CHAPTER 3

第 3 章

操作系统产品案例

3.1 μC/OS-Ⅱ嵌入式操作系统简介

嵌入式实时操作系统(Real-time Operating System,RTOS,简称嵌入式操作系统)是一种支持嵌入式系统应用的操作系统软件,是嵌入式系统极为重要的组成部分,通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系统具有通用操作系统的基本特点,如能够有效管理越来越复杂的系统资源;能够将硬件虚拟化,使得开发人员从繁忙的驱动程序移植和维护中解脱出来;能够提供库函数、驱动程序、工具集以及应用程序。与通用操作系统比较,嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点。

嵌入式操作系统具有以下三个方面的优点。

(1) 提高了系统的可靠性:在控制系统中,出于安全考虑,要求系统不能崩溃,而且要有自愈能力。因此,在硬件和软件设计方面都要求高可靠性和高抗干扰性,尽可能地减少安全漏洞和不可靠的隐患。较早的前后台系统软件设计在遇到强干扰时,运行程序容易产生异常、出错、跑飞,甚至死循环,造成了系统的崩溃。而在嵌入式操作系统管理的系统中,干扰可能只是引起若干进程中的一个被破坏,可以通过系统运行的系统监控进程对其进行修复。

(2) 提高了开发效率,缩短了开发周期:在嵌入式操作系统环境下,开发复杂的应用程序,通常可以按照软件工程中的解耦原则将整个程序分解为多个任务模块,每个任务模块的调试、修改几乎不影响其他模块。

(3) 嵌入式实时操作系统充分发挥了 32 位 CPU 的多任务潜力:但是,使用嵌入式操作系统需要额外的 ROM/RAM 开销,2%～5%的 CPU 额外负荷,以及内核的开销。

嵌入式操作系统是满足嵌入式系统并发需求、提高嵌入式软件开

发效率和可移植性的重要手段，也是实时应用程序必不可少的运行平台，一般可分为两大类：

（1）专门为嵌入式应用设计的 RTOS，如 μC/OS-Ⅱ、VxWorks、Nucleus、pSOS、AVIS、IOS、DeltaOS、PalmOS、EPOC、Windows CE、PPSM、ThreadX 和 Itron 等。

（2）经过扩展，提供嵌入式实时性能的通用操作系统，如 RT-Linux、嵌入式 Linux 和 Windows XP Embedded 等。

嵌入式操作系统一般采用微内核结构（如图 3-1 所示），基于优先权抢占的调度策略，具有任务管理、任务间同步和通信（如信号量、消息队列、异步信号、共享内存、管道等）、内存管理和中断管理等功能。衡量 RTOS 内核的技术指标主要有上下文切换时间（Context Switch Time）、中断响应时间（Interrupt Response Time）、内核代码最小尺寸、调度器实现的算法、系统调用的数量、系统对象的限制、内存保护和多处理器支持等。

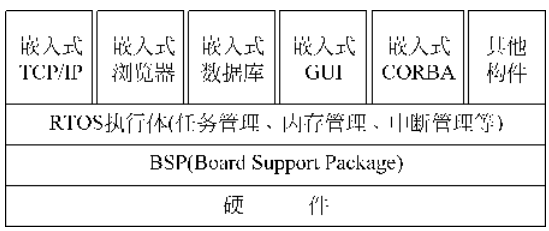

图 3-1　RTOS 的微内核结构

嵌入式系统的需求多种多样，不同的嵌入式操作系统又具有各自的特点，选择嵌入式操作系统时，主要确定嵌入式操作系统的特点是否满足应用需求，除考虑上述内核性能指标外，还应考虑下述问题：

- 除内核外，RTOS 提供的构件（如 TCP/IP 协议栈、嵌入式数据库、嵌入式 GUI 等构件）功能、性能如何，能否满足应用需求。
- 提供的开发平台功能和易用性如何。
- RTOS 的结构是否合理，这将影响到能否方便地增加新设备的驱动程序和应用程序的移植。
- 版权（license）和财务问题，包括 RTOS 和开发平台的一次性购置费用、RTOS 的版费（是 Royalty-pay 还是 Royalty-free）以及未来的升级费用等。
- 标准化支持，RTOS 的 API 是否符合相应标准，如 POSIX 1003.4 或 Itron。
- RTOS 的可剪裁问题。
- 整套产品的成熟度和可靠性以及市场竞争能力如何，是否具有持续发展的能力。

