TITLE: Writing Device Drivers - A Brief Look at OS/2 SMP
IBM Developer Connection News Volume 5
by Steve Mastrianni

The OS/2 SMP architecture is actually quite simple. Only one copy of OS/2 is ever running at one time no matter how many processors are present, so there's no need to synchronize multiple copies of the operating system. Access to the operating system is sychronized and serialized using processor spinlocks.

A spinlock is nothing more than a small section of code that executes in a tight loop until a variable is cleared. If you've ever had a bug in your OS/2 device driver where your code executed in a loop at ring 0, you know exactly what a spinlock is. You couldn't interrupt that loop with the debug kernel, and you usually had to power off and power on to reboot. OS/2 SMP spinlocks work the same way.

OS/2 SMP provides a level of hardware abstraction using the Platform Specific Driver, or PSD. Like a device driver that shields an application from the speicifics of a particular device, the PSD isolates the OS/2 kernel from the specific processor hardware. To provide this layer of abstraction, the PSD exports generic functions that the kernel can call. These functions are translated by the PSD into operations that are specific to the hardware platform.

PSDs are special flat-model device drivers, and are actually 32-bit DLLs loaded with the DEVICE= statement in the CONFIG.SYS file. Like OS/2 ADDs, they must conform to the 8.3 naming convention, and the name must not contain any drive or path information.

OS/2 SMP requires a PSD for system initialization. The system will display an error message if a valid PSD for the current platform cannot be installed. If any step does not complete successfully, the system initialization process will stop, and an error message will be displayed.

The following new physical DevHlps were introduced with OS/2 SMP.

DevHlp Function	Code	Description
CreateSpinLock	0x6f	Create a subsystem spinlock
FreeSpinLock	0x70	Free a subsystem spinlock
AcquireSpinLock	0x71	Acquire a spinlock
ReleaseSpinLock	0x72	Release a spin lock
PortIO	0x76	Processor-independent port I/O
SetIRQMask	0x77	Set/Unset an IRQ mask
GetIRQMask	0x78	Get state of current IRQ mask

An additonal virtual DevHlp, VDHPortIO, was also added.

OS/2 SMP provides a new set of APIs to enable applications to be optimized for the SMP environment. The following is a list of the APIs and their corresponding function. This information is subject to change.

API	Function
DosCreatSpinLock	Create a subsystem spinlock
DosFreeSpinLock	Free a subsystem spinlock
DosAcquireSpinLock	Acquire a subsystem spinlock
DosReleaseSpinLock	Release a subsystem spinlock
DosAllocThreadLocalMemory	Allocate memory for a thread
DosFreeThreadLocalMemory	Free memory allocated for a thread
DosQuerySysInfo (changed)	Return system information

More information on writing device drivers for OS/2 SMP can be found on the OS/2 for SMP CD-ROM and in the third edition of "Writing OS/2 2.x Device Drivers in C", scheduled for release later this year.

Steve Mastrianni is an industry consultant specializing in device drivers and real-time applications for OS/2. The author of "Writing OS/2 2.1 Device Drivers in C," Steve is regarded as one of the industry's leading experts in OS/2 and OS/2 device drivers. Steve can be reached on CompuServe @ 73354,746 or Internet @ stevemas@vnet.ibm.com.

TIP: Writing an OS/2 device driver? Get ready for OS/2 for PowerPC by layering your device driver similar to the ADD model, separating the hardware specific layer form the software layer. This will provide you with the least work when moving your drivers to OS/2 for PowerPC.

TIP: Still writing your driver in assembly language? Try to write them in C from now on. The current direction in operating systems is to make most device drivers hardware-independent and to have them run as an applications rather than kernel tasks. Writing your driver in a high-level language will make the move to systems such as OS/2 for PowerPC much easier.

TIP: Display a device driver header from the kernel debugger. The device driver header is always at the start of the device driver's first data segment. The device driver's header contains the device name, the code and data segment selectors, the IDC entry point address and a pointer to the next device driver header.

TECHNIQUE: To display the device driver header, enter .d dev ds:0 at the debugger command prompt. The kernel debugger displays the device driver's header with text strings defining each field. An example:

```
##.d dev ds:0
```

```
DevNext: 06f0:ffff
DevAttr: 8940
DevStrat:0000
DevInt: 0000
DevName: TESTCFG$
DevProtCS: 06f8
DevProtDS: 06f0
DevRealCS: 0000
DevRealDS: 0000
```