IA32.book Page 27 Friday, June 18, 2004 3:11 PM

3 Definition of Multitasking

The Previous Chapter

As background material, this chapter provided a basic description of the single-task OS and application environment.

This Chapter

As background material, this chapter provides a very basic introduction to the multitask OS environment.

The Next Chapter

As background material, this chapter provides a very basic introduction to the problems that a multitask OS must be prepared to deal with.

Concept

It is incorrect to say that a multitasking OS runs multiple programs (i.e., tasks) simultaneously. In reality, it loads a task into memory, permits it to run for a while and then suspends it. It suspends the program by creating a snapshot, or image, of all or many of the processor's registers in memory. In the IA32 architecture, the image is stored in a special data structure in memory referred to as a Task State Segment (TSS) and is accomplished by performing an automatic series of memory write transactions. In other words, the exact state of the processor at the point of suspension is saved in memory.

Having effectively saved a snapshot that indicates the point of suspension and the processor's complete state at the time, the processor then initiates another task by loading it into memory and jumping to its entry point. Based on some OS-specific criteria, the OS at some point makes the decision to suspend this task as well. As before, the state of the processor is saved in memory (in this task's TSS) as a snapshot of the task's state at its point of suspension.

At some point, the OS makes the decision to resume a previously-suspended task. This is accomplished by reloading the processor's registers from the previously-saved register image (i.e., its TSS) by performing a series of memory read transactions. The processor then uses the address pointer stored in the CS:EIP register pair to fetch the next instruction, thereby resuming program execution at the point where it had been suspended earlier.

The criteria that an OS uses in making the decision to suspend a program is specific to that OS. It may simply use timeslicing—each program is permitted to execute for a fixed amount of time (e.g., 10ms). At the end of that period of time, the currently executing task is suspended and the next task in the queue is started or resumed. The OS may assign priority levels to programs, thereby permitting a higher priority program to "preempt" a lower priority program that may currently be running. This is referred to as *preemptive multitasking*. The OS would also choose to suspend the currently executing program if the program needs something that is not immediately available (e.g., when it attempts an access to a page of information that is currently not in memory, but resides on a mass storage device).

An Example—Timeslicing

Prior to starting or resuming execution of a task, the OS *task scheduler* would initialize a hardware timer to interrupt program execution after a defined period of time (e.g., 10ms). The scheduler then starts or resumes execution of the task. The processor proceeds to fetch and execute the instructions comprising the task for 10ms. When the hardware timer expires it generates an interrupt, causing the processor to suspend execution of the currently executing task and to switch to the OS's task scheduler. The OS determines which task to run next.

Another Example—Awaiting an Event

Task Issues Call to OS for Disk Read

The application program calls the OS requesting that a block of data be read from a disk drive into memory. Once a disk read request is forwarded to the disk interface, the disk read/write head mechanism must be positioned over the target disk cylinder. This is a lengthy mechanical process typically requiring milliseconds to complete. When the head mechanism has positioned the read/write heads over the target cylinder, the disk interface must then wait for the start sector of the requested block to be presented under the read head. The duration of this delay is defined by the rotational speed of the disk drive as well as the circumference of the cylinder. Once again, this is a lengthy delay that can be measured in milliseconds. Only then can the data transfer begin.

Rather than awaiting the completion of the disk read, the OS would better utilize the machine's resources by suspending the task that originated the request and transferring control to another program so work can be accomplished while the disk operation is in progress.

OS Suspends Task

As described earlier, the processor saves its current state (i.e., its register image) in a special area of memory set aside for this task (the task’s TSS). Once this series of memory write transactions has completed, the task has been suspended.

OS Initiates Disk Read

The OS issues a disk read command to the disk controller. The disk controller begins to seek the heads to the target cylinder.

OS Makes Entry in Event Queue

The OS makes an entry in its event queue. This entry will be used to transfer control back to the suspended task when the disk interface completes the transfer.

OS Starts or Resumes Another Task

Rather than waiting for the completion of the disk read operation, the OS will start or resume another task.

IA32.book Page 30 Friday, June 18, 2004 3:11 PM

Disk-Generated Interrupt Causes Jump to OS

When the disk controller (or, in older machines, its associated DMA channel) completes the transfer of the requested information into system memory, it generates an interrupt request. This causes the processor to jump to the disk driver's interrupt service routine which checks the completion status of the disk operation to ensure a good completion.

Task Queue Checked

The OS then scans the event queue to determine which suspended task is awaiting this completion notification.

OS Resumes Task

The OS causes the processor to reload the suspended task's stored register image (its TSS) into the processor's registers. The processor then uses CS:EIP to determine what memory address to fetch its next instruction from. The resumed task then processes the data in memory that was read from the disk.