μC/OS-Ⅱ，读做“micro COS 2”，是一种免费公开源代码、结构小巧、具有可抢占实时内核的嵌入式操作系统（Real-time Operating System，RTOS），在世界各地都获得了广泛的应用。其前身是 μC/OS，最早出自于 1992 年美国嵌入式系统专家 Jean J. Labrosse 在《嵌入式系统编程》（*Embedded System Programming*，http://www.embedded.com）杂志 5 月和 6 月刊上刊登的文章连载，并将 μC/OS 的源码发布在该杂志的 BBS 上。μC/OS-Ⅱ是一种专门为嵌入式设备设计的实时内核，具有如下特点：

- μC/OS-Ⅱ具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点，最小内核可编译至2KB。
- 绝大部分代码是用C语言编写的，只有CPU硬件相关部分是用汇编语言编写的，总量约200行的汇编语言部分被压缩到最低限度，为的是便于移植到任何一种其他CPU上。
- 用户只要有标准的ANSI C交叉编译器、汇编器、连接器等软件工具，就可以将μC/OS-Ⅱ嵌入到开发的产品中。

严格地说，μC/OS-Ⅱ只是一个实时操作系统(Real-time Operating System，RTOS)内核，仅包含了任务调度、任务管理、时间管理、内存管理和任务间的通信和同步等基本功能，没有提供输入输出管理、文件系统、网络等额外服务。但由于μC/OS-Ⅱ良好的可扩展性和源码开放，这些非必须的功能完全可以由用户自己根据需要分别实现。

μC/OS-Ⅱ目标是实现一个基于优先级调度的抢占式的实时内核，并在这个内核之上提供最基本的系统API服务，如信号量、邮箱、消息队列、内存管理、中断管理、任务管理等。

目前μC/OS-Ⅱ已经被移植到40多种不同结构的CPU上运行，包括从8位到64位的各种CPU。尤其值得一提的是，该系统自从2.51版本之后，就通过了美国联邦航空管理局(Federal Aviation Administration，FAA)认证，可以运行在诸如飞机等对安全要求极为苛刻的系统之上。

鉴于μC/OS-Ⅱ可以免费获得代码，对于嵌入式RTOS而言，选择μC/OS-Ⅱ无疑是最经济的选择。μC/OS-Ⅱ操作系统的源代码和目标码可以在有资质的大学中免费提供给学生，用于非商业性目的，换言之，μC/OS-Ⅱ操作系统用于教学目的，不需要使用许可证和付费。如果以赢利为目的，将μC/OS-Ⅱ操作系统的目标码嵌入到产品中，则应得到“目标代码销售许可证”，这需要付费，具体价格要同作者Jean J. Labrosse联系。

3.2 系统结构

μC/OS-Ⅱ的内核结构如图3-2所示，其中，应用软件层是μC/OS-Ⅱ之上的代码，μC/OS-Ⅱ负责为其提供相应的接口函数；μC/OS-Ⅱ内核中与处理器无关部分和与应用相关部分都是用ANSI C语言编写的，在移植时无须更改；在移植过程中，具体涉及的文件是图中与处理器相关部分，包括一个头文件OS_CPU.H，一个汇编文件OS_CPU_A.S和一个C代码文件OS_CPU_C.C。

μC/OS-Ⅱ可以大致分成核心部分、任务处理、时间处理、任务同步与通信、CPU的移植等5个部分，其中：

(1) 核心部分(OSCore.c)。操作系统的处理核心，包括操作系统初始化、操作系统运行、中断进出的前导、时钟节拍、任务调度、事件处理等部分，能够维持系统基本工作的部分都在这里。

