基础应用篇

第 1 篇

第1章 实时操作系统的基本概念与线程基础知识

在进行嵌入式应用产品开发时，根据项目需求、主控芯片的资源状况、软件可移植性要求及软件开发工程师的技术背景等，可以选用一种实时操作系统（Real-Time Operating System，RTOS）作为嵌入式软件设计的载体。特别是随着嵌入式人工智能与物联网的发展，对嵌入式软件的可移植性要求不断增强，实时操作系统的应用也将变得更加普及。

作为本书的开始，本章从一般意义上阐述实时操作系统的基本含义；给出线程与调度的基本含义及相关术语；阐述线程的三要素、四种状态及三种基本形式。通过本章的学习，读者可以对实时操作系统的基本概念有一个初步的认识，这是应用编程及理解原理的基础。

1.1 实时操作系统的基本含义

实时操作系统是一种应用于嵌入式系统的系统软件，学习实时操作系统可以从了解其基本功能开始。本节首先简要介绍嵌入式系统的基本分类，随后阐述无操作系统（No Operating System，NOS）与实时操作系统下程序运行流程的区别，由此初步了解实时操作系统的基本功能，最后介绍实时操作系统与非实时操作系统的主要差异。

1.1.1 嵌入式系统的基本分类

嵌入式系统，即嵌入式计算机系统，它是不以计算机面目出现的计算机。这类计算机隐含在各种具体产品之中，在这些产品中，计算机程序发挥着核心作用。主要的嵌入式产品有手机、平板电脑、冰箱、工业控制系统、农业大棚控制系统、月球车等。应用于嵌入式系统的处理器，被称为嵌入式处理器。嵌入式处理器按其应用范围可以分为电子系统智能化（微控制器）和计算机应用延伸（应用处理器）两大类。

1．微控制器

一般来说，微控制器（Microcontroller Unit，MCU）与应用处理器的主要区别在于可靠性、数据处理量、工作频率等方面。相对应用处理器来说，微控制器的可靠性要求更高、数据处理量较小、工作频率较低。电子系统智能化类的嵌入式系统，主要用于嵌入式人工智能终端、物联网终端、工业控制、现代农业、家用电器、汽车电子、测控系统、数据采集等，这类应用

所使用的嵌入式处理器一般被称为微控制器。这类嵌入式系统产品从形态上看，更类似于早期的电子系统，但其内部，计算机程序起核心控制作用。本书阐述的实时操作系统主要面向微控制器。

2．应用处理器

应用处理器类的嵌入式系统，主要用于平板电脑、智能手机、电视机顶盒、企业网络设备等，这类应用所使用的嵌入式处理器一般被称为应用处理器或多媒体应用处理器（Multimedia Application Processor，MAP），多在非实时操作系统下进行编程。

1.1.2 无操作系统与实时操作系统

在嵌入式产品开发中，可以根据硬件资源、软件复杂程度、可移植性需求、研发人员的知识结构等各个方面综合考虑是否使用操作系统，若使用操作系统，应选择哪种操作系统。

1．无操作系统下程序运行流程

在无操作系统的嵌入式系统复位后，首先进行堆栈、中断向量、系统时钟、内存变量、部分硬件模块等初始化工作，然后进入无限循环。在这个无限循环中，CPU 根据一些全局变量的值决定执行各种功能程序（线程），这是第一条运行路线。若发生中断，将响应中断，执行中断服务程序（Interrupt Service Routines，ISR），这是第二条运行路线，执行完 ISR 后，返回中断处继续执行。从操作系统的调度功能的视角来理解，无操作系统中的主程序可以被简单地理解为一个实时操作系统内核，这个内核负责系统初始化和调度其他线程。

2．实时操作系统下程序运行流程及其基本功能

本书主要阐述面向嵌入式人工智能与物联网领域的实时操作系统的应用方法与原理。在基于实时操作系统的编程模式下，有两条线路：一条是线程线，编程时把一个较大工程分解成几个较小的工程（被称为线程或任务），由调度者负责这些线程的执行；另一条线路是中断线，与无操作系统情况类似，若发生中断，将响应中断，执行中断服务程序，然后返回中断处继续执行。

