

# To the editor of Real Time Magazine

Dear Editor,

In your magazine, issue 2 of 1999, you have published an executive summary of your INtime 1.20 Evaluation (page 9 and following). In my opinion your summary is a bit too negative at a few points, which I would like to correct here:

## ARCHITECTURE, TASKS

You mention that "processes cannot be created dynamically". The actual situation in INtime 1.20 and later is that all real-time processes are created from an EXE file (actually we call it RTA for Real Time Application) that resides on a Windows NT file system. As we anticipate that control over such files and processes belongs with the controlling (Windows NT) application, there is a Windows NT utility to load the RTA file and create a real-time process for it on an INtime node (the more recent release 2.0 includes a new API to make this even simpler).

So a more correct summary is: "all real-time processes are created from a controlling Windows NT process; such real-time process creation can take place at any time after the INtime node comes alive and tools are also provided to locate running processes and to kill such processes (which then releases all the process's resources). A tool is also provided to perform the real-time process creation as a Windows NT service at system start".

## ARCHITECTURE, INTERRUPTS

INtime 1.20 does not *only* use the thread interrupt model, but instead offers the programmer various choices, depending on the needs of the particular interrupt:

1. Every interrupt has its own Interrupt Service Routine (ISR), which is called directly by the hardware, either by an interrupt call (if a INtime thread is already running) or by a hardware task switch (if Windows NT is in control). The ISR executes in a special context with interrupts disabled and can only call upon a small subset of the INtime API.

If the ISR needs limited processing and does not rely on thread communication (for example it accesses input/output ports but does not need to synchronize with semaphores) it can do all interrupt handling and simply return (meaning an ISR call overhead of only a few microseconds).

If more time consuming or thread-involved processing is needed, the ISR signals one or more threads and after that the normal priority-based, pre-emptive scheduling takes over, probably (because of thread priorities) switching to (one of) the interrupt thread(s).

2. For an interrupt thread (IT) there are a few design choices, but basically an IT is just another thread, where for example its priority and state determine when it becomes running.

## TOOLS & DEVELOPMENT METHOD

Reading this paragraph in the executive summary may lead to the impression that the development method is severely hampered by omitting the option of running a real-time process on Windows NT itself. But in reality a real-time program is completely different from a Windows NT program:

- A Windows NT program (using the Win32 API, for example written in Visual C/C++ possibly with use of Microsoft's Foundation Classes, or written in Visual Basic) has to do with the user interface, meaning control and reaction to such elements as windows, menus, mouse events, dialog boxes and so on; it also controls storage of data either locally or on remote data base servers. This leads to fairly complex programs, where user-friendliness often is more important than raw speed.
- A real-time process takes care of setting up hardware devices such as A/D and D/A converters, handles interrupts and is responsible for correctly timed reaction to hardware and software events. Typically the result is a much smaller program where response times are critical; background processing is hardly ever present, because placing that inside Windows NT avoids interference with real-time requirements.

Hence we believe that being able to execute a real-time process inside the Windows NT environment does not help much in proving the correctness of such a process: it can only prove that in *another environment* the calculations seem to be correct, but the critical behaviour (the response times) can never be tested on Windows NT itself (why else do we need a real-time extension). So not being able to run a real-time process on Windows NT itself is not an omission, but a conscious choice of the design team to avoid confusion of the real-time programmer.

## FINAL NOTE

When the evaluation was performed by Real-Time Consult, INtime version 1.20 was the latest version. Since that time, RadiSys Corporation has proceeded its development and the newer release 2.0 has many enhancements, such as support for distributed systems and Embedded C++ classes. Information on these newer features can be found at our website ([www.radisys.com](http://www.radisys.com)) or can be requested from the author. ■