

Presentation Abstract

Presentation Title	Real-Time Operating Systems 101
Author(s)	Richard Kowalski
Point of Contact (POC)	Phil Loftis
POC E-mail	Philip.D.Loftis@nasa.gov
POC Fax	304.367.2035
Presentation Abstract	<p>This presentation explains the difference between real-time operating systems (RTOSs) and general purpose operating systems. It also delves into the following components, features, and services that are a part of the RTOS so one can develop a working knowledge of its flexible and robust real-time behavioral requirements. A task (process) is an abstraction of a running program and is the logical unit of work scheduled by the RTOS. It is typically represented by a task control block (TCB) data structure that contains its state of execution. A thread is a lightweight process that shares resources with other processes or threads. Each thread must run from within some process and make use of the resources of that process. The RTOS provides thread management. RTOSs must provide the following specific functions with respect to tasks: scheduling, dispatching, intercommunication and synchronization. The kernel of the RTOS is the smallest portion that provides for these functions. A scheduler determines which task will run next in a multitasking system, while a dispatcher performs the necessary administration to start that task. Inter-task communication and synchronization assures that the tasks cooperate via mutexes, semaphores and messages. The RTOS kernel also provides the following services: security management, file management, memory management to support a dedicated hardware Memory Management Unit (MMU), time management, I/O services, resource allocation and interrupt handling. RTOS behavior should be known via determinism. Determinism is the ability of the RTOS to meet deadlines, minimize jitter and bound priority inversion. This also includes the interrupt latency (how fast an interrupt can be serviced), the maximum time it takes for every system call, and the maximum time the RTOS and device drivers mask the interrupts. An RTOS is designed for embedded safety-critical systems and has timing constraints it must meet. If it cannot respond to events (interrupts, task switches, completing a calculation, etc.) within a specific time, then the result is in error. "Soft" real-time systems have some flexibility in the timing. "Hard" real-time systems have no flexibility for critical deadlines. An RTOS has to be multi-threaded and preemptible and must also support error handling, fault protection, and multiprocessing (multiple CPU's). It must support a scheduling method that guarantees response time to critical tasks. The method of scheduling can be predetermined logical sequences verified by Rate Monotonic Analysis (RMA) or Earliest Deadline First (EDF). It can be priority-based preemptive scheduling in a multitasking environment. Another method is Round Robin time slice scheduling at the same priority for all tasks. The context switching (dispatching) time is the most important speed issue for RTOSs. Context switching is how quickly a task can be saved, and the next task made ready to run. Other RTOS speed issues are the time it takes for a system call to complete. Task threads must be able to be given a static or dynamic priority. Priority inversion is a problem where a higher priority task is blocked by a low priority task that has exclusive access to a resource. The problem occurs when a medium priority task is running, preventing the low priority one from finishing and releasing the resource. Priority inheritance has to exist and is a temporary state used to resolve priority conflicts.</p>