and the Windows NT operating system kernel, using the hardware-tasking model built into the Intel processor. Unfortunately, this restricted INtime to Intel processors. In addition, INtime did not support symmetrical multiprocessing (SMP). With dual-processor SMP systems already widely available, and quad-processor systems starting to emerge, this limitation is a major drawback.

While all three approaches promised to improve the deterministic control performance of the Windows NT operating system, they also introduced key problem areas with their designs, their complexity, their supportability and their long-term viability.

### *Design Issues*

To make sure that the Windows NT operating system could not interfere with the response of the real-time systems, all the real-time kernels execute above the highest hardware IRQL. They operated as if they were the highest priority interrupt of the Windows NT operating system. From the perspective of the operating system, they can be considered as high priority interrupt service routines (ISRs). Typical drivers of the Windows NT operating system limit their ISRs to code path lengths of less than 2,500 instructions, and delays of up to 80 microseconds on a 200MHz Pentium Pro.

By executing extensive control applications at elevated IRQL, all of these approaches had the potential to violate Windows guidelines for well-designed drivers. Many devices can only tolerate interrupts being blocked for a short period of time before they lose data. For instance, the 16550 UART has a 16-character FIFO queue. At 19.2 Kbps, that queue can receive roughly 8 milliseconds of data before it has to discard characters. Similarly, network adapters typically have 8K to 64K of RAM to queue received packets. If queue is exceeded, then new frames are discarded and reacquired later by the time-out logic.

Though some of these vendors had strategies for preventing starvation of the Windows NT operating system, they typically involved time-slicing the system at a fairly high frequency (every 250-500 microseconds) or requiring SMP systems. The time-slice approach would greatly increase the likelihood of pre-empting critical sections of interrupt service routines, which was considered a design risk. Standard drivers are designed so they can prevent preemption by elevating the IRQL. With these real-time extension systems that does not work. These drivers would be operating in an untested mode in which critical sections may be pre-empted, and introduce undesirable behavior. Furthermore, if preemption problems existed, they would be extremely difficult to reproduce and isolate, and they would manifest themselves as random system crashes. As we have already mentioned, SMP systems work very well without extensions, so why go to the trouble of modifying the Windows NT operating system if you are going to always use SMP systems anyhow?

In addition, at this elevated IRQL, there is no way to synchronize with the kernel and to directly call kernel services, or for that matter, any device drivers or system libraries whatsoever. Approaches that do not partition the real-time system's memory could use kernel services to load images from disk and to load dynamic link libraries (DLLs) before they started their real-time threads; however, the partitioned memory approach precluded even this arrangement. Consequently, these systems had to provide their own process and image loaders.

The inability to leverage existing device drivers was a major disadvantage of this approach. Drivers for the hard disks, serial ports, parallel ports, and TCP/IP networking are critical to writing soft control applications. Also, these real-time executives required device-partitioning schemes that are not recognized or supported by detect feature on the Windows NT operating system, or the upcoming enhancements to plug and play system technology.

We expect the universal serial bus (USB) and FireWire (IEEE 1394) peripheral device standards to play an important role in soft control, and these schemes preclude leveraging of Microsoft's class drivers for these subsystems. USB is slated to replace both the keyboard, printer and communications ports on modern PCs. USB provides as many as 126 devices per port, with throughput rates of 4-12Mbps and built-in support for removal and insertion under power (RIUP). FireWire is expected to provide high-speed interfacing of devices (200+ Mbps), also with multi-drop and RIUP capabilities.

In all of the real-time environments, you could not link against static versions of standard Microsoft run-time libraries because they include operating system synchronization and DLL dependencies. Consequently, the extension technology vendor has to provide their own fully re-entrant, multi-threaded, run-time libraries including floating-point and transcendental function support, to allow their real-time kernel's threads and interrupt routines to use floating-point math. Reliance on the real-time system vendors' libraries made it impossible to incorporate key Microsoft technologies directly into the control products. Our product plans view integration of the control system with Microsoft technologies such as DLLs and component object model (COM) components including ActiveX controls as two key driving forces for open control. Other key driving forces include direct integration with Windows-based MMI and SQL database systems.

