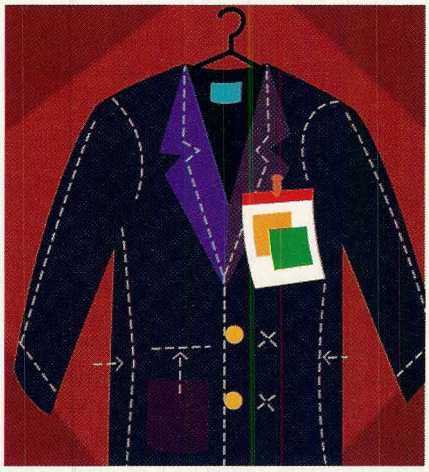
By Douglas Hamilton

Tailoring OS/2 Apps for NT

IN MY LAST COLUMN, I outlined my reasons for concluding that NT is a lot more real than many of us had suspected and mentioned that anyone who's been working with OS/2 is going to feel right at home with the NT API (application programming interface). This time we'll take a little closer look at that API to see what's involved in porting an OS/2 application to NT.



Scott Baldwin

At the same time, keep in mind that the NT API is Win32 (32-bit Windows). NT is just the first implementation of Win32. So, when we talk of porting to NT, we're talking about producing an application that would run not just on NT but also on any future Win32 systems, including rumored future versions of DOS.

Jumping right in, a large number of NT system calls map one-for-one with those in OS/2. Reading and writing from open file handles, moving a file pointer (seeking), sleeping for a period, creating threads, etc., all map fairly directly.

All the names have changed, however. OS/2 calls all have three-character prefixes identifying the functional groups to which they belong (DOS for kernel functions, VIO for text window display functions, GPI for the PM graphics calls, Win for the windowing functions, etc.)

A lot of folks thought that made their code somewhat verbose, but it *did* mean one could tell something of what a function did just from looking at that prefix. It also meant that if one avoided creating symbols beginning with those specific prefixes, there was very little risk of collision with names that were already used.

On NT, that sort of prefixed naming convention has been dropped. Names of the functions are intended to be descriptive, but that's as far as it goes. So, while OS/2 had DosRead, and DosWrite calls for reading and writing a file, the corresponding NT calls are ReadFile and WriteFile. As in all systems, however, sometimes the naming in NT is just a little odd; e.g., to open a file—even one that exists—the right call is CreateFile.

An interesting philosophical difference between the systems shows up in error reporting. Under OS/2, all system calls return a value that indicates whether the call succeeded and, if not, what went wrong.

Under NT, the return value is simply a Boolean indicating success or failure; if it failed and, it's important to know why, the application must call GetLastError, which returns the actual error code set by the last system call made by that thread. So, even though

there's some genuine one-to-one mapping in functionality, there is a structural difference, albeit one that can sometimes be masked using the C preprocessor. For example, to painlessly convert all OS/2s DosWrite calls to NT's WriteFile, a simple #define can be used:

```
#define DosWrite(file, buffer, length, written)\
( WriteFile(file, buffer, length, written, Null) ?\
0 : GetLastError())
```

Some things are quite legitimately different. On NT, the kernel thinks of all the file system partitions as mounted into one directory tree. There's really only one current directory. Recreating the "fiction" of individual current directories on each logical drive is actually done by way of environmental variables inherited from parent to child that specify the current directories. For example, dumping the environmental variables, one might find:

```
=C:=\nt
=D:=\doug\util
=E:=\tmp
```

In some minor cases, functionality OS/2 provided is missing. Since logical drives are only an application-level view, there's no kernel call for setting or getting the current drive; that has to be done by setting or parsing the current directory. Also, it's possible to get the timestamp on a directory but not to set it.

Another example is the whole Extended Attribute (EA) mechanism introduced with HPFS in OS/2 1.2. The idea behind EAs was that one could associate an arbitrary set of attribute names and values with any file or directory. Unfortunately, without much of an architecture in place to tell how to use them, EAs turned out to be mostly just a solution in search of a problem. Having concluded that EAs were a mistake, the NT kernel preserves any EAs it finds if a file is moved or copied, but deliberately hides them from the application layer.