可以进一步理解为，实时操作系统是一个标准内核，包括芯片初始化、设备驱动及数据结构的格式化。软件开发工程师可以不直接对硬件设备和资源进行操作，而是通过标准调用方法实现对硬件的操作，所有的线程都由实时操作系统内核负责调度。也可以这样理解，实时操作系统是一段嵌入目标代码中的程序，系统复位后首先执行它，用户的其他应用程序（线程）都建立在实时操作系统之上。不仅如此，实时操作系统将 CPU 时间、中断、I/O、定时器等资源都封装起来，留给用户一个标准的应用程序编程接口（Application Programming Interface，API），并根据各个线程的优先级，合理地在不同线程之间分配 CPU 时间。实时操作系统的基本功能可以简单地概括为，实时操作系统为每个线程建立一个可执行的环境，方便线程间传递消息，在中断服务程序与线程之间传递事件，区分线程执行的优先级，管理内存，维护时钟及中断系统，并协调多个线程对同一个 I/O 设备的调用。简而言之，实时操作系统负责线程管理与调度、线程间的同步与通信、存储管理、时间管理、中断处理等工作。

3．实时操作系统的应用场合

一个具体的嵌入式系统产品是否需要使用操作系统，使用何种操作系统，必须根据系统的具体要求做出合理的决策，这就依赖于对系统的理解和所具备的操作系统知识。是否使用操作系统，可以从以下几个方面来考虑。

第一，系统是否复杂到必须需要使用操作系统的程度。

第二，硬件是否具备足够的资源来支撑操作系统的运行。

第三，是否需要并行运行多个较复杂的线程，线程间是否需要进行实时交互。

第四，应用层软件的可移植性是否能得到更好的保证。

此外，如果决定使用操作系统，那么选择哪一种操作系统呢？是否是实时操作系统？这还要从性能、熟悉程度、是否免费、是否有产品使用许可、是否会出现收费陷阱等角度考虑。

本书阐述的 Mbed OS 是由 ARM 公司于 2014 年推出的一款免费的开源嵌入式实时操作系统。

1.1.3　实时操作系统与非实时操作系统

我们知道，操作系统（Operating System，OS）是一套用于管理计算机硬件与软件资源的程序，是计算机的系统软件。通常，我们使用的个人计算机系统，在硬件上一般由主机、显示屏、鼠标、打印机等组成。操作系统提供设备驱动管理、进程管理、存储管理、文件系统、安全机制、网络通信及使用者界面等功能，这类操作系统有 Windows、Mac OS、Linux 等。

嵌入式操作系统（Embedded Operating System，EOS）是一种工作在嵌入式微型计算机上的系统软件。一般情况下，它固化到微控制器或应用处理器内的非易失存储体中，它具有一般操作系统最基本的功能，负责嵌入式系统的软、硬件资源的分配、线程调度、同步机制、中断处理等。

嵌入式操作系统有实时与非实时之分。一般情况下，应用处理器使用的嵌入式操作系统对实时性要求不高，这类操作系统主要有 Android、iOS、嵌入式 Linux 等。而以微控制器为核心的嵌入式系统，如工业控制设备、军事设备、航空航天设备、嵌入式人工智能与物联网终端等，大多对实时性的要求较高，希望能够在较短的确定时间内完成特定的系统功能或中断响应，应用于这类系统的操作系统就是实时操作系统。

相对于实时操作系统而言，适合应用处理器的嵌入式操作系统一般不再追求实时性指标，这类操作系统主要有：最初由 Andy Rubin 开发，2005 年后由 Google 持续改进的 Android，在智能手机中得到广泛应用；于 2007 年首发的、由苹果公司推出的 iOS 操作系统等。而实时操作系统中，实时性是其关注的重点，这类操作系统主要有：于 2014 年首发的、由 ARM 公司出品的 Mbed OS；于 2003 年首发的、由亚马逊公司资助的 FreeRTOS；于 1992 年首发的、由 Jean Labrosse 持续改进的 μC/OS；于 1989 年首发的、后由 NXP 公司推出的 MQX；上海睿赛德电子科技有限公司于 2006 年发布的 RT-Thread（Real Time-Thread）等。

与一般运行在个人计算机或服务器上的通用操作系统相比，实时操作系统的突出特点是实时性。一般的通用操作系统（如 Window、Linux 等）大都是从“分时操作系统”发展而来的。在单中央处理器（Central Processing Unit，CPU）的条件下，分时操作系统的主要运行方式是，对于多个线程，CPU 的运行时间被分为多个时间段，并且将这些时间段平均分配给每

