

Distributed Real Time Computing with Windows NT

Windows NT by itself may not be the optimal solution for real-time computing, but there are various options to extend the operating system. One of these, RadiSys Corporation's INtime real-time extension for Windows NT has been enhanced to provide distributed options, resulting in a highly scalable feature set. This article discusses why the feature set has been extended and how it effects scalability.

WHY DOES WINDOWS NT NEED A REAL-TIME EXTENSION?

Many researchers have investigated the real-time capabilities of Microsoft's standard Windows NT operating system (at the time this article was written version 4.0 with SP4 was the current version). All have arrived at the same conclusion: although Windows NT may be able to provide a timely response to events, the system architecture can never guarantee fully deterministic behaviour [1].

One reason for the lack of guaranteed deterministic behaviour has to do with the way Windows NT handles interrupts. Windows NT interrupts cannot be controlled by an application; instead, the Windows NT kernel uses a FIFO mechanism to deal with interrupt requests. Each interrupt request (IRQ) results in a call to a small Interrupt Service Routine (ISR), which has all interrupts disabled; this ISR generates an NT kernel request to perform a Deferred Procedure Call (DPC), which is placed into a FIFO queue of all outstanding DPC requests. This method of deferring interrupts via a FIFO queue can be disastrous. For example, when a system is flooded with mouse interrupts (which are rarely important for stable system operation) the FIFO queuing of pending interrupts (DPCs) can cause uncontrollable delays in the handling of other more critical interrupts. Avoiding the FIFO queue by doing all processing inside the ISR is not acceptable either, because it makes the system unresponsive to all interrupts (due to the fact that all interrupts are disabled during a Windows NT ISR), resulting in a system with proper behaviour for one interrupt only.

Some system designers may claim that they have tuned the total Windows NT system, including device drivers and application programs, in order to achieve acceptable determinism. But with the current rate of Service Packs and new releases of the operating system, plus the need for system feature adjustment typically required by customers, it becomes highly improbable that such tuning can be effective in the long term, after a few cycles of updating and modifications.

There are essentially two methods for correcting the lack of determinism in Windows NT:

- Use a separate computer system running a proper real-time operating system (RTOS) and connect it by a networking link to the Windows NT system.
- Extend the Windows NT system in such a way that at least part of the system provides the desired level of determinism.

The first approach may look attractive because there are many reliable and powerful RTOSes available from various suppliers. A disadvantage is that the application developer has to work with two different operating systems and their development environments. The most significant disadvantage is cost: even the smallest application requires two complete hardware systems.

Extending Windows NT is an approach taken by only a few suppliers. Modifying the Windows NT kernel would be the best method, but is not a realistic option because the Windows NT kernel source code is not available and Microsoft has never expressed any intent to make this change to its kernel – such a kernel change could even effect current Windows NT applications in a negative sense! Thus, another approach must be used to extend the determinism of Windows NT.

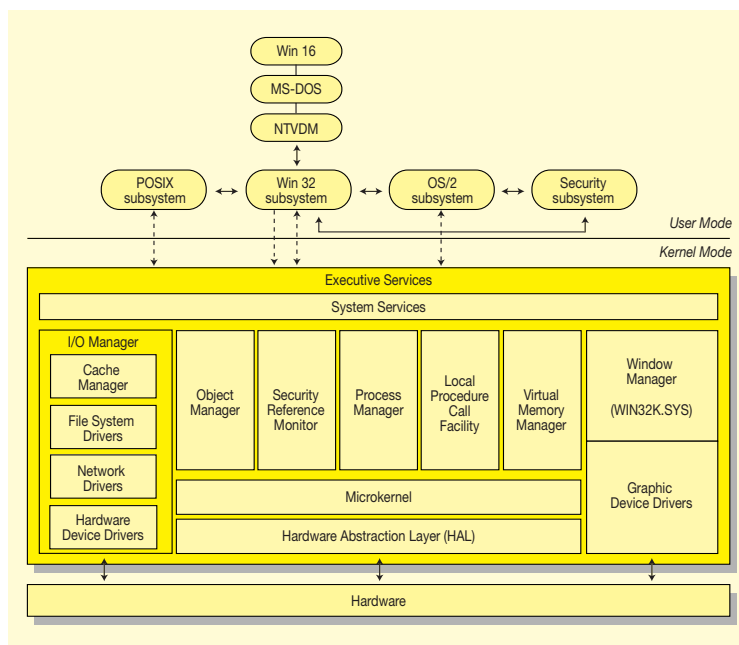


Figure 1. Windows NT architecture

HOW DOES INTIME SOFTWARE ADD REAL-TIMENESS?