(2) 任务处理部分(OSTask.c)。任务处理部分中的内容都是与任务的操作密切相关的，包括任务

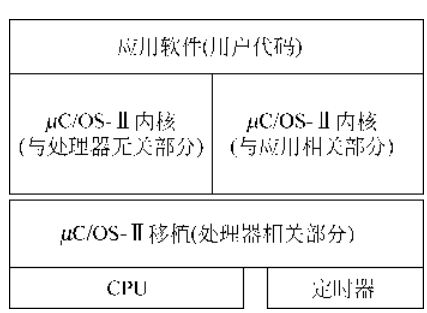

图3-2 μC/OS-Ⅱ内核结构

的建立、删除、挂起、恢复等。因为 μC/OS-Ⅱ是以任务为基本单位调度的，所以这部分内容也相当重要。

(3) 时钟部分(OSTime.c)。μC/OS-Ⅱ中的最小时钟单位是 timetick(时钟节拍)，任务延时等操作是在这里完成的。

(4) 任务同步和通信部分。为事件处理部分，包括信号量、邮箱、邮箱队列、事件标志等部分，主要用于任务间的互相联系和对临界资源的访问。

(5) 与 CPU 的接口部分。指 μC/OS-Ⅱ所使用 CPU 硬件的移植部分。由于 μC/OS-Ⅱ是一个通用性的嵌入式操作系统，所以在一些底层关键技术上的实现，还是需要根据具体 CPU 的具体内容和要求作相应的移植。这部分内容由于牵涉到 CPU 硬件，所以通常用汇编语言编写，主要包括中断级任务切换的底层实现、任务级任务切换的底层实现、时钟节拍的产生和处理、中断的相关处理部分等内容。

对 μC/OS-Ⅱ内核中常用的管理机制描述如下。

1. 任务管理

μC/OS-Ⅱ中最多可以支持 64 个任务，分别对应优先级 0～63，其中 0 为最高优先级，63 为最低级，系统保留了 4 个最高优先级的任务和 4 个最低优先级的任务，所有用户可以使用的任务数最多为 56 个。

μC/OS-Ⅱ提供了任务管理的各种函数调用，包括创建任务、删除任务、改变任务的优先级、任务挂起和恢复等。系统初始化时会自动产生两个任务：一个是空闲任务，它的优先级最低，该任务仅给一个整形变量做累加运算；另一个是系统任务，它的优先级为次低，该任务负责统计当前 CPU 的利用率。

2. 时间管理

μC/OS-Ⅱ的时间管理是通过定时中断实现的，该定时中断一般为 10ms 或 100ms 发生一次，时间频率取决于用户对 CPU 硬件的定时器编程实现。

中断发生的时间间隔是固定不变的，该中断也成为一个时钟节拍。μC/OS-Ⅱ要求用户在定时中断的服务程序中，调用系统提供的与时钟节拍相关的系统函数，如中断级的任务切换函数和系统时间函数。

3. 内存管理

在 ANSI C 中是使用 malloc 和 free 两个函数动态分配和释放内存。但在嵌入式实时系统中，多次这样的操作会导致内存碎片，且由于内存管理算法的原因，malloc 和 free 的执行时间也是不确定。

μC/OS-Ⅱ中将连续的大块内存按分区管理，每个分区中包含数个大小相同的内存块，但不同分区之间的内存块大小可以不同。用户需要动态分配内存时，系统选择一个适当的分区，按块来分配内存。释放内存时将该块放回它以前所属的分区，这样能有效解决碎片问题，同时执行时间也是相对固定的。

4. 任务间通信与同步

对一个多任务的操作系统来说，任务间的通信和同步是必不可少的。μC/OS-Ⅱ中提供了四种同步对象，分别是信号量、邮箱、消息队列和事件，所有这些同步对象都有创建、等待、发送、查询等操作，用于实现进程间的通信和同步。

5. 任务调度

μC/OS-Ⅱ采用的是可抢占型实时多任务内核，在任何时候都运行就绪的最高优先级任务。

μC/OS-Ⅱ的任务调度是完全基于任务优先级的抢占式调度，也就是最高优先级的任务一旦处于就绪状态，则立即抢占正在运行的低优先级任务的处理器资源。为了简化系统设计，μC/OS-Ⅱ规定所有任务的优先级不同，因为任务的优先级也同时唯一标志了该任务本身。