个线程，让每个线程轮流运行一段时间，或者每个线程独占 CPU 一段时间，如此循环，直到完成所有线程为止。这类操作系统注重所有线程的平均响应时间，而较少关注单个线程的响应时间。对单个线程来说，其注重每次执行的平均响应时间，而不关注某次特定执行的响应时间。在实时操作系统中，要求能“立即”响应外部事件的请求，这里的“立即”含义是相对于一般操作系统而言的，即在更短的时间内响应外部事件。与通用操作系统不同，实时操作系统注重的不是系统的平均表现，而是要求每个线程在最坏情况下都要满足其实时性。也就是说，实时操作系统注重的是个体表现，更准确地讲是个体在最坏情况下的表现。

1.2 实时操作系统中的基本概念

在实时操作系统中，线程与调度是两个最重要的概念，本节首先阐述这两个概念，然后给出实时操作系统的其他相关术语的解释，理解这些基本概念是学习实时操作系统的关键一环。这里的内核是指实时操作系统的核心部分，是实时操作系统厂家提供的程序，而线程则是指应用程序设计者编制的程序，它在内核的调度下运行。

1.2.1 线程与调度的基本含义

线程与调度是实时操作系统中两个不可分割的重要概念，透彻地理解它们，对实时操作系统的学习至关重要。

1．线程的基本含义

线程是实时操作系统中十分重要的概念之一。在实时操作系统下，把一个复杂的嵌入式应用工程按一定规则分解成一个个功能清晰的小工程，然后设定各个小工程的运行规则，交给实时操作系统管理，这就是基于实时操作系统的基本编程思想。这一个个小工程被称为线程（Thread），实时操作系统管理这些线程，被称为调度（Scheduling）。

要给实时操作系统中的线程下一个准确而完整的定义并不容易，可以从不同视角理解线程。从线程调度视角来理解，可以认为实时操作系统中的线程是一个功能清晰的小程序，是实时操作系统进行调度的基本单元；从实时操作系统的软件设计视角理解，就是在软件设计时，需要根据具体应用，划分出独立的、相互作用的程序集合，这样的程序集合被称为线程，每个线程都被赋予一定的优先级；从 CPU 视角来理解，在单 CPU 下，某一时刻 CPU 只会处理（执行）一个线程，或者说只有一个线程占用 CPU。实时操作系统内核的关键功能就是以合理的方式为系统中的每个线程分配时间（即调度），使之正常运行。

实际上，根据特定的实时操作系统，线程可能被称为任务（Task），也可能使用其他名词，含义或许稍有差异，但本质不变，也不必花过多精力追究其精确含义。掌握任务设计方法，理解调度过程与底层驱动原理，提高程序的健壮性、规范性、可移植性、可复用性，提升嵌入式系统的实际开发能力等才是学习实时操作系统的重点。要真正理解与掌握利用线程进行基于实时操作系统的嵌入式软件开发，需要从线程的状态、结构、优先级、调度、同步等视角来认识实时操作系统，这将在后续章节中详细阐述。

2．调度的基本含义

在多线程系统中，实时操作系统内核（Kernel）负责管理线程，或者说为每个线程分配 CPU 时间，并且负责线程间的通信。

调度就是决定该轮到哪个线程运行了，它是内核最重要的职责。每个线程根据其重要程度的不同，被赋予一定的优先级。不同的调度算法对实时操作系统的性能有较大影响，基于优先级的调度算法是实时操作系统常用的调度算法，其核心思想是，总是让处于就绪态、优先级最高的线程先运行。然而，何时高优先级的线程可以掌控 CPU 的使用权，由实时操作系统的内核类型决定。基于优先级的内核可分为不可抢占型和可抢占型两种。

1.2.2　内核的基本概念

在实时操作系统场景下编程，芯片在启动时会先运行一段被称为实时操作系统内核的程序代码，这段代码的功能是开辟用户线程的运行环境，准备对线程进行调度。实时操作系统一般由内核与扩展部分组成，内核的最主要功能是线程调度，扩展部分的最主要功能是提供 API。内核的基本概念主要有时间嘀嗒、代码临界段、不可抢占型内核（Non-Preemptive Kernel）与可抢占型内核（Preemptive Kernel）、实时性相关概念及实时操作系统的实时性指标等。

1．时间嘀嗒

时钟节拍（Clock Tick）有时也直接译为时间嘀嗒，它是通过定时器产生的周期性中断，以便内核判断是否有更高优先级的线程进入了就绪态。

2．代码临界段

代码临界段也称为临界区，是指处理时不可分割的代码，一旦这部分代码开始执行，就不允许任何中断“打扰”。为确保临界区代码的执行，在进入临界区之前要关中断，并且临界区代码执行完后应立即开中断。

