第 1 章

RT-Thread 概述

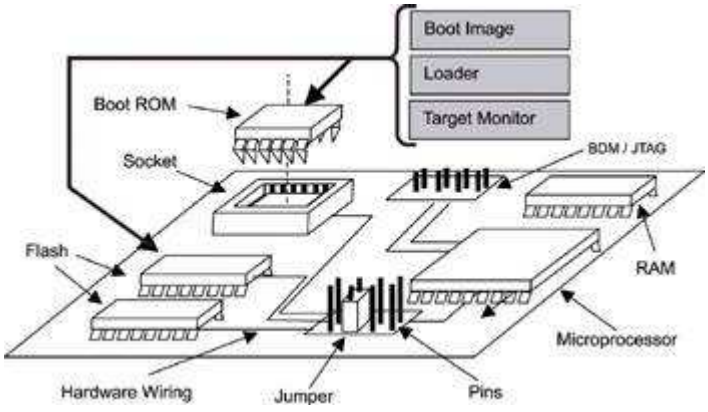
RT-Thread 是一个开放源代码的实时操作系统，是由国内 RT-Thread 工作室发起的一个开源项目，目前是在 GPL 的许可证下发布。读者可通过访问 http://www.rt-thread.org 来了解该项目的最新进展，包括在线的内核 API 参考及访问代码仓库获得最新的代码。

RT-Thread 是一个面向实时的操作系统，同时兼顾嵌入式设备大多具备一定实时系统的概念，所以它也是一个嵌入式操作系统，所以下面先简要介绍一下实时系统与嵌入式系统。

1.1 嵌入式系统

嵌入式系统是具备有特殊目的的计算系统，它具有特殊的需求，并运行预先定义好的任务。如常见的嵌入式系统：电视用的机顶盒，网络中的路由器等，它们都是为了一专用目的而设计的。从硬件资源上来讲，为了完成这一专有功能，嵌入式系统提供有限的资源，一般是恰到好处，在成本上满足一定的要求。从电子产品的角度来说，嵌入式系统最终会由一些芯片及电路组成，有时会包含一定的机械控制等，在控制芯片当中会包含一定的计算单元。总的来说，嵌入式系统提倡的是为了一个专用目的，其功能够用就好。

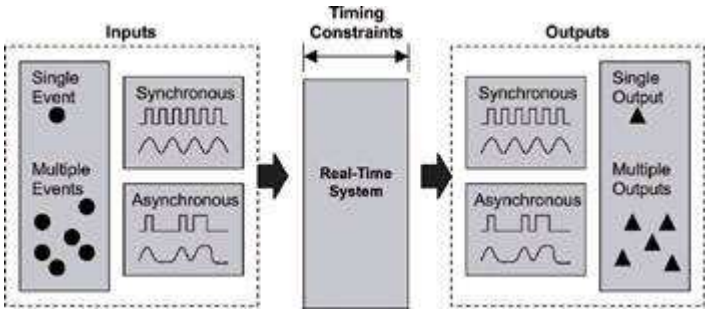
图 1-1 嵌入式系统



嵌入式系统中会包含微控制器，用于存放代码的 Flash，Boot Rom，运行时代码用到的内存，调试时需要的 JTAG 接口等。

1.2 实时系统

实时计算可以定义成这样一类计算，即系统的正确性不仅取决于计算的逻辑结果，而且还依赖于产生结果的时间，关键有两点：正确地完成和在给定的时间内完成，而且两者重要性是等同的。而针对于在给定的时间内功能性的要求可以划分出常说的两类实时系统，软实时和硬实时系统。可以先看一个示例图：



对于输入的信号、事件，实时系统应该能够在规定的时间内得到正确的响应，而不管这些事件是单一事件、多重事件还是同步信号或异步信号。对于一个具体的例子，可以考虑子弹射向玻璃杯的问题：
一颗子弹从 20 米处射出，射向一个玻璃杯。假设子弹的速度是 *v* 米/秒，那么经过 *t1=20/v* 秒后，子弹将击碎玻璃杯。而有一系统在看见子弹射出后，将把玻璃杯拿走，假设这整个过程将持续 *t2* 秒的事件。如果 *t2* < *t1*，那么这个系统可以看成是一个实时系统。

