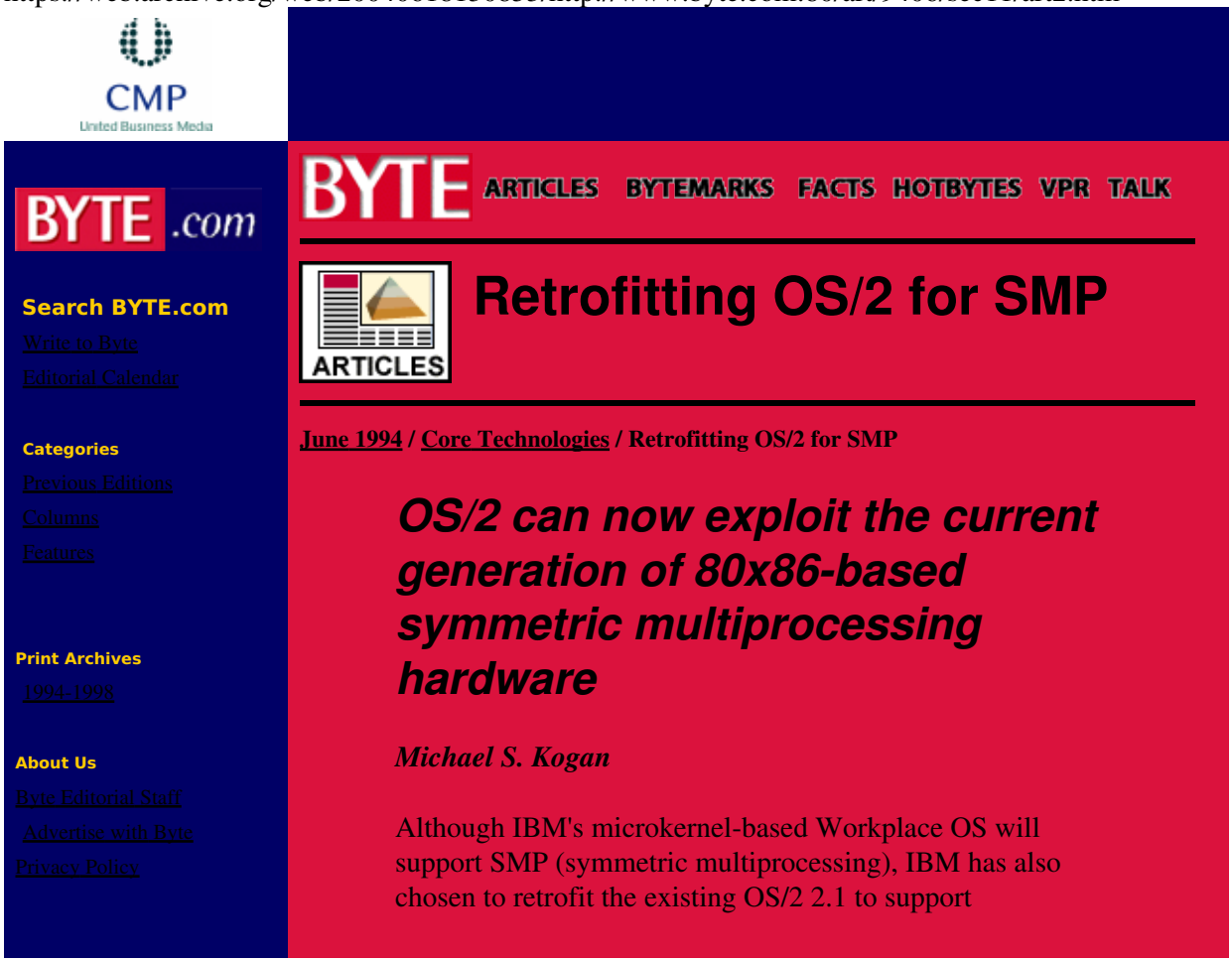

80x86-based SMP platforms. The two systems are very different beasts. In Workplace OS, SMP arises naturally from the Mach foundation. In OS/2, SMP is a retrofit. Fortunately, the necessary techniques are well understood--many Unix kernels have undergone the same transformation. To enable OS/2 2.x for SMP, global system data must be partitioned on a per-processor basis and be restructured to support true concurrent execution of processes and threads. Algorithmic enhanceme nts are needed in the areas of multitasking, memory management, synchronization, and interrupt management.

