Real-Time Kernels

A real-time kernel is software that manages the time and resources of a microprocessor, microcontroller or Digital Signal Processor (DSP).

The design process of a real-time application involves splitting the work into tasks, each responsible for a portion of the job. A task (also called a thread) is a simple program that thinks it has the Central Processing Unit (CPU) completely to itself. On a single CPU, only one task executes at any given time. A task is also typically implemented as an infinite loop.

The kernel is responsible for the management of tasks. This is called multitasking. Multitasking is the process of scheduling and switching the CPU between several tasks. The CPU switches its attention between several sequential tasks. Multitasking provides the illusion of having multiple CPUs and maximizes the use of the CPU. Multitasking also helps in the creation of modular applications. One of the most important aspects of multitasking is that it allows the application programmer to manage the complexity inherent in real-time applications. Application programs are easier to design and maintain when multitasking is used.

µC/OS-III is a preemptive kernel, which means that µC/OS-III always runs the most important task that is ready-to-run as shown in Figure 1-2.

Figure - µC/OS-III is a preemptive kernel
(1) A low-priority task is executing.

(2) An interrupt occurs, and the CPU vectors to the ISR responsible for servicing the interrupting device.

(3) The ISR services the interrupt device, but actually does very little work. The ISR will typically signal or send a message to a higher-priority task that will be responsible for most of the processing of the interrupting device. For example, if the interrupt comes from an Ethernet controller, the ISR simply signals a task, which will process the received packet.

(4) When the ISR finishes, µC/OS-III notices that a more important task has been made ready-to-run by the ISR and will not return to the interrupted task, but instead context switch to the more important task.

(5) The higher-priority task executes and performs the necessary processing in response to the interrupt device.

(6) When the higher-priority task completes its work, it loops back to the beginning of the task code and makes a µC/OS-III function call to wait for the next interrupt from the device.

(7) The low-priority task resumes exactly at the point where it was interrupted, not knowing what happened.

Kernels such as µC/OS-III are also responsible for managing communication between tasks, and managing system resources (memory and I/O devices).

A kernel adds overhead to a system because the services provided by the kernel require time to execute. The amount of overhead depends on how often these services are invoked. In a well-designed application, a kernel uses between 2% and 4% of a CPU’s time. And, since µC/OS-III is software that is added to an application, it requires extra ROM (code space) and RAM (data space).

Low-end single-chip microcontrollers are generally not able to run a real-time kernel such as µC/OS-III since they have access to very little RAM. µC/OS-III requires between 1 Kbyte and 4 Kbytes of RAM, plus each task requires its own stack space. It is possible for µC/OS-III to work on processors having as little as 4 Kbytes of RAM.
Finally, µC/OS-III allows for better use of the CPU by providing approximately 75 indispensable services. After designing a system using a real-time kernel such as µC/OS-III, you will not return to designing a foreground/background system.