任务调度将在以下情况下发生：

(1) 高优先级的任务因为需要某种临界资源，主动请求挂起，让出处理器，此时将调度就绪状态的低优先级任务获得执行，这种调度也称为任务级的上下文切换。

(2) 高优先级的任务因为时钟节拍到来，在时钟中断的处理程序中，内核发现高优先级任务获得了执行条件(如休眠的时钟到时)，则在中断态直接切换到高优先级任务执行，这种调度也称为中断级的上下文切换。

这两种调度方式在μC/OS-Ⅱ的执行过程中非常普遍，一般来说前者发生在系统服务中，后者发生在时钟中断的服务程序中。

调度工作的内容可以分为两部分：最高优先级任务的寻找和任务切换。最高优先级任务的寻找是通过建立就绪任务表实现的，在μC/OS-Ⅱ中，每一个任务都有独立的堆栈空间，并有一个称为任务控制块(Task Control Block，TCB)的数据结构，其中第一个成员变量就是保存的任务堆栈指针。任务调度模块首先用变量OSTCBHighRdy记录当前最高级就绪任务的TCB地址，然后调用OS_TASK_SW()函数进行任务切换。

3.3 系统移植与扩展

1. μC/OS-Ⅱ移植条件

要将μC/OS-Ⅱ移植到一种新的CPU上，并使之正常运行，处理器必须满足以下要求：

- 处理器的C编译器能产生可重入型代码。
- 处理器支持中断，并且能产生定时中断。
- 在程序中可以开/关中断。
- 处理器能支持一定数量的数据存储硬件堆栈。
- 处理器有将堆栈指针以及其他CPU寄存器的内容读出、并存储到堆栈或内存中去的指令。

2. μC/OS-Ⅱ移植问题

移植过程中，需要注意以下三个问题。

(1) 数据类型的重定义：对于操作系统，一般都使用高级语言进行开发。高级语言都有自己的数据类型，但由于不同的处理器字长和存储模式的不同，同一数据类型在不同处理器中会有不同的解释。

(2) 堆栈结构的设计：当同一个操作系统应用于不同处理器或同一处理器的不同应用系统时，由于各应用系统所追求的性能特点各异，就会要求与性能有很大关系的堆栈结构尽可能与本系统所追求的性能一致。

(3) 任务切换时的状态保存与恢复：这是多任务操作系统最主要的工作，也是最频繁的工作，所以任务切换在实现时的正确与否是操作系统运行时的基本保证，同时它的简洁与

否决定操作系的效率。

3. μC/OS-Ⅱ移植解决方案

对于前面提到的三个问题，它们大都与处理器密切有关。μC/OS-Ⅱ的大部分程序用C语言编写，只有与处理器有关的部分是用汇编语言完成。移植所要做的工作也是在不同的处理器上用不同的汇编语言改写与处理器有关的代码及其他与处理器特性相关的部分。

在μC/OS-Ⅱ中，与处理器有关的文件有三个，它们分别是OS_CPU.H、OS_CPU_A.S和OS_CPU_C.C，移植工作主要是对这些文件进行依据处理器特性的改写。下面介绍这三个文件的作用，以及如何在移植过程中有效地解决前面提出的三个问题。

1) OS_CPU.H

因为不同的处理器有不同的字长及存储模式，为了确保μC/OS-Ⅱ的可移植性，在OS_CPU.H中进行了一系列的类型定义。μC/OS-Ⅱ代码不使用C语言的short、int和long等数据类型，因为它们是与编译器相关的，不可移植。相反，在μC/OS-Ⅱ中定义的整型数据结构既是可移植的又是直观的。在具体进行移植的整数数据结构定义时，需要参考编译环境的编译器手册，以确定具体类型的字节长度。具体的部分整数数据定义如：

```
typedef unsigned char BOOLEAN;
typedef unsigned char INT8U;          /* 无符号8位整数 */
typedef unsigned char INT8S;          /* 有符号8位整数 */
```