Anyone faced with porting a text application will be encouraged to know that NT does offer text windows. In contrast to IBM, which has been doing everything possible to discourage text applications, refusing to convert the Vio and Kbd functions to 32-bit and offering spurious arguments that text windows aren't portable, Microsoft has taken a characteristically more pragmatic approach: They've recognized that text applications aren't going away any time soon and that having applications available for their system—even if they're not all done the Microsoft way—is better than not having them at all. So if you like text windows, enjoy! NT has them.

BUT NT'S TEXT WINDOWS do differ rather substantially from the ones on OS/2. NT's text windows are simply rectangular arrays of character cells with color attributes. For example there isn't any support for decoding American National Standards Institute escape sequences (for setting colors, moving the cursor, etc.). Programmers needing that functionality must build it themselves.

Reading the keyboard is also different. On OS/2, the keyboard always returns actual characters (including those representing the outboard keys). By contrast, NT's keyboard looks more like a message queue: Applications see all the individual key down and key up events (including those events for the individual shift keys). Also, if a key is held down, the event record may specify a repeat count.

That repeat count is actually a bit troublesome, since there's no way to take just one of the repeated series of keystrokes or to read the record, decrement the count and push the record back onto the head of the input queue. Consider a scenario where, based on the first keystroke, a process should spawn a child to read the rest: I'm not sure there's any good way to make that work.

Most OS/2 developers will appreciate that the symbolic names (and even the numeric values) for the various error codes are pretty much the same for NT. An ERROR_PATH_NOT_ FOUND, for example, is the same thing on both systems. That, at least, eliminates most of the effort that might otherwise have gone into rethinking how to respond to the various errors that may arise when your program is run.

IN OTHER AREAS, THE amount of work going to NT depends on whether one's coming from OS/2 1.x or 2.0. For example, under OS/2 1.x, most multithreaded applications are likely to be chock-full of RAM semaphores. They have the advantage of being extremely fast, not requiring a system call to initialize and flexibility; they could be used for either mutual exclusion (mutex: which ensures that if two threads wanted the same resource, only one got it at a time) or for event signaling.

All those RAM semaphores have to be redone under either 2.0 or NT, both of which distinguish between mutex and event semaphoring and require system calls for initialization. Here's a situation where NT is actually more like OS/2 2.0 than OS/2 1.x is. It's also a case where if one is porting from 1.x and not sure whether to do the 2.0 or the NT port first, we might listen to the Cheshire cat in "Alice in Wonderland:" "If you don't know where you're going, any path will get you there." Doing either the NT or 2.0 port will help you in porting to the other system, should you need to do so later.

That same rule turns up in another way. Under OS/2 2.0, we coded our applications with copious references to 16-bit USHORTs for everything from buffer lengths to return codes to temporary variables because that's what the 1.x API used for everything.

Under 2.0, all those variables have to be changed to 32-bit ULONGs and, under NT, they become 32-bit DWORDs. Can there really be anyone left who misses the point that hard coding how data is represented into the names of the datatypes is a really terrible idea?

Obviously, for anyone porting from OS/2 1.x and faced with wholesale editing to change all those USHORTs, this might be a good time to create a new abstract datatype, call it a UAPI, perhaps, typed as appropriate for each target system–and be done with it once and for all.

Finally, there are some very pleasant differences between OS/2 and NT. One I especially like is the support for symmetric multiprocessors (SMPs). An SMP is a system with several processors, all of which have equal access to memory and I/O. Once the domain of supercomputering, multiprocessing is now edging down to high-end desktops.

Under NT, any multithreaded application is, by definition, able to take advantage of an SMP. NT transparently schedules the various threads across all the available processors without the application designer having to do anything special.

Understandably, anyone faced with porting a major OS/2 application package to NT is going to have to do some work. Some things are genuinely different and even things that are mostly the same will require tweaking and tuning to use the new environment to its best advantage. But there's certainly enough similarity to make doing the job largely mechanical.

Considering the payoff of reaching a completely new market—and potentially the entire DOS market when DOS gets a Win32 front-end—my guess is that most OS/2 vendors will find a closer look at NT well worth the effort.

Douglas A. Hamilton is president of Hamilton Laboratories in Wayland, Mass., and author of the Hamilton C Shell, an advanced interactive command processor and tools package for OS/2. He can be reached on BIX as hamilton, on MCI Mail at 389-0321, or reach him care of Editor at the address on page 8.