3．不可抢占型内核与可抢占型内核

不可抢占型内核要求每个线程主动放弃 CPU 的使用权。不可抢占型调度算法也称为合作型多线程，各个线程彼此合作共享一个 CPU。但异步事件还是由中断服务程序来处理的。中断服务程序可使高优先级的线程从挂起状态变为就绪态。当中断服务程序执行结束后，使用权还是回到原来被中断的那个线程，直到该线程主动放弃 CPU 的使用权后，新的高优先级的线程才能获得 CPU 的使用权。

当系统响应时间很重要时，须使用可抢占型内核。在可抢占型内核中，一个正在运行的线程可以被中断，而让另一个优先级更高且变为就绪态的线程运行。如果在中断服务程序的执行过程中有高优先级的线程进入就绪态，则当中断完成时，被中断的线程会被挂起，更高优先级的线程开始运行。

4．实时性相关概念及实时操作系统实时性指标

硬实时（Hard Real-Time）要求在规定的时间内必须完成操作，这是在设计操作系统时保证的，通常将具有优先级驱动、时间确定性、可抢占调度的实时操作系统称为硬实时系统。

软实时（Soft Real-Time）则没有那么严格，只要求按照线程的优先级，尽可能快地完成操作。

实时操作系统追求的是调度的实时性、响应时间的可确定性、系统的高度可靠性，所以评价一个实时操作系统时一般可以从线程调度、内存开销、系统响应时间、中断延迟等方面来考量。

1）线程调度的时间指标

实时操作系统的实时性和多线程能力在很大程度上取决于它的线程调度机制。在大多数的商用实时系统中，为了让操作系统能够在发生突发事件时迅速获得系统使用权，以便对事件做出反应，大都提供了抢占式线程调度功能，也就是说操作系统有权主动中止应用程序（应用线程）的执行，并且将执行权交给拥有最高优先级的线程。

调度延时（Scheduling Latency）：指当一个更高优先级的线程从就绪到开始运行的这段时间。简而言之，就是一个线程被触发后，由就绪到开始运行的时间。

线程切换时间（Context-Switching Time）：由于某种原因使一个线程退出运行时，实时操作系统保存它的运行现场信息，并插入到相应列表，依据一定的调度算法重新选择一个新的线程使之投入运行，这一过程所需时间称为线程切换时间。线程切换时间越短，实时操作系统的性能就越高。

恢复时间（Recovery Time）：指从线程执行结束后，系统恢复执行主程序所需要的时间。

2）最小内存开销

在实时操作系统的设计过程中，由于成本限制，嵌入式系统产品的内存一般都不大，而在有限的内存空间内不仅要装载实时操作系统，还要装载用户程序，因此最小内存开销是一个重要的指标，这是设计实时操作系统与设计其他操作系统的明显区别之一。

3）系统响应时间

系统响应时间（System Response Time）：指用户发出处理要求到系统给出应答信号的时间，需要满足一定的时间约束。控制要满足一定的实时性要求，就是响应时间小于临界时间。系统响应时间由反应时间和处理时间两部分组成，其中反应时间指从提交外部中断到 CPU 开始处理中断的时间，处理时间指 CPU 完成中断处理的时间。提高系统响应时间，可以从缩短反应时间和处理时间两个方面入手。反应时间是电信号的传导时间，对于不同频率的处理器，这个时间相差不大。因此，在实际的应用程序中往往通过改进算法来提高处理效率，缩短处理时间，从而缩短系统响应时间，满足系统的实时性要求。

4）中断延迟

中断是一种硬件机制，用于通知 CPU 发生了一个异步事件。CPU 一旦识别出一个中断，就会在保存线程的上下文后跳转到该中断服务程序执行，处理完这个中断后从就绪列表中选择最高优先级的线程开始执行。当实时操作系统运行在核心态或执行某些系统调用的时候，不会因为外部中断的到来而立即执行中断服务程序，只有当实时操作系统重新回到用户态时才会响应外部中断请求，这一过程所需的最大时间就是中断禁止时间。

中断延迟（Interrupt Latency）时间：指系统确认中断开始直到执行中断服务程序第一条指令为止，整个处理过程所需要的时间。中断禁止时间越短，中断延迟时间就越短，系统的实时性就越高。