另外，在该文件中，还定义了变量OS_STK_GROWTH。在不同的微处理器或者微控制器中，堆栈的生长方向有可能不同。虽然绝大多数的微处理器的堆栈是从上往下长，但是某些处理器是用从下往上生长的堆栈方式工作的。μC/OS-Ⅱ利用定义的变量OS_STK_GROWTH很好地解决了堆栈生长方式的不同，设置OS_STK_GROWTH为0表示堆栈从下往上长，设置OS_STK_GROWTH为1表示堆栈从上往下长。

为了μC/OS-Ⅱ能够保护临界段代码免受多任务或中断服务程序的破坏，与所有的实时内核一样，μC/OS-Ⅱ需要先禁止中断再访问代码的临界段，并且在访问完毕后重新开中断。中断禁止时间是实时内核的重要指标之一，因为它将影响到用户的系统对实时事件的响应能力。虽然μC/OS-Ⅱ尽量使中断禁止时间达到最短，但是μC/OS-Ⅱ的中断禁止时间还主要依赖于处理器结构和编译器产生的代码质量。通常每个处理器都会提供一定的指令禁止/开中断，因此用户的C编译器必须要有一定的机制直接从C中执行这些操作。有些编译器能够允许用户在C源代码中插入汇编语言声明，这样就使得插入处理器指令打开和禁止中断变得容易。其他一些编译器实际上包括了语言扩展功能，可以直接从C中开和禁止中断。为了隐藏编译器厂商提供的具体实现方法，μC/OS-Ⅱ定义了两个宏禁止和开中断：OS_ENTER_CRITICAL()和OS_EXIT_CRITICAL()。

2) OS_CPU_C.C

该文件包含10个函数，分别是OSTaskStkInit()、OSTaskCreateHook()、OSTaskDelHook()、OSTaskSwHook()、OSTaskIdleHook()、OSTaskStatHook()、OSTimeTickHook()、OSInitHookBegin()、OSInitHookEnd()、OSTCBInitHook()。除第一个堆栈初始化函数OSTaskStkInit()必须实现外，其他函数需声明，可以不包含代码，当需要额外地扩展功能时可以将这些函数加入。函数OSTaskStkInit()用于初始化任务的堆栈结构，不同的处理器在进行函数调用时需保存的寄存器不同及用户设计的堆栈结构不同，所以必须根据所要移

植的处理器进行重写。

3) OS_CPU_A.S

按照 μC/OS-Ⅱ本身所要求的代码结构,在该文件中,要求用户编写四个汇编语言函数:OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()和 OSTickISR()。这四个函数主要用来进行多任务及中断服务程序之间的切换,任何操作系统在实行任务调度时,都需要对处理器中的部分寄存器进行保存和恢复,所以对不同的处理器这部分函数需要用不同的汇编语言进行改写,且各寄存器保存的次序要同 OSTaskStkInit()函数中初始化的堆栈结构一致。

下面简要介绍一下这四个函数的作用。

(1) OSStartHighRdy():该函数在系统调用 OSStart()启动多任务之后,从处于就绪态且优先级最高的任务 TCB 中获得该任务的上下文堆栈指针 sp,通过 sp 依次将任务的 CPU 上下文恢复,然后系统将控制权交给优先级最高的任务。该函数仅在多任务启动时被执行一次,用来启动第一个任务,即优先级初始化最高的任务。

(2) OSCtxSw():当某任务被阻塞时便会调用该函数完成任务级的上下文切换。该函数先将被阻塞任务的 CPU 上下文保存到该任务的上下文堆栈中,然后获得此时系统中最高优先级任务的上下文堆栈指针 sp,并根据该 sp 恢复最高优先级任务的 CPU 上下文,最后使优先级最高的任务执行,完成一次任务切换。

(3) OSIntCtxSw():该函数完成中断级的上下文切换。在中断服务子程序的最后,如果发现系统中有比被中断任务优先级更高的任务处于就绪态,便会调用该函数在中断退出后并不返回被中断任务,而是直接调度就绪且具有最高优先级的任务执行,这样做的目的是能够尽快地让高优先级的任务得到响应,保证系统的实时性能。该函数的原理基本上与任务级的上下文切换相同,但是由于进入中断时已经保存过被中断任务的 CPU 上下文,因此不需要再进行类似的上下文保存操作,只需要对堆栈指针做相应的调整。

