Introduction

“A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time application process data as it comes in, typically without buffering delays.” (wikipedia.org)

Key factors in an RTOS are minimal interrupt latency and minimal thread switching latency. An RTOS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

For embedded devices, the *general* rule is that an RTOS is used when the application needs to do more than a few simple actions. An RTOS allows an application to be structured in a manner that scales as more application/system features are added (e.g. communication stacks, power management, etc.).

A RTOS has the following goals

- **Small latency**: It is real-time after all!
- **Determinism**: Again, it is real-time. You need to know how long things take to process to make sure deadlines are met.
- **Structured Software**: With an RTOS, you are able divide and conquer in a structure manner. It's straight-forward to add additional components into the application.
- **Scalability**: An RTOS must be able to scale from a simple application to a complex one with stacks, drivers, file systems, etc.
- **Offload development**: An RTOS manages many aspects of the system which allows a developer to focus on their application. For example an RTOS, along with scheduling, generally handles power management, interrupt table management, memory management, exception handling, etc.

介绍

“实时操作系统（RTOS）是一种操作系统（OS），旨在提供实时应用程序流程数据，通常没有缓冲延迟。”（wikipedia.org）

RTOS中的关键因素是最小的中断延迟和最小的线程切换延迟。RTOS的重要性在于它可以响应的速度或可预测程度，而不是它在给定时间段内可以执行的工作量。

对于嵌入式设备，*一般*规则是当应用程序需要执行多个简单操作时使用RTOS。RTOS允许应用程序以随着添加更多应用程序/系统功能（例如通信栈，电源管理等）而扩展的方式构建。

RTOS具有以下目标

- **小延迟**：毕竟是实时的！
- **决定论**：再次，它是实时的。您需要了解处理过程需要多长时间才能确保满足最后期限。
- **结构化软件**：使用RTOS，您可以以结构方式划分和征服。将其他组件添加到应用程序中是很简单的。
- **可伸缩性**：RTOS必须能够从简单的应用程序扩展到具有堆栈，驱动程序，文件系统等的复杂应用程序。
- **卸载开发**：RTOS管理系统的许多方面，允许开发人员专注于他们的应用程序。例如，RTOS以及调度通常处理电源管理，中断表管理，内存管理，异常处理等。

Standard types of threads

For this workshop, we are going to use the term **thread** as a generic term for any execution block. Here are the typical threads in every RTOS-based application.

- **Interrupt Service Routine (ISR)**: Thread initiated by a hardware interrupt. An ISR runs to completion. ISRs all share the same stack.
- **Tasks**: Thread that can block while waiting for an event to occur. Tasks are traditionally long-living threads (as opposed to ISRs which run to completion). Each task has it's own stack which allows it to be long-living.
- **Idle**: Lowest priority thread that only runs when no other thread is ready to execute. Generally Idle is just a special task with the lowest possible priority.

标准类型的线程

对于本次研讨会，我们将使用术语**thread**作为任何执行块的通用术语。以下是每个基于RTOS的应用程序中的典型线程。

- **中断服务程序（ISR）**：由硬件中断启动的线程。ISR运行完成。ISR都共享相同的堆栈。
- **任务**：在等待事件发生时可以阻塞的线程。任务传统上是长期生存的线程（与运行完成的ISR相反）。每个任务都有自己的堆栈，可以让它长寿。
- **空闲**：最低优先级线程，仅在没有其他线程准备好执行时运行。通常，空闲只是一项具有最低优先级的特殊任务。

Schedulers

Every RTOS has a scheduler at its core. The scheduler is responsible for managing the execution of threads in the system. There are two main ways a scheduler manages this:

- **Preemptive Scheduling**: This is the most common type of RTOS scheduler. With a preemptive scheduler, a running thread continues until it either
  - finishes (e.g. an ISR completes)
  - a higher priority thread becomes ready (in this case the higher priority thread preempts the lower priority thread)
  - the thread gives up the processor while waiting for a resource (e.g. a task calls `sleep()`).

  Both TI-RTOS and FreeRTOS have preemptive schedulers. This workshop will focus on preemptive schedulers.
- **Time-slice Scheduling**: This type of scheduling guarantees that each thread is given a slot to execute. This type of scheduling is generally not conducive to real-time application. The TI-RTOS kernel supports time-slicing scheduling with Tasks if desired.

调度程序

每个RTOS都有一个调度程序。调度程序负责管理系统中线程的执行。调度程序管理这有两种主要方式：

- **抢占式调度**：这是最常见的RTOS调度器类型。使用抢先式调度程序，正在运行的线程一直持续到它
  - 完成（例如ISR完成）
  - 优先级较高的线程就绪（在这种情况下，优先级较高的线程抢占优先级较低的线程）
  - 线程在等待资源（例如任务调用`sleep()`）时放弃处理器。

  TI-RTOS和FreeRTOS都具有抢占式调度程序。本次研讨会将重点关注先发制人的调度。
- **时间片调度**：这种类型的调度保证每个线程都有一个要执行的槽。这种类型的调度通常不利于实时应用。如果需要，TI-RTOS内核支持使用Tasks进行时间分片调度。

Other key terms

- **Thread-safe**: A piece of code is thread-safe if it manipulates shared data structures in a manner that guarantees correct access (reading/writing) by multiple threads at the same time. Please note, thread-safety is not just an RTOS issue (e.g. interrupts modifying the same memory must be careful).
- **Blocked**: A task is blocked if it is waiting on a resource and not consuming any of the CPU. For example, if a task calls `Task_sleep()` or `Semaphore_pend()` (with a non-zero timeout and the semaphore is not available), the task is blocked and another thread is allowed to run. Note: spinning on a register in a tight loop is not blocking...that's polling.
- **Bare-metal**: Common name for an application that does not use an RTOS.

其他关键术语

- **线程安全**：如果一段代码以保证多个线程同时正确访问（读/写）的方式操作共享数据结构，则它是线程安全的。请注意，线程安全不仅仅是RTOS问题（例如，必须小心修改相同内存的中断）。
- **已阻止**：如果任务正在等待资源而不占用任何CPU，则会阻止该任务。例如，如果任务调用`Task_sleep()`或`Semaphore_pend()`（具有非零超时且信号量不可用），则会阻止该任务并允许另一个线程运行。注意：在紧密循环中旋转寄存器不会阻塞...这是轮询。
- **裸机**：不使用RTOS的应用程序的通用名称。