和嵌入式系统类似，实时系统上也存在一定的计算单元，对系统的环境、里面的应用有所预计，也就是很多实时系统所说的确定性：对一个给定事件，在一给定的事件 t 秒内做出响应。对多个事件、多个输入的响应的确定性构成了整个实时系统的确定性。

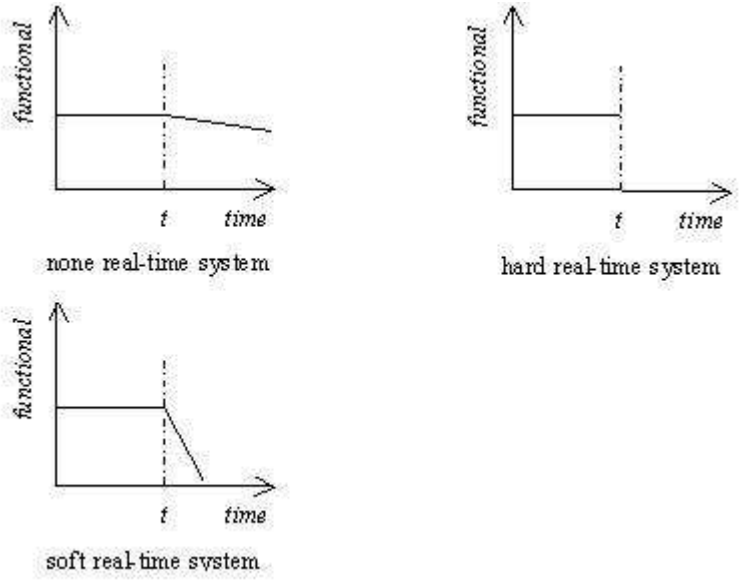
嵌入式系统的应用领域十分广泛，并不是其所针对的专用功能都要求实时性的，只有当系统中对任务有严格时间限定时，才有系统的实时性问题。具体的例子包括实验控制、过程控制设备、机器人、空中交通管制、远程通信、军事指挥与控制系统等。而对打印机这样一个嵌入式应用系统，人们并没有严格的时间限定，只有一个"尽可能快的"期望要求，因此，这样的系统称不上是实时系统。

1.3 软实时与硬实时

从上所述，实时系统是非常强调两点的：时间和功能的正确性。判断一个实时系统的正确性也正是这样，在给定的时间内正确地完成相应的任务。但也会有这种系统，偶尔也会在给定时间之外才能正确地完成任务，这种系统通常称为软实时系统。这也就构成了硬实时系统和软实时系统的区别。硬实时系统严格限定在给定的时间内完成任务，否则就可能导致灾难的发生，例如导弹的拦截，汽车引擎系统等。而软实时系统，可以允许一定的偏差，但是随着时间的偏移，整个系统的正确性也随之下降，例如一个 DVD 播放系统可以看成一个软实时系统，可以允许它偶尔的画面或声音延迟。