(4) OSTickISR():系统的时钟中断服务程序,同样需要用汇编在进入中断服务程序时保存寄存器,及退出时的恢复。但是,此函数实现的功能可以用其他形式完成,所以,该函数在具体实现时并不是必须的。

μC/OS-Ⅱ是利用这些函数完成任务切换时的状态保存与恢复,从而解决前面提出的第三个问题。

4. μC/FS 嵌入式文件系统

在嵌入式应用中,嵌入式设备中使用的存储器一般不是 PC 上的硬盘,取而代之的是 Flash 闪存芯片、记忆棒、小型闪存卡等,专为嵌入式系统设计的存储设备。Flash 是目前嵌入式系统中广泛应用的主流存储器,在 Flash 上构建一种文件系统,需要考虑 Flash 的硬件特点。同时,根据应用的场合,也需要考虑嵌入式领域的需求。基于上述特殊应用环境和 Flash 芯片特性的双重考虑,嵌入式平台对文件系统提出了如下的要求。

- 崩溃恢复:嵌入式系统的运行环境一般比较恶劣,但同时对可靠性有较高的要求,这对嵌入式文件系统提出了较高的要求,无论程序崩溃或系统掉电,都不能影响文件系统的一致性和完整性。文件系统的写入、垃圾回收等操作对系统异常中断都非常敏感,极易造成数据丢失和数据垃圾,因此在文件系统设计和选用时应多加考虑。
- 损耗平衡:由于 Flash 本身的特点,Flash 的存储块擦除次数有限,文件系统对 Flash 的使用需考虑该特性,最好能均匀使用 Flash 的每个块,以延长 Flash 的使用寿命。

- 垃圾回收：任何存储器在被分配、使用一段时间后，都会出现空闲区和文件碎片，可能导致系统空间不够用，因此需要进行垃圾回收操作，以保证存储空间的高效使用。通常 Flash 擦除操作以块为单位，垃圾回收也应以块为单位。
- 高效的空间管理机制：目前在应用领域中 Flash 的存储空间都相对较小，对空间的使用和管理都提出了更高的要求。

μC/FS 是 μC/OS-Ⅱ扩展的嵌入式文件系统，用 ANSI C 编写，因此，可在任何架构的 CPU 上运行，其特性包括：

- 支持 MS-DOS/MS-Windows 兼容的 FAT12 和 FAT16。
- 支持多种设备驱动程序。通过使用多种不同的设备驱动程序，μC/FS 可以同时存储多种不同类型的存储设备。
- 支持多种存储设备。
- 支持多种操作系统。μC/FS 能够很容易地集成到不同的操作系统中，因此使得用户能够在多线程环境下进行文件操作。
- 具有简单的设备驱动数据结构。μC/FS 设备驱动仅需要用户提供最基础的读写块的函数，所以它的移植非常的方便。
- 为用户提供了与 ANSI C sudio.h 类似的 API。
- 提供了 SmartMedia 卡支持。
- 提供了 MultiMedia & SD 卡支持。

μC/FS 的结构一般分为四个层次，如图 3-3 所示。

- API 层：该层是 μC/FS 和用户应用之间的接口，包含了若干文件操作的函数，如 FS_Fopen、FS_Fwrite 等，API 层将用户对文件的操作命令传递到文件系统层。目前，文件系统层只支持 FAT 格式，但 API 层能同时处理不同的文件系统层，因此，能够同时使用 FAT 文件系统和其他文件系统。

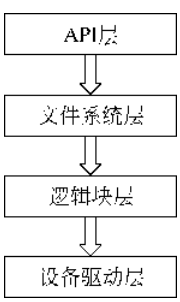

图 3-3 μC/FS 基本结构

- 文件系统层：该层将文件的操作命令传递给逻辑块层。在命令被传递之后，文件系统便调用逻辑块层，并为某设备指定相应的设备驱动。
- 逻辑块层：该层的功能主要是同步设备驱动的访问，并为文件系统层提供接口。它调用相应的设备驱动程序完成块操作。
- 设备驱动层：该层提供对硬件进行存储操作的底层函数。