1.2.3　线程的基本概念

线程的基本概念主要有线程的上下文及线程切换、线程间通信、死锁、线程优先级、优先级驱动、优先级反转、优先级继承、资源、共享资源与互斥等。

1．线程的上下文及线程切换

线程的上下文（Context）即 CPU 内寄存器。当多线程内核决定运行其他线程时，将保存正在运行线程的上下文，这些内容保存在随机存储器（Random Access Memory，RAM）中的线程当前状况保存区（Task's Context Storage Area），也就是线程自己的堆栈之中。完成入栈工作后，就把下一个将要运行线程的当前状况从其线程堆栈中重新装入 CPU 的寄存器中，开始下一个线程的运行，这一过程称为线程切换或上下文切换。

2．线程间通信

线程间通信是指线程间的信息交换，其作用是实现同步及数据传输。同步是指根据线程间的合作关系，协调不同线程间的执行顺序。线程间通信的主要方式有事件、消息队列、信号量、互斥量等。线程间通信、优先级反转、优先级继承、资源、共享资源与互斥等概念将在后续章节中详细阐述。

3．死锁

死锁是指两个或两个以上的线程无限期地互相等待对方释放其所占资源而造成的一种阻塞现象。死锁产生的必要条件有四个，即资源的互斥访问、资源的不可抢占、资源的请求保持及线程的循环等待。解决死锁问题的方法是破坏产生死锁的任一必要条件，如规定所有资源仅在线程运行时才分配，其他任意状态都不可分配，破坏其资源请求保持特性。

4．线程优先级、优先级驱动、优先级反转、优先级继承

在一个多线程系统中，每个线程都有一个优先级（Priority）。

优先级驱动（Priority Driven）：在一个多线程系统中，正在运行的线程总是优先级最高的线程。在任何给定的时间内，总是把 CPU 分配给优先级最高的线程。

优先级反转（Priority Inversion）：当一个线程等待比它优先级低的线程释放资源而被阻塞时，可能出现其他中等优先级线程先于高优先级线程被运行的现象，这种现象被称为优先级反转，这是一个需要在编程时必须注意的问题。优先级继承技术可以解决优先级反转问题，目前市场上大多数商用操作系统都使用了优先级继承技术。

优先级继承（Priority Inheritance）：它是用来解决优先级反转问题的技术。当优先级反转发生时，较低优先级线程的优先级暂时被提高，以匹配较高优先级线程的优先级。这样，就可以使较低优先级线程尽快地被执行，并释放较高优先级线程所需要的资源。

5．资源、共享资源与互斥

资源（Resources）：任何被线程占用的实体均可称为资源。资源可以是输入/输出设备，如打印机、键盘及显示器，也可以是一个变量、结构或数组等。

共享资源（Shared Resources）：可以被一个以上线程使用的资源叫作共享资源。为了防止

数据被破坏，每个线程在与共享资源打交道时，必须独占资源，即互斥。

互斥（Mutual Exclusion）：互斥是用于控制多线程对共享数据进行顺序访问的同步机制。在多线程应用中，当两个或更多的线程同时访问同一数据区时，就会造成访问冲突。互斥能使它们依次访问共享数据而不引起冲突。

1.3 线程的三要素、四种状态及三种基本形式

线程是完成一定功能的函数，但不是所有的函数都可以被称为线程。一个函数只有在给出其线程描述符及线程堆栈的情况下，才可以被称为线程，才能够被调度运行。本节先介绍线程的三要素（线程函数、线程堆栈和线程描述符），然后介绍线程的四种状态（终止态、阻塞态、就绪态和激活态），最后介绍线程的三种基本形式（单次执行、周期执行和资源驱动）。

1.3.1 线程的三要素

从线程的存储结构上看，线程由线程函数、线程堆栈和线程描述符三个部分组成，这三个组成部分也称为线程的三要素。线程函数就是线程要完成具体功能的程序；每个线程拥有自己独立的线程堆栈空间，用于保存线程在调度时的上下文信息及线程内部使用的局部变量；线程描述符是关联线程属性的程序控制块，用于记录线程的各个属性。

1．线程函数

一个线程对应一段函数代码，完成一定功能，可被称为线程函数。从代码上看，线程函数与一般函数并无区别，被编译链接生成机器码之后，一般存储在 Flash 中。但是从线程自身视角来看，线程认为 CPU 就是属于它自己的，并不知道还有其他线程存在。线程函数也不是用来被其他函数直接调用的，而是由实时操作系统内核调度运行的。要使线程函数能够被实时操作系统内核调度运行，必须先登记线程函数，设置线程优先级、堆栈大小，给线程编号等，不然当运行多个线程时，实时操作系统内核无法知道先运行哪个线程。由于任何时刻只能有一个线程在运行（处于激活态），因此当实时操作系统内核通过调度使一个线程运行时，之前运行的线程就会退出激活态。CPU 被处于激活态的线程独占，从这个角度来看，线程函数与无操作系统中的“main”函数性质相近，一般被设计为永久循环，认为线程一直在执行，永远独占处理器。但也有一些特殊性，这将在第 7 章中讨论。