如下图 1-2 所示，从功能性、效用的角度上，实时系统可以看成：
非实时系统 随着给定时间 t 的推移，效用缓慢的下降。

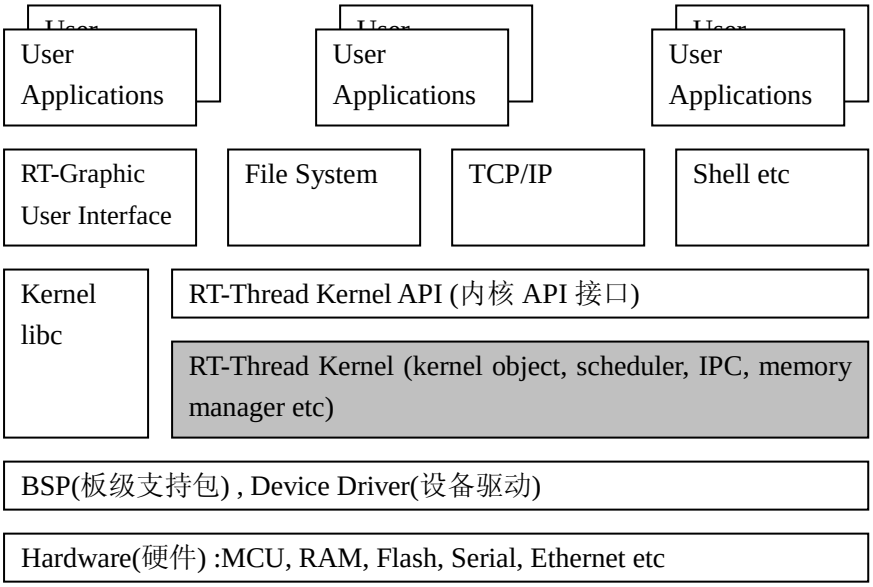
硬实时系统 在给定时间 t 之后，马上变为零值。
软实时系统 随着给定时间 t 的推移，效用迅速的走向零值。



1.4 RT-Thread 概述

在 RT-Thread 设计之初，就定下了宽度可伸缩的目标，既适合于一些资源非常紧张的系统，也适合于一些资源丰富对性能要求比较高的场合，甚至未来会支持多核高性能的系统。截止目前（2009 年初），RT-Thread 已经发展到了 0.3.x 版本，0.2.x 系列版本也在稳步完善中。

RT-Thread 包括了我们自主研发的强实时核心，它能够为上层服务、应用提供基础的实时支持，其总体结构如图所示：



本文主要介绍 RT-Thread 的实时核心以及一些移植方面的细节，各个组件会在后续的文档中陆续介绍。

1.5 RT-Thread 内核功能概览

1.5.1 任务/线程调度

在 RT-Thread 中线程是最小的调度单位，调度算法是基于优先级的全抢占式多线程调度，支持 256 个线程优先级，0 优先级代表最高优先级，255 优先级留给空闲线程使用；支持创建相同优先级线程，相同优先级的线程采用可设置时间片的轮转调度算法；调度器寻找下一个最高优先级就绪线程的时间是恒定的(O(1))。系统不限制线程数量的多少，只和物理平台的具体内存相关。

1.5.2 任务同步机制

系统支持信号量、互斥锁作为线程间同步机制。互斥锁采用优先级继存方式以防止优先级翻转问题。信号量的释放动作可安全用于中断服务例程中。同步机制支持线程按优先级等待或按先进先出方式获取信号量或互斥锁。

1.5.3 任务间通信机制

系统支持事件、快速事件、邮箱和消息队列等通信机制。事件支持多事件"或触发"及"与触发"，适合于线程等待多个事件情况。快速事件支持事件队列，事件发生时确定哪个线程阻塞在相应事件上的时间是确定的。邮箱中一封邮件的长度固定为 4 字节，效率较消息队列要高效。通信设施中的发送动作可安全用于中断服务例程中。通信机制支持线程按优先级等待或按先进先出方式获取。

1.5.4 时间管理

系统使用时钟节拍来完成同优先级任务的时间片轮转调度;线程对内核对象的时间敏感性是通过系统定时器来实现的，此外，定时器也支持一次性超时及周期性超时。

1.5.5 内存管理

系统支持静态内存池管理及动态内存堆管理。从静态内存池中获取内存块时间恒定，而当内存池空时，可把申请内存块的线程阻塞(或立刻返回，或等待一段时间仍未获得返回，取决于内存块申请时设置的等待时间)，当其他线程释内存块到内存池时，将把阻塞线程唤醒。动态堆内存管理对于不同的系统资源情况，提供了面向小内存系统的管理算法及大内存系统的 SLAB 内存管理算法。

1.5.6 设备管理

系统实现了按名称访问的设备管理子系统，可按照统一的 API 界面访问硬件设备。