3.4 系统应用程序开发

1. μC/OS-Ⅱ应用程序基本结构

每一个 μC/OS-Ⅱ应用程序至少要有一个任务，而每一个任务必须被写成无限循环的形式。以下是一种典型结构：

```
void task(void * pdata)
{
  INT8U err;
  InitTimer();                    //可选
```

```
    For(; ; )
      {
      //你的应用程序代码
       ⋮
      OSTimeDly(1);              //可选
      }
}
```

以上就是一个 μC/OS-Ⅱ应用程序的基本结构，至于为什么要写成无限循环的形式？因为系统为每一个任务保留一个堆栈空间，由系统在任务切换的时候恢复上下文，并执行一条 reti 指令返回。如果允许任务执行到最后一个花括号（那一般都意味着一条 ret 指令）的话，很可能会破坏系统堆栈空间，从而使应用程序的执行不确定，换句话说，就是程序“跑飞”了。所以，每一个任务必须被写成无限循环的形式。程序员一定要相信，自己的任务是会放弃 CPU 使用权的，而不管是系统强制（通过中断服务程序 ISR）还是主动放弃（通过调用 μC/OS-Ⅱ API）。

现在介绍一下上面程序中的 InitTimer()函数。该函数应该由系统提供，程序员有义务在优先级最高的任务内调用它，而且不能在 for 循环内调用。注意，这个函数是和所使用的 CPU 相关的，每种系统都有自己的 Timer 初始化程序。在 μC/OS-Ⅱ的帮助手册内，作者特地强调绝对不能在 OSInit()或者 OSStart()内调用 Timer 初始化程序，那会破坏系统的可移植性，同时带来性能上的损失。所以，一个折中的办法就是在优先级最高的程序内调用，这样可以保证当 OSStart()调用系统内部函数 OSStartHighRdy()开始多任务后，首先执行的是 Timer 初始化程序。或者，专门开一个优先级最高的任务，只做一件事情，那就是执行 Timer 初始化，之后通过调用 OSTaskSuspend()将自己挂起来，永远不再执行，不过这样会浪费一个 TCB 空间。对于那些 RAM 吃紧的系统来说，还是不用为好。

2. 一些重要的 μC/OS-Ⅱ API 介绍

任何一个操作系统都会提供大量的 API 供程序员使用，μC/OS-Ⅱ也不例外。由于 μC/OS-Ⅱ面向的是嵌入式开发，并不要求大而全，所以内核提供的 API 也就大多和多任务息息相关，主要包括任务类、消息类、同步类、时间类和临界区与事件类五类 API。任务类和时间类是必须首先掌握的两种 API，下面介绍其中六个比较重要的 API。

1) OSTaskCreate 函数

该函数应该至少在 main 函数内调用一次，在 OSInit 函数调用之后调用，作用是创建一个任务。目前有四个参数，分别是任务的入口地址、任务的参数、任务堆栈的首地址和任务的优先级。调用本函数后，系统首先从 TCB 空闲列表内申请一个空的 TCB 指针，然后将根据用户给出参数初始化任务堆栈，并在内部任务就绪表内标记该任务为就绪状态，最后返回，这样一个任务就创建成功了。

2) OSTaskSuspend 函数

该函数很简单，一看名字就该明白它的作用，它可以将指定的任务挂起。如果挂起的是当前任务的话，那么还将引发系统执行任务切换先导函数 OSShed 进行一次任务切换。这个函数只有一个参数，是指定任务的优先级。那为什么是优先级呢？事实上在系统内部，优先级除了表示一个任务执行的先后次序外，还起着区分每一个任务的作用，换句话说，优先级也就是任务的 ID。所以，在 μC/OS-Ⅱ中，不允许出现相同优先级的任务。

3）OSTaskResume 函数

该函数和上面的函数正好相反，它用于将指定的已经挂起的函数恢复成就绪状态。如果恢复任务的优先级高于当前任务，那么还将引发一次任务切换。其参数类似与OSTaskSuspend 函数，为指定任务的优先级。需要特别说明是，本函数并不要求和OSTaskSuspend 函数成对使用。