2．线程堆栈

线程堆栈是独立于线程函数之外的 RAM，是按照先进后出策略组织的一段连续存储空间，是实时操作系统中线程概念的重要组成部分。在实时操作系统中，每个线程都有自己私有的堆栈空间。在线程的运行过程中，堆栈用于保存线程的上下文、线程运行过程中的局部变量。此外，当线程调用普通函数时，它还会为线程保存返回地址等参数变量。

虽然前面已经简要描述过线程的上下文的概念，但是这里还要多说几句，以便对线程堆栈用于保存线程上下文有充分的认识。在多线程系统中，每个线程都认为 CPU 寄存器是自己的。当一个线程正在运行，实时操作系统内核决定不让该线程继续运行，而转去运行别的线程时，就要把 CPU 的当前状态保存在属于该线程的线程堆栈中；当实时操作系统内核再次决

定让其运行时，就从该线程的线程堆栈中恢复原来的 CPU 状态，就像未被暂停过一样。

在系统资源充裕的情况下，可分配尽量多的堆栈空间，可以是 K 数量级的（如常用 1024 字节），但若是系统资源受限，就得精打细算了，具体的数值要根据线程的执行内容确定。线程堆栈的组织及使用由系统维护，对于用户而言，只要在创建线程时指定其大小即可。

3．线程描述符

在创建线程时，系统会为每个线程创建一个唯一的线程描述符（Task Descriptor，TD），它相当于线程在实时操作系统中的“身份证”，实时操作系统就是通过线程描述符来管理线程和查询线程信息的。虽然在不同操作系统中，线程描述符的名称不同，但含义相同，例如在 Mbed OS 中被称为线程控制块（Thread Control Block，TCB），在 μC/OS 中被称任务控制块（Task Control Block，TCB），在 Linux 中被称为进程控制块（Process Control Block，PCB）。线程函数只有配备了线程描述符才能被实时操作系统内核调度，未配备线程描述符、驻留在 Flash 中的线程函数代码只是通常意义上的函数，不会被实时操作系统内核调度。

多个线程的线程描述符被组成链表，存储在 RAM 中。每个线程描述符中都包含指向前一个节点的指针、指向后一个节点的指针、线程状态、线程优先级、线程堆栈指针、线程函数指针（指向线程函数）等字段，实时操作系统内核通过线程描述符来执行线程。

在实时操作系统中，一般情况下使用列表来维护线程描述符，使用就绪列表管理就绪的线程，使用延时列表管理延时等待的线程，使用条件阻塞列表管理因等待事件、消息等而阻塞的线程。在 Mbed OS 中，还提供了一个等待列表来管理因等待事件、消息等而阻塞的线程。当实时操作系统内核调度线程时，可以通过就绪列表的头节点查找链表，获取就绪列表上所有线程描述符的信息。

1.3.2　线程的四种状态

实时操作系统中的线程一般有四种状态，分别为终止态、阻塞态、就绪态和激活态。在任一时刻，线程被创建后所处的状态一定是以上四种状态之一。

1．线程状态的基本含义

（1）终止态（Terminated，Inactive）：线程执行已经完成或被删除，不再需要使用 CPU。

（2）阻塞态（Blocked）：又可称为挂起态。线程未准备好，不能被激活，因为该线程需要等待一段时间或某些情况发生；当等待时间到或等待的情况发生时，该线程才变为就绪态。处于阻塞态的线程描述符存放于阻塞列表或延时列表中。

（3）就绪态（Ready）：线程已经准备好可以被激活，但未进入激活态，因为其优先级等于或低于当前正在运行的线程，一旦获取 CPU 的使用权就可以进入激活态。处于就绪态的线程描述符存放于就绪列表中。

（4）激活态（Active，Running）：又称为运行态，该线程正在运行中，线程拥有 CPU 使用权。

如果一个激活态的线程变为阻塞态，那么实时操作系统将执行线程切换操作，从就绪列表中选择优先级最高的线程进入激活态，若有多个具有相同优先级的线程处于就绪态，则就绪列表中的首个线程先被激活。也就是说，每个就绪列表中相同优先级的线程是按先进先出