OS/2 on a Single 80x86 Processor

UP (uniprocessor) OS/2 relies heavily on two assumptions that SMP threatens: that the kernel can't be preempted and that interrupts can be disabled. When a thread runs in user mode (at ring 3, the least-trusted 80x86 privilege level), it can be preempted; when it runs in kernel mode (at the most-trusted level, ring 0), it cannot. A thread requesting system service calls EnterKMode to save its context, switches to a kernel (ring 0) stack, and then enjoys full access to the resources of the system. Once in kernel mode, the thread must finish its job quickly and return to user mode, or block awaiting some event.

The first assumption is that threads are never preempted in kernel mode--if interrupted, execution resumes after interrupt service, even if a higher-priority thread becomes ready to run. So while a thread in kernel mode is interruptible, it is not pr eemptible.

Hardware interrupts are processed in interrupt mode. When an interrupt occurs, the processor switches to ring 0, OS/2 switches to an interrupt stack, and the interrupt handler runs. If the interrupt occurred in user mode and a thread moves to the ready state, a reschedule will be forced by consecutive calls to EnterKMode and ExitKMode. If the interrupt occurred in kernel mode, no dispatch cycle occurs, and control is immediately returned to the interrupted thread.

The other assumption is that interrupts can be disabled using the CLI/STI (clear/set interrupt enable flag) idiom, to synchronize kernel-mode, user-mode, and interrupt-mode access to shared data structures.

OS/2 SMP Architecture

The problem with the first assumption--that the kernel is nonpreemptible--is that a kernel thread running on one processor can stomp on a data structure shared with a kernel thread running on another processor. The initial OS/2 SMP release solves this problem by avoiding it. Much of the ke rnel is treated as a large critical section that can be executed by just one thread--not one per processor, but one per system. Entry points into the OS/2 kernel must acquire a semaphore called a spinlock, the primary synchronization construct used in any MP (multiprocessor) system to guarantee mutually exclusive access to MP-critical sections of code. It's called a spinlock because a processor executes a tight spin loop waiting for the lock to be released. On 80x86 processors, threads acquire spinlocks using the XCHG instruction.

Will the kernel spinlock compromise the scalability of OS/2 SMP? Yes, but probably not as severely as you might think. Applications spend most of their time in user mode. Threads that do enter the kernel don't spend much time there. Some kernel components, including the scheduler and semaphores, are multithreaded and can run on multiple processors. Finally, many DLL-based system services--including all of Presentation Manager--run in user mode, where they can exploit SMP.

The problem with the second assumption--that interrupts can be disabled to synchronize kernel-mode, user-mode, and interrupt-mode access to shared data structures--is that disabling interrupts on one processor does not lock out interrupt service code running on another processor. So spinlocks are used throughout the kernel, instead of CLI/STI and simpler RAM-based kernel semaphores, and are also used by device drivers and DLLs to guard access to data structures. New APIs enable the kernel, device drivers, and DLLs to manage spinlocks. Vendors of third-party drivers and DLLs will have to use these APIs to exploit SMP.

Because SMP systems handle interrupt management and interprocessor connection and communications in different ways, OS/2 SMP defines a set of interfaces for PSDs (platform-specific drivers) that isolate the kernel and device drivers from the specifics of the SMP hardware platform. A PSD, similar in concept to (but much simpler than) the Windows NT HAL (hardware abstraction layer), co ntains code to support the low-level functions of initialization, processor management, hardware interrupt management, and interprocessor communications.

Interprocessor communication enables a thread running in kernel mode on one processor to force another processor to perform an event, such as flushing the TLB (translation lookaside buffer), rescheduling, or suspending operation. Most SMP hardware platforms allow interprocessor software interrupts to be sent on a per-processor or broadcast basis. Messages are stored in the memory that is shared by all processors in the SMP configuration.

Device Drivers

In the first release of OS/2 SMP, all hardware interrupts vector to one processor. If they didn't, existing device drivers that use CLI/STI to synchronize the execution of their kernel-mode and interrupt-mode portions would break. By performing all interrupt processing on a single processor, OS/2 SMP provides a UP-compatible environment.

If OS/2 SMP is running on a platform that is no t 8259-compatible, such as the APIC (advanced programmable interrupt controller) found in new Pentium P54C chips, existing device drivers that directly program the 8259 instead of using DevHelps for this purpose will not work.