4）OS_ENTER_CRITICAL 宏

很多人都以为它是个函数，其实不然，仔细分析一下 OS_CPU. H 文件，它和下面谈到的 OS_EXIT_CRITICAL 都是宏，都涉及特定 CPU 的实现，一般都被替换为一条或者几条嵌入式汇编代码。由于系统希望向上层程序员隐藏内部实现，故而一般都宣称执行此条指令后系统进入临界区。其实，它就是关个中断而已，只要任务不主动放弃 CPU 使用权，别的任务就没有占用 CPU 的机会，相对这个任务而言，它独占了 CPU，所以说进入临界区了。这个宏尽量少用，因为它会破坏系统的一些服务，尤其是时间服务，并使系统对外界响应性能降低。

5）OS_EXIT_CRITICAL 宏

该宏是和上面介绍的宏一般配套使用，在系统手册里的说明是退出临界区，其实它就是重新开中断。需要注意的是，它必须和上面的宏成对出现，否则会带来意想不到的后果，最坏的情况下系统会崩溃。程序员尽量少使用这两个宏调用，因为它们的确会破坏系统的多任务性能。

6）OSTimeDly 函数

该函数一般是程序员调用最多的一个函数，完成功能很简单，就是先挂起当前任务，然后进行任务切换，在指定的时间到来之后，将当前任务恢复为就绪状态，但是并不一定运行，如果恢复后是优先级最高就绪任务的话，那么运行之。简单说，就是可以让任务延时一定时间后再次执行它，或者说，暂时放弃 CPU 的使用权。一个任务可以隐式调用这些导致放弃 CPU 使用权的 API，但那样多任务性能会大大降低，因为此时系统仅依靠时钟机制在进行任务切换。一个设计风格良好的任务，应该在完成一些操作主动后放弃资源的使用权。

有关 μC/OS-Ⅱ操作系统组成、移植、开发工具等更详细的内容见教材《μC/OS-Ⅱ——源码公开的实时嵌入式操作系统》及配套光盘。

CHAPTER 4

第 4 章

软件工程环境产品案例

4.1 嵌入式软件开发平台简介

从软件工程角度看，CASE(Computer Aided Software Engineering)工具最初由单独的软件工具(tool)发展到工作平台(workbench)，不断发展、完善，最终形成 CASE 开发环境(CASE development environment)。开发工具是用于开发应用程序的一类系统软件，从软件结构层次角度看，介于操作系统和应用程序之间，支持应用程序开发的某一功能或过程，如编辑器(editor)、编译器(compiler)、调试器(debugger)、性能优化分析器(profiler)等。工作平台是支持一到两个行为(activity)的工具集(tools set)，这里的行为是指一个相关的任务集，如编码行为包括编辑、编译、链接、调试等任务。行为和软件工程生命周期模型中的一个阶段并不完全相同，一个行为的任务可以跨越数个阶段，如一个编码工作平台可以用于快速原型生成，也可以用于实现、集成和维护阶段。CASE 开发环境是支持软件生命周期中大部分阶段的工具集，包括需求分析、规格说明、系统设计、编码、测试、产品分配与维护等。CASE 开发环境一般包括四类：基于编程语言的环境、面向结构的环境、工具箱环境和集成环境。

嵌入式软件开发平台(development platform)是为用户开发(包括需求分析、规格说明、设计、编码、测试、产品分配与维护等阶段)嵌入式应用程序而提供的高起点、综合的支撑环境(supporting environment)，包括面向领域的应用程序基本框架、可重用的构件库、参考设计、应用示例、开发工具集、RTOS、相关文档以及对平台进行管理、配置的设施等技术实体，主要功能是将工具集成在一起支持某种软件开发方法或与某种软件开发模型相适应，是用户开发应用程序的重要基础，强调知识成果的积累和重用，是平台开发模式思想的集中体现。

通常，嵌入式软件开发平台支持嵌入式软件生命周期中的一个或多个开发阶段，甚至全部阶段，如设计平台、IDE(主要支持编码阶段