TITLE: Writing Device Drivers - Where to Start?
IBM Developer Connection News Volume 2
by Steve Mastrianni

I get a lot of questions from developers just starting to write device drivers. One of the most common questions is "How do I get started writing OS/2 device drivers?" Well, that depends on your background.

Because device drivers interact with the OS/2 kernel, you should have a good understanding of the basic functions provided by OS/2, such as multithreading, priorities, memory allocation, and addressing.

A majority of the questions I get involve a misunderstanding of how various types of addresses work. Driver writers must be able to work with virtual, linear, physical, and real addresses. Since the device driver interacts with the processor at the machine level, a good understanding of the processor architecture is also invaluable. Failures in a device driver usually hang the system, and tracking them down can be tedious without knowing where to look.

If you're writing your device drivers in C, you'll need a 16-bit C Compiler such as Microsoft C Version 5.1 or Version 6.0. You'll also need an assembler, such as the Microsoft Version 6.0 Macro Assembler. Previous versions (such as Version 5.1) will also work.

If you're writing Virtual Device Drivers (VDDs), you'll need a 32-bit C compiler, such as IBM C Set/2 or C Set++ (recommended) or the special 32-bit compiler, CL386, included in the Device Driver Source Kit (DDK). The DDK also includes the kernel debugger and ASDT32, which you will need to debug your device drivers. The Periscope Debugger is available commercially.

You also should get the OS/2 Technical Library, a 50-pound collection of developer reference books, which includes the OS/2 Physical Device Driver Reference, Virtual Device Driver Reference, and Presentation Driver Reference. The library is also available as part of the OS/2 Online Book Collection CD-ROM.

You can get the book, Writing OS/2 2.1 Device Drivers in C, 2nd edition. It's the only tutorial on writing OS/2 2.x device drivers. Call 1-800-842-3636 to order.

Support for device driver writers is free via IBM's DUDE (Dynamic Upload and Download Environment). Periodically, the DUDE team archives the questions into a file (removing names), which you can download. See the Directory of this Newsletter on how to access the DUDE.

You can download the file CDISK.ZIP from the Libraries section, Device Drivers, of OS2DF1 on CompuServe. CDISK.ZIP includes several sample device drivers, including a VDD sample, all written in C.

I've always written my drivers in C; IBM has historically written them in assembler. This is evidenced by the code in the PDD reference, which contains MASM examples of DevHlp calls. With the advent of Workplace OS, IBM has begun to document C-language interfaces to the DevHlps. Future releases of the DDK and driver references will include these.

Another question I get asked frequently is "How can I initialize a memory mapped adapter during initialization (Init time)?" Very often, adapters must have their memory loaded with a program or initialized. Many device driver writers experience their first Trap 13 (General Protection Fault) when attempting to perform this operation during Init.

The most common cause is that driver writers sometimes forget that Init is run as a ring 3 thread of the system. Mapping a physical address to a virtual address with PhysToVirt yields a GDT-based pointer, which is not usable from a ring 3 thread. Ring 3 threads do not have GDT access; only LDT access.

The solution to this problem is to map the physical address of the adapter to a virtual address that's mapped into the application's LDT. This is done with the PhysToUVirt DevHlp call, rather than the PhysToVirt call.

If you have a lot of data to download to the adapter, keep it in a disk file, and use the standard DosOpen, DosRead, and DosClose APIs. This is possible because Init is running as a ring 3 thread, which is the same ring level at which most applications run.

"How can device drivers can transfer data at interrupt time?"

This is a little tricky; but easy once it's been explained. Remember that Init runs as a ring 3 thread with access to the application program's LDT. The rest of the time, your device driver operates in the lowest ring, ring 0. While at ring 0, the device driver has full access to the GDT and for the most part, the entire system.

The problem is that when an interrupt occurs, your program might not be the program that is currently running. For example, your program might be blocked waiting for I/O or waiting on a semaphore. Because of this, the context, or current environment at that instant, might not be known.

Trying to map an application's buffer address at interrupt time will not work. To maintain addressability in any context, the application's buffer address must be mapped to a GDT selector. The selector, however, must be allocated during Init, and then mapped to the GDT selector for later use during interrupt time. Remember that even though you map the selector during Init, you can't use it during Init.

Steve Mastrianni is an industry consultant specializing in device drivers and real-time applications for OS/2. The author of "Writing OS/2 Device Drivers in C", Steve is regarded as one of the industry's leading experts in OS/2 and OS/2 device drivers.

TIPS

o To access a GDT selector during Init, start a timer handler that will be called within 32ms at ring 0; then perform the access.

```
o If you need to post a 32-bit shared even semaphore at interrupt
  time, which normally can't be done, allocate and arm a context
  hook.  This hook will be called at task time and therefore will
  be able to post the semaphore.
```