OS/2 SMP can support interrupts across processors in two ways. Existing device drivers that use spinlocks for synchronization, and that use the existing DevHelp interfaces for interrupt management, support MP in a driver-granular way. Multithreaded drivers modified for reentrancy can support a more fine-grained kind of MP. OS/2 SMP provides the necessary infrastructure to make these modifications easy--new DevHelps for spinlock management, and the platform-specific extension architecture.

Memory Management

In OS/2 SMP, multiple processes and threads can be active and mapped in memory. Therefore, the system must duplicate several major data structures and provide efficient access to processor-specific data. Each processor has its own page directory, GDT (global descript or table), TSS (task state segment), and PCB (processor control block).

The OS/2 SMP kernel keeps the contents of page directories consistent across processors. Whenever a shared page directory entry or page table entry is modified, all processors flush their TLBs to ensure memory coherency.

The OS/2 GDT contains special descriptors that the kernel uses to map data structures related to the current process. On a context switch, OS/2 2.1 updates the base address of these descriptors rather than using multiple descriptors, so that these data structures appear at the same virtual address regardless of which process is running. To avoid the need to rewrite code that expects these data structures to appear at a fixed virtual address, the SMP kernel allocates a GDT for each processor. The GDT page that contains the special selectors is committed on a per-process basis, and the other GDT pages are shared among all GDTs.

The TSS controls stack switching on ring transitions. One TSS is allocated per processor, to enable concurrent ring transitions by threads running on different processors. PCBs are the main data structures used to access the per-processor data structures, such as the page directory, GDT, TSS, and spinlock information.

A final new data structure, called the PSA (processor save area), contains global kernel variables and per-processor data. The PSA contents are unique on a per-processor basis but appear at the same virtual address across processors. The PSA contains pointers to the processor's PCB and other processor-specific data, including links to the current process, the thread for the processor, and scheduling information.

The Uniprocessor Compatibility Challenge

OS/2 1.x made ring 2 accessible for IOPL (I/O privilege) segments, enabling 16-bit OS/2 programs to use I/O-sensitive instructions such as IN, OUT, CLI, and STI. IN and OUT access I/O ports; CLI and STI disable/enable interrupts. In the UP environment, interrupt disabling synchronizes user-mode threads accessing shared resources, manages critical sections between user-mode threads and device-driver interrupt handlers, and ensures that I/O device commands are delivered to devices without interruption.

To ensure backward compatibility and make CLI/STI work in an SMP environment, OS/2 SMP does several things. IOPL moves from ring 2 to ring 0, and the TSS IOBM (I/O bit map) is cleared. Since IOPL is ring 0, the TSS IOBM governs I/O port access. By clearing the IOBM, the process still has access to the I/O ports. Most important, though, general protection faults occur when CLI and STI instructions issue from ring 2, enabling the kernel to take control and emulate the interrupt-flag semantics.

The kernel handles interrupt-flag emulation with the special CLI/STI spinlock. When a thread executes a CLI, the OS/2 SMP kernel acquires the CLI/STI spinlock; when a thread executes an STI, the kernel releases the CLI/STI spinlock. Thus, only one user-mode thread at a time can execute with interrupts disabl ed.

Existing OS/2 programs that use the INC and CMPXCHG instructions for synchronization can potentially run into problems. These instructions don't generate a memory lock and can't ensure that only a single processor is accessing the memory. For these programs, OS/2 SMP provides a UP execution mode that prevents a process's threads from running concurrently on different processors. To make a program run in UP mode, you'll use a utility to stamp its EXE header.

Applications that have a UP priority dependency can also have problems. An application might, for example, assume that an idle class thread will not run while a regular class thread is running. Again, UP mode is the way to ensure compatibility.

OS/2 SMP should be an effective first implementation, delivering real scalability for multithreaded OS/2 programs as well as concurrent DOS/Windows sessions. An attractive solution for inexpensive, small-footprint clients and servers, it should be available by the time you read this as a p reloaded product on systems from IBM, Compaq, Wyse, and AST.

Illustration: Architectural Overview of OS/2 SMP Platform-specific drivers isolate the kernel and device drivers from the specifics of each SMP platform.

Michael S. Kogan is president of Kogan Associates, Inc. (Atlantic Beach, FL), a consulting firm specializing in personal computer software and systems. He is coauthor of The Design of OS/2 (Addison-Wesley, 1992), a freelance writer for computer publications, and a visiting professor at Nova University. He can be reached on CompuServe at 76711,212 or on BIX c/o "editors."

Up Level Next Search Comment Subscribe

BYTE TOP OF PAGE