Rather than comparing all extension approaches, we will discuss the INtime real-time extension to Windows NT [2].

The first version of INtime operated as a small real-time kernel, stealing memory from Windows NT to load its own code and real-time applications. Thereafter the INtime kernel makes sure that real-time threads receive processor time as required for real-time operations, and leave all remaining processor time for Windows NT and its applications. Since typically real-time work comes in short bursts, this usually leaves the user (if any) and non-real-time threads with an apparently normal Windows NT system.

Taking control of the hardware away from the Windows NT kernel is possible by sophisticated employment of mechanisms built into the 32-bit Intel processor architecture (the INtime kernel requires an Intel Pentium or later processor). One such mechanism is a hardware task, which can switch the complete processor context of a the running program (in this case the entire operating system) in a matter of microseconds. The Windows NT kernel assists in this passing of hardware control by having all its hardware access centralised in a single Hardware Abstraction Layer (HAL). It is fairly easy to modify the working of the Windows NT HAL without affecting the Windows NT kernel and its applications. An example of the kind of change applied to the INtime version of the Windows NT HAL is the 'disable all interrupts' function. In the standard Windows NT HAL this function is implemented using the CLI processor instruction, which disables interrupt recognition at the processor interrupt pin. The equivalent INtime HAL function disables all interrupts controlled by Windows NT and its drivers, but keeps real-time interrupts enabled.

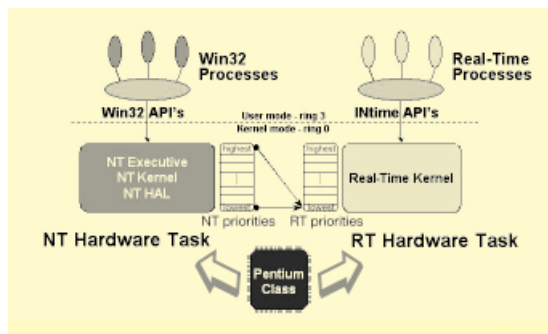


Figure 2. INtime architecture

The basis for the INtime kernel comes from the iRMX operating system, an RTOS that has been in use and is still being deployed in thousands of real-time systems since the early 1980s. Using an existing kernel rather than developing a new one avoids the risk of race conditions that are hard to detect.

The upper layers of the iRMX operating system were intended to provide full system capabilities to a stand alone real-time operating system, and as such are easily overshadowed by Windows NT's vast file access and networking capabilities. RadiSys therefore decided to abandon these layers and instead provide the

INtime application programs with easy access to all Windows NT functionality; since such functions require Windows NT to execute, they do not behave deterministically, but in practice this is already true for most file systems and networks. If the need exists, an INtime designer can install a secondary file system or network, fully controlled by INtime drivers and thus providing real-time access.

A major move from the iRMX style was the introduction of a Windows NT based development environment for INtime: real-time programs are developed using the standard Microsoft Visual C/C++ compiler, using some dedicated code wizards and a real-time debugger. Since the INtime API was designed to be close to the Win32 API, a Windows NT programmer does not experience a steep learning curve when developing INtime applications.

WHO NEEDS DISTRIBUTED SYSTEMS?

As mentioned before, the first release of the INtime real-time extension for Windows NT operated on the same processor as Windows NT, sharing its memory and peripherals. Since that time, customer requests made it necessary to go beyond this single system model.

One reason for these requests is the need for additional processing power: some Windows NT systems need a lot of number crunching or graphic processing, which means continuous access to all available CPU power. Although one could imagine that supporting symmetric multi-processor systems might alleviate this problem, RadiSys decided that such a modification of the INtime kernel would introduce undesirable reliability problems.

Overall system reliability is another reason for distributing or partitioning system responsibilities: a PC may crash its Windows NT system because of device driver issues or hardware malfunctioning, which could hamper the included real-time kernel. One solution to this is to utilise multiple hardware environments.

All in all this resulted in the need to execute the INtime kernel stand-alone, but in such a way that the Windows NT program used for visualisation and data base access would not require any change: enter scalability.

DISTRIBUTED SYSTEMS, THE INTIME STYLE

Since the INtime kernel originates from the iRMX operating system it was not difficult to have it run stand-alone on a separate piece of hardware. In the INtime nomenclature we distinguish two modes of operations:

- **local INtime node:** the INtime kernel operates on the same hardware as the Windows NT system, sharing the main memory and processor; it typically executes one or more real-time applications that control hardware interrupts.
- **remote INtime node:** a separate hardware system executes a stand-alone version of the INtime kernel. Such a system does not require the same level of "horsepower" as does Windows NT. For example, the minimal processor required for a remote INtime node is a 386 CPU; a display and keyboard are not necessary and the system may boot from flash memory, opening the door for black boxes.