（First In First Out，FIFO）的策略进行调度的。

在一些操作系统中，还把线程分为中断态和休眠态。对于被中断的线程，实时操作系统把它归为中断态；休眠态是指该线程的相关资源虽然仍驻留在内存中，但不被实时操作系统内核调度的一种状态，其实它就是一种终止的状态。

2．线程状态之间的转换

实时操作系统线程的四种状态是动态转换的，有的情况是由系统调度自动完成的，有的情况是由用户调用某个系统函数完成的，还有的情况是等待某个条件满足后完成的。线程的四种状态转换关系图如图 1-1 所示。

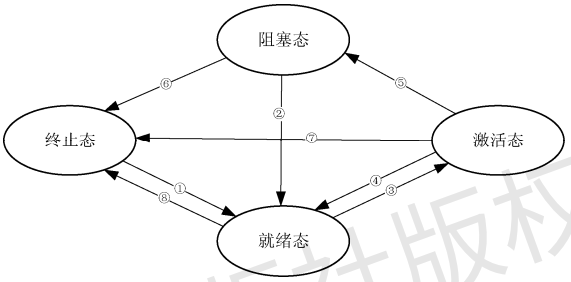

图 1-1　线程的四种状态转换关系图

1）终止态转为就绪态

终止态转为就绪态（图 1-1 中的①线）：线程准备重新运行，根据线程优先级进入就绪态。例如，在 Mbed OS 中，调用 svcRtxThreadNew()函数再次创建线程。

2）阻塞态转为就绪态、终止态

阻塞态转为就绪态（图 1-1 中的②线）：阻塞条件被解除，如中断服务程序或其他线程运行时释放了线程等待的信号量，从而使线程再次进入就绪态；延时列表中的线程延时到达唤醒的时刻。例如，在 Mbed OS 中，会自动调用 svcRtxThreadResume()函数。

阻塞态转为终止态（图 1-1 中的⑥线）：如在 Mbed OS 中，调用 svcRtxThreadTerminate()函数。

3）就绪态转为激活态、终止态

就绪态转为激活态（图 1-1 中的③线）：就绪线程被调度而获得了 CPU 资源进入运行；也可以直接调用函数进入激活态。例如，在 Mbed OS 中，调用 svcRtxThreadYield()函数。

就绪态转为终止态（图 1-1 中的⑧线）：如在 Mbed OS 中，调用 svcRtxThreadTerminate()函数。

4）激活态转为就绪态、阻塞态、终止态

激活态转为就绪态（图 1-1 中的④线）：正在执行的线程被高优先级线程抢占后进入就绪列表；或使用时间片轮询调度策略时，时间片耗尽，正在执行的线程让出 CPU；或被外部事件中断。

激活态转为阻塞态（图 1-1 中的⑤线）：正在执行的线程等待信号量、等待事件或者等待 I/O 资源等，如在 Mbed OS 中，调用 svcRtxThreadSuspend ()函数。

激活态转为终止态（图 1-1 中的⑦线）：如在 Mbed OS 中，调用 svcRtxThreadExit ()函数。

1.3.3　线程的三种基本形式

线程函数一般分为两个部分：初始化部分和线程体部分。初始化部分实现对变量的定义、初始化及设备的打开等，线程体部分负责完成该线程的基本功能。线程的一般结构如下：

```
void    thread_a ( uint32_t    initial_data )
{
        //初始化部分
        //线程体部分
}
```

线程的基本形式主要有单次执行线程、周期执行线程和资源驱动线程三种，下面介绍这三种线程的结构特点。

1．单次执行线程

单次执行线程是指线程在创建完之后只会被执行一次，执行完成后就会被销毁或阻塞的线程。其线程函数结构如下：

```
void    thread_a ( uint32_t initial_data )
{
      //初始化部分
      //线程体部分
      //线程函数销毁或阻塞
}
```

单次执行线程由三部分组成：线程函数初始化、线程函数执行和线程函数销毁或阻塞。线程函数初始化包括对变量的定义和赋值、打开需要使用的设备等；线程函数的执行是该线程的基本功能实现；线程函数的销毁或阻塞，即调用线程销毁或阻塞函数将自己从线程列表中删除。销毁与阻塞的区别在于销毁除了停止线程的运行，还将回收该线程所占用的所有资源，如堆栈空间等；而阻塞只是将线程描述符中的状态设置为阻塞态而已。举例来说，在水质监测系统中，主线程需要创建传感器采集数据线程和处理线程，需要对小灯、串口、传感器等外设进行初始化，当启动完传感器采集数据线程和处理线程后，就阻塞主线程，然后实时操作系统内核开始线程调度，此时的主线程就是单次执行线程。