### *Complexity Issues*

Not only did the real-time executives introduce development constraints and the risk of unpredictable behavior, but they also increased system complexity. For high-speed, floating-point intensive applications such as advanced General Motion Control (GMC), Computer Numeric Control (CNC), and coordinated drive control these improvements in determinism may be needed and are certainly desirable. Still, we felt there were much simpler alternatives available for performing these activities without assuming the risks associated with modifying the Windows NT operating system.

PCI bus-mastering coprocessor boards can perform extremely high-speed control on the PC bus without interfering with the operation of the Windows NT operating system. This approach provides guaranteed real-time performance for advanced GMC and CNC applications without constraining the control application running on the Windows NT operating system. These coprocessor board approaches provide more firewall than any of the real-time extension approaches since they can continue to run even if the main processor executes a hard stop (CLI; HLT;) or encounters a critical NMI.

One final complexity issue involved the need to partition devices and drivers into a real-time set and a set for the Windows NT operating system. Though this may seem reasonable on the surface, it overlooks a common class of devices that need to be shared between the real-time environment and the environment of the Windows NT operating system: the disk drives, the serial ports, the parallel ports, and the network adapter. In addition, it forces industrial interface card vendors to provide two sets of device drivers: one for the Windows NT operating system, and one for the real-time executive.

Providing a Local Procedure Call (LPC) mechanism between the real-time executive and the call server of the Windows NT operating system can provide an interface between the two systems, but this also adds another layer of complexity to the puzzle. LPC proxies and stubs available to access the services of the Windows NT operating system. Some vendors are providing routines to access the registry, the disk drive, and the generic drivers of the Windows NT operating system. Of course, when the real-time thread uses services in the Windows NT operating system, it is no longer real-time, and the real-time system is no longer independent of the operating system.

### *Reliability Issues*

In terms of system reliability, we felt that, at best, an extended version of the Windows NT operating system would be slightly less reliable than standard Windows NT operating system. With over 5,000,000 users, an extensive test program for its development, and an extensive OEM program for hardware compatibility testing, we felt that the Windows NT operating system was extremely reliable. Problems are often tracked to problem hardware or device drivers that are not on the hardware compatibility list or may soon be taken off.

On the other hand, these real-time extensions are still in development or version 1.0 code. Despite liberties taken by vendors to try to tie their reputations to past projects, they all represent new designs. Since most of these systems are still at the beta-test stage, it is impossible for us to determine their reliability at maturity. Given their corporate resources and the wide variety of solution providers working with them, it is very unlikely that they will have a platform certification program even comparable to the Microsoft program.

Of the extension vendors, only Radisys INtime makes the claim that its environment can survive unexpected kernel exceptions in the kernel or its device drivers. We question whether continuing operation after this point is safe (as discussed earlier in the "Understanding the Blue Screen of the Windows NT Operating System" section). For our soft control designs, INtime does not provide a rich enough environment to host our entire control application. If we had based our system off of INtime, or any of the alternatives, we would have been dependent on the successful execution of both the Windows NT operating system and the real-time extensions.

### *Support Issues*

Perhaps the most distressing issue confronting these extension technologies was long-term support. In general, Microsoft does an outstanding job of issuing service packs and providing information regarding their purpose and function. However, reliance on a third-party vendor to test the extended environment, propagate the request through its solution providers, and then have the solution provider provide a unified service pack to the customer was considered a support nightmare. The problem only gets worse for major revisions of the Windows NT operating system. We felt the risk of poor support increased over time, as technologies diverge from those that were installed, and the vendors' interests move to newer accounts.

Since all of the real-time executive vendors provide modified HALs, they also have to deal with associated OEM HAL conflicts. Microsoft licenses the HAL code to OEMs so they can add hardware-specific support for their advanced platforms. Features such as symmetrical multiprocessing, power management, multiple busses, and the like may require HAL customization to be supported. Unfortunately, the HAL is a monolithic DLL, and customizations cannot be merged. Consequently, some OEM hardware will not be supported by these real-time extensions, with advanced server platforms being the most likely category of incompatibility.