Remote INtime nodes can be linked to the Windows NT PC by a dedicated serial link or via TCP/IP over Ethernet. The method of communication is generic enough so that other media types could also be supported, such as a CompactPCI backplane or a Firewire link.

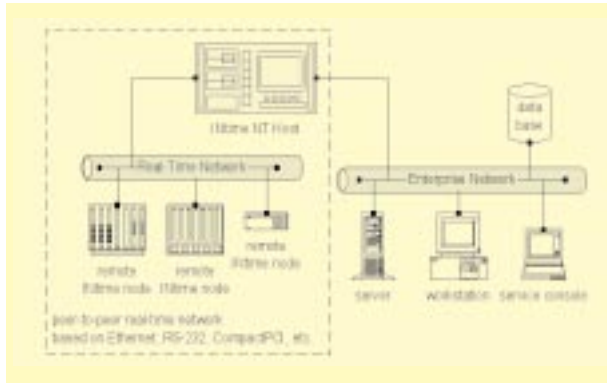


Figure 3. INtime distributed systems

For a Windows NT application there is no difference between communicating with a local or a remote INtime node application: in both cases the Windows NT application needs only a "handle" to the INtime object. It uses the handle as a parameter in the functions that form the communication API.

As an example, if a Windows NT application needs to send a command to a real-time application, it can use an INtime mailbox. During its initialisation, the Windows NT application obtains the name of the INtime system (for example from the registry or via a user dialog), then looks up the handle of the mailbox and uses the handle for data transfer. The only difference between local and remote INtime nodes is in the name of the INtime node used to look up the handle.



Figure 4. The INtime node browser control

Since the actual Windows NT application is the same for both a system with only a local INtime node and a system with a Windows NT PC and one or more remote INtime nodes, the result is a highly scalable system model. In all cases the developer uses the well-known Windows NT visual developer tools.

DISTRIBUTED SYSTEM MANAGEMENT

When software is distributed over more than one system, the situation may arise that one partner in a conversation disappears. The event of such a disappearance could become clear using time-outs, but to make

it easier to handle such cases in a general way, the INtime kernel includes a Distributed System Manager (DSM). This DSM monitors the connection between remote nodes and the Windows NT PC and issues signals to the system when an event occurs such as:

- a remote system goes off-line (due either to a hardware failure or a software crash)
- a remote partner program has logged off
- a remote system has come back on-line
- a new remote partner program wants to communicate
- a request comes for the local program to shut down

Such events are typically handled by having a separate real-time thread waiting for them. Note that a waiting real-time thread does not use any processor time.

CONCLUSION

The first version of the INtime real-time extension for Windows NT added deterministic real-time capabilities to Windows NT, by taking over control of the hardware that Windows NT runs on, but still leaving most of the hardware available to Windows NT and its applications.

With the addition of distributed system support, it is now possible to develop a real-time application that uses Windows NT for its user interface and wide-area connectivity and that can utilise the optimum hardware for the task. Small installations can be implemented by sharing the Windows NT system hardware with the real-time applications. In larger installations the real-time work may be spread over one or more remote INtime nodes, separately from the Windows NT hardware and tasks.

The scalable software is easily controlled because neither the Windows NT side of the system nor the real-time program rely on the specifics of the hardware configuration. This provides system designers with scalable real-time software with a familiar (interface), without putting extra burden on the application developers. ■

REFERENCES

- [1] Windows NT as a Real-Time OS?; Real-Time Magazine 97-2.
- [2] Windows NT Real-Time Extensions: an overview; Real-Time Magazine 97-2 and also: INtime 1.20 Evaluation – Executive Summary; Real-Time Magazine 99-2.

Jan Baan is specializing in software products for RadiSys Corporation in Europe and as such he is the technical contact for the INtime real-time extension for Windows NT. Besides evangelizing and supporting INtime, he also takes actively part in the further development of INtime and other software products. Before joining RadiSys in 1996, he worked for many years with Intel Corporation in technical positions concentrating on operating systems such as iRMX (real-time), OS/2 and Windows NT; responsibilities were pre- and post-sales support, customer training and consulting. Earlier positions were in system software development at a Netherlands-based software house and at Philips Computer Industry.