CENG 448 — Real Time Operating Systems

What Is an RTOS?

Dr. Larry D. Pyeatt

South Dakota School of Mines and Technology

Early Days

Applications included code to initialize and interact with the hardware directly.

Non-portable, because low-level code to operate the hardware had to be re-written.

Step one in the evolution of Operating Systems: Provide abstraction for different versions of hardware that perform the same (or similar) task.

Example: A standard function for reading/writing. A small software layer (driver) can make all I/O devices look the same. Devices become interchangeable.

This abstraction was a good idea.

General Purpose OS

- Support multitasking
- Manage resources
- Provide abstraction for applications
- Provide OS services to the applications
- Provide protection
- Provide security

All of this is to provide a good environment for the computer user to conveniently and efficiently interact with the machine.

Real Time OS

An RTOS is not so concerned with the user experience, but must provide:

- Very high reliability
- Ability to scale up or down to fit the application and hardware
- Guaranteed performance
- Faster performance
- Modest memory requirements
- Scheduling of tasks to support real-time response
- Diskless booting
- High portability

Protecting tasks from each other is not as big a concern, unless Todd is on your software team, and is writing one of the tasks.

RTOS

- A minimal kernel to provide task scheduling, message passing, and possibly other services
- A library of functions to provide
  - abstraction
  - resource management
  - communication between tasks
  - other optional modules

Scheduler

- Schedulable Entities (Software objects that can compete for execution time.)
  - Threads
  - Tasks
  - Processes
- Multitasking
  - Concurrent execution
  - Parallel execution
- Context Switching
- Dispatcher
- Scheduling Algorithms

Context Switching

Each task has a task control block TCB.

- TCB is created as part of the task
- Kernel uses the TCB to control the task
- The TCB contains everything the kernel needs to know about the task
- The TCB is updated as the task runs

When the scheduler determines that a different task should be given execution time, the kernel:

1. saves the current task's context in its TCB
2. loads the new task's context from its TCB

The first task is frozen while the other task runs.

Context switch time is important.

Dispatcher

The dispatcher handles the details of transferring the flow of control between the kernel, tasks, and interrupt service routines (ISRs).

Scheduler

Preemptive Priority-Based scheduling allows tasks with higher priority to preempt tasks with lower priority. Basically, every time the kernel code is entered (interrupt or system call) the scheduler will choose the highest priority task on the run queue.

Round-Robin scheduling gives each task an equal share of the CPU, by always taking the first task on the run queue. Every time a task gets to run, it is moved to the end of the queue. Typically done with a timer (time slicing). This can be combined with a priority scheme, where some groups of tasks have higher priority, or get a longer time slice.

Kernel Objects and Services

Common kernel objects are

- Tasks (TCB)
- Semaphores
- Message Queues

Services allow the user application to set timers, manage interrupts, allocate/deallocate memory, etc.

Characteristics

- Reliability
- Predictability
- Performance
- Compactness
- Scalability