Furthermore, we felt the presence of intrusive real-time extension systems might invalidate Microsoft's and its OEMs' platform certification testing. Since the real-time extensions introduce untested stalling and preemption activity into the standard device drivers, some of these drivers were considered likely to exhibit problems. Furthermore, problem isolation and problem accountability were likely to be unusually difficult to resolve given this environment.

There is also a major initiative underway to improve the plug and play capabilities of Windows NT operating system version 5.0. While this is an improvement long awaited by end-users, it is likely to be a major support issue for real-time extension vendors. Since the device driver model for the Windows NT 5.0 operating system is going to change, there will be strong momentum to move to the new platform. The plug and play system does not understand device partitioning and will have to be overridden or cleverly misdirected in order to configure systems with real-time extensions.

### *Long-term Issues*

Beyond the design constraints, the added complexity, the reliability issues, and the expected support issues, are the long-term issues of technology evolution. First, the expected improvements in system performance of the Windows NT operating system (as a result of Moore's law, increases in processor speed fostered by Intel and others, and the increasing number of affordable SMP designs) will naturally extend the operating system in terms of real-time performance. In addition, Microsoft's continued development of the Windows NT operating system, including support of 64-bit architectures, will provide even further performance advantages. Over time, the niche-application driving forces for modifying the Windows NT operating system will by degrees fade away.

A second evolutionary force critical to picking a good platform for soft control is the software maturation curve. Soft control is in its infancy, and introducing it into the highest-speed, most time-critical applications first, is simply not prudent. Most of the current soft control discussions center on factors that limit best-case performance. However, at the state of product maturity in the marketplace these limits won't be achieved. The products need to be field-tested and proven in a wide variety of applications before the prudent end-user will consider them for the most mission-critical applications.

Several vendors opted to collect one data point on the Windows NT operating system, present it out of context, and then claim that it is unacceptable for control. These vendors have now committed their R&D to a path of proprietary extensions that will limit their ability to adopt standard Microsoft technologies.

In the first half of 1998 Microsoft introduced Wolfpack redundancy and clustering, and OnNow technology with the Windows NT 5.0 operating system. This technology provides fast resume rebooting capability, but is unlikely to work for a system that doesn't give the Windows NT operating system access to all of its memory. In addition, the Windows NT 5.0 operating system will usher the next generation of Plug and Play (PnP), including dynamic driver loading and removal and insertion under power (RIUP) for PCMCIA, USB and FireWire devices.

It is very questionable whether a small vendor can keep pace with Microsoft in the operating system marketplace. For instance one of the largest extension vendors is a company of about 300 people however, and the majority of their revenue is derived from their hardware products. It is estimated that less than 20 people in this company are directly involved in the real-time extension project. It is key for these real-time extension vendors is to provide solutions that build on, not around, the Windows NT operating system.

Our strategy at Rockwell Automation is to track Microsoft operating system development. This provides customers with proven, well-tested technology while maintaining completely open systems.

Furthering our open systems strategy we currently have a number of Windows CE developments underway. Windows CE at version 3.0 will provide interrupt latencies of 50 uSec with the ability to theoretically prove worst-case performance ("hard real-time"), and a platform for small footprint systems.

It is our intention to offer SoftLogix controllers for both Windows NT & CE.

### *Other Areas for Improvement*

Despite the many issues involved in extending the capabilities of the Windows NT operating system, we still feel that there are many areas where real-time extension vendors can add significant value. We feel most of the vendors unfortunately have been myopically focused on deterministic control performance. Other key issues impacting the use of the Windows NT operating system for soft control include:

- simplifying software maintenance, both of the operating system and of the soft control system software
- speeding up boot times (Microsoft OnNow on the horizon)
- improving self-diagnostics
- supporting repair while running - removal and insertion under power, RIUP (Microsoft PnP for supports RIUP for USB, FireWire and PCMCIA)
- improving the mean-time-to-repair (MTTR) the system, by making the platform more of an appliance that can be swapped out (Microsoft's Zero Administration Windows [ZAW] is also trying to address this issue)

### *Summary*

Vendors of real-time extension technologies and of control applications built on these technologies have tried to represent the Windows NT operating system as not being appropriate for soft control applications. Based on our testing and experience, we feel that the standard Windows NT operating system is adequate for a wide variety of soft control applications, and we feel it is highly preferable over less compatible, lower-volume options available from the real-time extension vendors.

