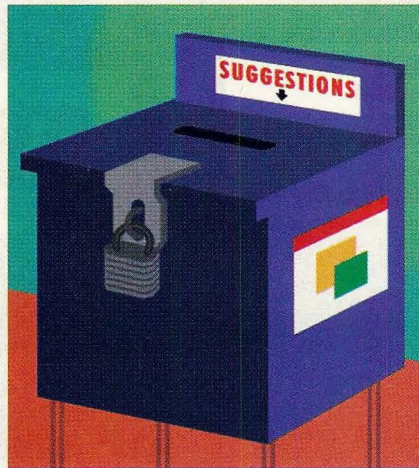


# Windows NT

By DOUGLAS A. HAMILTON

## A Few Suggestions for Microsoft

LIKE WINDOWS NT very much. Technically, there are a number of very attractive points in its favor. The base capabilities, multiple processes, pipes and other interprocess communication (IPC) facilities, virtual memory and so on make NT capable of matching any alternative tit for tat.



The microkernel approach with application-layer subsystems providing client/server support for the Win 3.x, Win32, POSIX and other interfaces goes one step further, distancing NT from almost anything else. Advantages include better protection and security, portability to other hardware architectures and a growth path to additional subsystems as they are needed.

Symmetric multiprocessor (SMP) support is shaping up to be far more important than I expect most of us might have guessed. Without NT, there really is no point in having an SMP. If you're running multiple applications in several windows, or even one application

with several active threads, NT transparently allocates the work across all the available processors. Suddenly, an SMP actually makes sense.

My guess is that a year from now we'll see a flood of \$1,000 SMP motherboards with two to four 486s, all made possible by NT. For developers faced with trying to run several big makes at once and others with heavy load requirements, an SMP is going to be a "gotta have" machine.

Having lived with NT for the last six months, I think a few things could be improved.

Hands down, the strangest decision was the way current directories are handled. The NT kernel tries to get by with a notion of only a single current directory, rather than the DOS or OS/2 notion of a current directory on each of however many logical drives you have. Doing a SetCurrentDirectory to a path on another partition also changes the current drive. Admittedly, DOS-style drive letters are pretty archaic and a poor fit for anyone with lots of drives spread out over a large network. The problem, though, is that users and applications have come to expect them. So NT has to hack to simulate them. If you list out the environment by typing set at the command prompt, you'll notice some weird entries like this:

```
=C:=C:\WINNT\SYSTEM
```

In this example, the current directory on C: is \WINNT\SYSTEM. The variable name =C: has been chosen just to differentiate from names anyone else might choose. What should be a kernel (or Win32 subsystem) responsibility to manage the current directories is thus dumped overboard onto the application.



Traditionally, environment variables have been used only to pass an environment to a child process, but under NT they even affect simple file system calls like `OpenFile`. If you try opening `c:\hello.c`, what you get depends upon the current setting of the `=C:` environment variable, violating the principle that the result of calling a function should depend only upon parameters that are explicitly passed, rather than on a lot of miscellaneous global variables.

Another problem with environmental variables is that the designers intended that the environment block should be opaque, meaning you're not supposed to know (or ask) how it's laid out. To get or set an environmental variable, an application is supposed to use only the approved `GetEnvironmentVariable` and `SetEnvironmentVariable` calls.

I'm guessing here, but I suspect this is so the actual environment can be (in some future release) kept in Unicode but translated on the fly into standard ASCII for older applications. The problem is that get and set is not enough. Suppose you'd simply like to get the complete list of all the variables. You're going to have to go look inside the environment block yourself because there's no way to ask the system simply to give you a list.

I was also struck by the decision to make environment variables case-insensitive. Under DOS and OS/2, `COMMAND.COM` and `CMD.EXE` routinely translated everything to uppercase, but that was a limitation of only those command processors, not of the underlying system; the `getenv` and `putenv` library functions certainly supported case-sensitive names. Experience has already shown that case-sensitive filenames are important to some markets. That's why NT offers options to support them. So it's not clear why NT architects chose to break with the past and make the environment case-insensitive.

Also, one particular environmental variable, the `PATH` variable, has a strange restriction: It's not possible for users to customize their accounts to put something ahead of the system defaults. I'm told this is for security purposes so the system can't be fooled about where to find some of its critical components when a user logs on. However, that

sounds like an excuse, not a reason.

Signal handling on NT is currently pretty weak. It's not anywhere near the richness found on UNIX, where one might send different kinds of signals to indicate different kinds of exceptional events. Worse, in the July build, if you hit `Ctrl+C`, an interrupt is sent to every process running in the current window. There's no way to say, when you create a process, that it should be considered a background activity and not be interrupted. Arguably, OS/2 is even uglier architecturally, but in practice is not as much of a problem.

## Interrupt Handlers

Under OS/2, interrupt handlers go onto a stack and interrupts are delivered to the last process to have pushed its handler onto the stack. It's the responsibility of the parent to take the interrupt and kill off any children. If you make the mistake of trying to run something that has an interrupt handler in the background, the result is weird, but since the kinds of simple things one might run in the background don't usually have interrupt handlers, it generally works.

Fortunately, by the time you read this, the next build of NT should be out, bringing a welcome improvement. The fix is in several parts: `CreateProcess` is being enhanced to allow children to be created that will be sheltered from `Ctrl+C`. The child can also be made the root of a new list of processes. To interrupt the child and any descendants, `GenerateConsoleCtrlEvent` is called, giving it a handle to the child.

Process creation is both better and worse than under OS/2. It's better because there's no need to know just what kind of application you're starting. Under OS/2, every different kind of application—full screen, text windowed, PM or DOS—must be started differently. Under NT, you can start anything just by calling `CreateProcess` and passing it the executable filename. But it's worse if you do care what's going to happen. There's no way to know immediately before or immediately after calling `CreateProcess` whether the child will run in the same or separate window, short of looking at the `.EXE` file yourself. Admittedly, the file formats aren't that complex to decipher, but

NT really does need something corresponding to OS/2's `DosQAppType`.

Another problem that would be simple to fix is the way abnormal terminations are handled. If a process crashes, it just goes away. I suppose there may be vendors who prefer that—this way, if their apps crash their customers are left in the dark without any clearly incriminating evidence.

I think most vendors want to know if their products fail, hoping you'll report it along with enough data to fix it. NT should put up a pop-up with a register dump and the exception code when an application fails. Even better would be an option for snapshotting the process state to disk so the vendor could look at it with a post-mortem debugger.

Of course, first we need a decent debugger. NTSD, the command-line oriented debugger, is a little like a time warp back to an era of front panel lights and switches. It doesn't understand local variables at all, and on a MIPS machine doesn't even understand source code. Another problem is that if you're debugging a console app, it really should create a new console to run it. Instead, it runs in the same window.

Windbg, the Quick C derivative, is certainly prettier, but not a lot better. Maybe it's just me, but I have trouble doing even simple things like examining the call stack with Windbg. As bad as it is, I use NTSD. NT really needs `CodeView`.

Finally, the documented Win32 API is not actually the native API of the system. Win32 is actually built on top of the underlying NT API. Although I've already identified this as a strength, it's not free. There are a number of straightforward functions missing, even though they must be possible. For example, there's no way to timestamp a directory or format a diskette using only the Win32 API.

All the NT problems I've listed here are minor and fixable. I still believe developers who port their product to NT are making the right choice. ■

*Douglas Hamilton is president of Hamilton Laboratories (Wayland, Mass.) and author of the Hamilton C Shell, an advanced interactive command processor and tools package for OS/2 and Windows NT. Reach Douglas on WIX as hamilton or care of Editor at the address on page 10.*