2．周期执行线程

周期执行线程是指需要按照一定周期执行的线程。其线程函数结构如下：

```
void    thread_a ( uint32_t initial_data )
{
      //初始化部分
      ……
      //线程体部分
      while(1)
      {
            //循环体部分
      }
}
```

初始化部分同单次执行线程一样，包括对变量的定义和赋值、打开需要使用的设备等。与单次执行线程不同的是，周期执行线程的函数体内存在永久循环部分，由于该线程需要按照一定周期执行，因此在该线程内一般会调用延时函数，使线程转入阻塞态，进入延时列表中。当延时时间到时，线程就会转入就绪态，进入就绪列表中。举例来说，在水质监测系统中，我们需要得到被监测水域的酸碱度和各种离子的浓度，但是不需要时时刻刻都在监测数据，因为这些物理量的变化比较缓慢，所以使用传感器采集数据时可以调用延时函数，每隔半个小时采集一次数据，此时的物理量采集线程就是典型的周期执行线程。

3．资源驱动线程

除了上面介绍的两种线程类型，还有一种线程形式，那就是资源驱动线程。这里的资源主要指线程同步与通信中的事件、信号量、互斥量等。这种类型的线程比较特殊，它是操作系统特有的线程类型，因为只有在操作系统下才会出现资源共享使用的问题，同时引出操作系统中另一个主要问题，那就是线程同步与通信。该线程与周期执行线程的区别在于它的执行时间不是确定的，只有当它所要等待的资源可用时，它才会转入就绪态，否则被加入等待该资源的阻塞列表中。资源驱动线程的函数结构如下：

```
void    thread_a ( uint32_t initial_data )
{
    //初始化部分
    ……
    while(1)
    {
        //调用等待资源函数
        //线程体部分
    }
}
```

初始化部分和线程体部分与之前两种类型的线程类似，主要区别就是在线程体执行之前会调用等待资源函数，以等待资源实现线程体部分的功能。仍以水质监测系统为例，数据处理是在物理量采集完成后才能进行的操作，所以在系统中使用一个信号量用于两个线程之间的同步，当物理量采集线程完成时就会释放这个信号量，而数据处理线程一直在等待这个信号量，当等到这个信号量时，就可以进行下一步的操作。系统中的数据处理线程就是一个典型的资源驱动线程。

以上就是三种线程基本形式的介绍，其中周期执行线程和资源驱动线程从本质上来讲可以归结为一种，也就是资源驱动线程。因为时间也是操作系统的一种资源，只不过时间是一种特殊的资源，特殊在该资源是整个操作系统的实现基础，系统中大部分函数都是基于时间这一资源的，所以在分类中将周期执行线程单独作为一类。

1.4 本章小结

在实时操作系统下编程与无操作系统下编程相比有一个显著的优点，这个优点就是有个调度者，指挥、协调各个线程的运行，这样编程者可以把一个大工程分解成一个个小工程，

交由实时操作系统管理，这符合软件工程的基本原理。

线程是实时操作系统中最重要的概念之一。在实时操作系统下，把一个复杂的嵌入式应用工程按一定规则分解成一个个功能清晰的小工程，然后设定各个小工程的运行规则，交给实时操作系统管理，这就是基于实时操作系统的基本编程思想。这一个个小工程被称为线程，实时操作系统管理这些线程，被称为调度。读者可以分别从线程调度、软件设计及 CPU 等不同视角来理解线程。从线程调度视角来看，实时操作系统中的线程是一个功能清晰的小程序，是实时操作系统调度的基本单元；从软件设计视角来看，线程是独立的、相互作用的程序集合；从 CPU 视角来看，任何时刻只有一个线程占用 CPU。调度就是以合理的方式为每个线程分配时间，使之运行。

一个函数只有在给出其线程描述符及线程堆栈的情况下，才可以被称为线程，才能够被调度运行。线程一般有四种状态：终止态、阻塞态、就绪态和激活态。在任一时刻，线程被创建后所处的状态一定是以上四种状态之一。线程有三种基本形式，分别是单次执行线程、周期执行线程及资源驱动线程。