Though improvements beyond the current level of performance are desirable for some applications, extension technologies that push the Windows NT operating system beyond its limits tend to be:

- Immature
- Limited in their degree of integration with the Windows NT product
- Increase system complexity
- Raise longer-term concerns about support and reliability.

We believe these technologies will be of interest for niche applications requiring tighter deterministic control performance, if they demonstrate a high degree of compatibility with the Windows NT operating system, and if they can maintain that compatibility as the operating system evolves.

# Conclusions

## *Restatement of Why We Chose the Windows NT Operating System*

Rockwell Automation was faced with a critical decision regarding our operating system strategy for open PC-architecture-based soft control. Market forces made it clear that Windows compatibility was key to the success of soft control, and the Windows NT operating system was clearly preferable for mission-critical applications. Even after targeting Windows NT-compatible systems, various alternatives existed in the marketplace including the standard Windows NT operating system and several extender technologies (Radisys INtime, VenturCom RTX, Imagination Systems Hyperkernel, etc.).

This detailed research of the capabilities of the Windows NT operating system, and the capabilities and risks of the extension technologies, we chose the standard Windows NT operating system as our primary soft control platform. The performance of the Windows NT operating system on fast Pentium processors and symmetrical multiprocessing systems was found to more than adequate for a wide variety of soft control applications. The design and mission of the Windows NT operating system were also found to be highly compatible with the goals of soft control.

Support by the Windows NT operating system of RAID mirrored disk arrays and parallel disk arrays was found to be an excellent fit for control applications. In addition, key new technologies from Microsoft, including OnNow, Zero Administration Windows, Wolfpack clustering, enhanced plug and play, and support of removal and insertion under power and dynamic driver loading for Universal Serial Bus, IEEE 1394 FireWire and PCMCIA devices, will provide many exciting new capabilities in the near future.

Though real-time extension technologies offered hope of improved performance over that of the Windows NT operating system, we felt that:

- Real-time extension technologies were not required and precluded true open control.
- The technologies were immature.
- They increased system complexity.
- They raised long-term risks in terms of hardware compatibility, support, and access to future technologies.
- Where improved performance was required, the extension technologies held little benefit over PCI bus mastering control coprocessor boards or dedicated controllers, both of which introduced none of the inherent risks associated with modifying, blocking, and pre-empting the Windows NT operating system.

Rockwell Automation

Rockwell Automation helps its customers receive a superior return on their investment by bringing together leading brands in industrial automation, creating a broad spectrum of easy-to-integrate products. These are supported by local technical resources available worldwide, a global network of system solutions providers, and the advanced technology resources of Rockwell.

## Worldwide representation.

Argentina • Australia • Austria • Bahrain • Belgium • Bolivia • Brazil • Bulgaria • Canada • Chile • China, People's Republic of • Colombia • Costa Rica • Croatia • Cyprus Czech Republic • Denmark • Dominican Republic • Ecuador • Egypt • El Salvador • Finland • France • Germany • Ghana • Greece • Guatemala • Honduras • Hong Kong Hungary • Iceland • India • Indonesia • Iran • Ireland • Israel • Italy • Jamaica • Japan • Jordan • Korea • Kuwait • Lebanon • Macau • Malaysia • Malta • Mexico Morocco • The Netherlands • New Zealand • Nigeria • Norway • Oman • Pakistan • Panama • Peru • Philippines • Poland • Portugal • Puerto Rico • Qatar • Romania • Russia Saudi Arabia • Singapore • Slovakia • Slovenia • South Africa, Republic of • Spain • Sweden • Switzerland • Taiwan • Thailand • Trinidad • Tunisia • Turkey • United Arab Emirates United Kingdom • United States • Uruguay • Venezuela

Rockwell Automation Headquarters, 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
Rockwell Automation European Headquarters SA/NV, avenue Herrmann Debrouxlaan, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Rockwell Automation Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1789-1.2 – July 1998