Broadview®
www.broadview.com.cn

嵌入式实时操作系统

μC/OS原理与实践（第2版）

卢有亮 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

嵌入式实时操作系统

μC/OS原理与实践

（第2版）

卢有亮 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书内容包括：实时操作系统基础、任务管理、中断和时间管理、事件管理、消息管理、内存管理、移植、工程实践及μC/OS-III分析、移植与应用实践等。本书内容翔实，图文并茂，采用逐步深入、反复印证的方法，从数据结构的设计入手，再到代码分析、示例验证的剖析方法，逐层深入讲解，给出在虚拟平台下的移植示例和针对各章内容示例，并给出了基于ARM Cortex M3内核的STM32系统上移植和工程实例。

本书适用于计算机、电子、通信、自动化及相关专业大学本科、研究生，也适用于广大嵌入式开发工程技术人员、电子技术研究人员、操作系统研究人员。

图书在版编目（CIP）数据

嵌入式实时操作系统μC/OS原理与实践 / 卢有亮编著.— 2版.—北京：电子工业出版社，2014.4

ISBN 978-7-121-22517-8

Ⅰ.①嵌… Ⅱ.①卢… Ⅲ.①实时操作系统 Ⅳ.①TP316.2

中国版本图书馆CIP数据核字（2014）第033585号

策划编辑：张月萍

责任编辑：徐津平

特约编辑：赵树刚

印　　刷：北京中新伟业印刷有限公司

装　　订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路173信箱　　邮编：100036

开　　本：787×1092　1/16　　印张：18　　字数：461千字

印　　次：2014年4月第1次印刷

定　　价：49.00元

前　言 PREFACE

智能系统的盛行使21世纪前10年成为手指尖在触摸屏上滑动拖曳的时代。不少高级科技人员解决了一个又一个困难，使裸奔的软件在中断和循环的纠缠中走了很远很久。在ARM处理器走出江湖之后，处理器的处理速度和闪存Flash、静态存储器SRAM的容量都飞速提升，高性能处理器的出现也使高端的复杂处理程序采用嵌入式来实现，如物联网、智能手机。存储容量的扩充使嵌入式操作系统有了用武之地。在STM32使用的ARM Cortex处理器中，具有主堆栈MSP和进程堆栈PSP，具有PendSV和Systick中断，这些很明显是配合了μC/OS操作系统。

本书的第1版内容充实，有流程图等辅助手段，笔者在博客提供了PPT、实验教程和代码，受到了读者的好评，并被一些有所作为的老师引为教材，不少工程师也因此尝到了熟读代码的甜头。因此，第2版的创作有了足够的动力。这本书是笔者独自完成的，第2版的改版经过和很多读者的交流及论坛的咨询交流。第1版的缺陷也显而易见，缺少了硬件平台，只是在VC下仿真学习。因此，笔者设计的亮点STM32开发板弥补了这一个缺陷，也是第2版修改和增加的移植、工程实例及μC/OS-III的基础平台。当然，实验平台是选项，如果喜欢在VC下学习仍然是可以的，而在其他嵌入式系统及开发板下对笔者提供的代码的配置信息进行修改，也可以胜任。

第2版中将提供在STM32（ARM Cortex内核系列芯片）下的移植和例程，增加应用性的工程示例。2013年μC/OS-III逐步进入市场，第2版也包含了这方面的内容。为方便读者阅读代码，本书目前配套的亮点嵌入式开发板的资源也在附录中列出，不选择开发板的同样可以下载代码。另外以技术论坛作为交流平台或翻转课堂，论坛地址在序言最后给出。

内容划分

第1章是操作系统和嵌入式实时操作系统的基本原理。第2章是操作系统最核心的任务管理，需要对数据结构和源代码仔细体会。第3章是中断和事件管理。第4章、第5章是事件和消息部分，包含了各种事件和消息机制。第6章是内存管理。第7章是移植的流程分析和在虚拟平台及STM32下的移植。第8章是全新的工程实践部分，给出一个在STM32下的完整的工程示例。第9章是与时俱进的μC/OS-III，并将工程实践的代码在μC/OS-III上实现了一遍。

本书特色

- 采用逐步深入，反复印证的方法。
- 采用从数据结构的设计入手，再到代码分析、示例验证的剖析方法。给出在虚拟平台下的移植示例和针对各章内容的示例。

- 给出在实际嵌入式系统下的工程示例。
- 表格、图形化的风格。
- 适用面广，适合于广大IT类学生及工作者。
- 对于没有学习过操作系统原理的读者无障碍。
- 与时俱进地扩展到μC/OS-III。
- 学习本课程的先导知识是C语言、软件技术基础或数据结构，可以同步学习微机原理或嵌入式系统设计。另外，本人的另一本著作《基于STM32的嵌入式系统原理与设计》可以与本书交相辉映。

作为本科生等教材的建议是：第1、2、3章详细讲解，第4、5、6章的内容每章选择2～3节讲解。第7、8章的内容可作为实践部分。另外如果要上实验，则可以选择在Windows下的虚拟实验，在论坛和博客提供有实验的PPT和代码。另外，也可以选择使用亮点STM32开发板作为实验教学平台。本书在每章后提供了习题，笔者也编写了PPT，适合32～48学时对高年级本科生或低年级研究生讲授。同时欢迎广大技术人员引为学习资料，欢迎进论坛和访问笔者的博客进行交流。

没有资源只有一本书不能成为平台，亮点嵌入式就是这么一个平台，本书就是核心。本书相关资源地址如下：

- 亮点嵌入式技术交流论坛http://www.eeboard.com/bp。
- 笔者新浪博客http://blog.sina.com.cn/u/2630123921。
- 配套μC/OS开发板（教学实验）：http://brightpoint.taobao.com（唯一地址，非免费）。

目前可以提供的资源主要有：

- 教学课件。
- 15个学时的实验教程代码和PPT。
- 亮点STM32开发板及配套μC/OS实例代码。

感谢读者对本书的认可，欢迎读者到论坛和博客获取资料、交流及提出宝贵意见。

笔者

2014年于成都

目　录 CATALOGUE

第1章　实时操作系统基础

嵌入式实时操作系统是当今IT行业的前沿技术，无论是电子专业或计算机专业，还是其他专业的读者，只要有一定的计算机基础，通过本书的学习都可以掌握μC/OS这一流行的嵌入式实时操作系统并使用到具体项目开发中去。本书详细讲解了μC/OS 2.91并分析了最新的μC/OS 3.0，还给出了在嵌入式环境下移植和应用的实例。

在工作中，嵌入式系统的开发者不得不与包含了大量程序代码和调度策略的实时操作系统打交道。因此，本章逐步给出实时操作系统的概念，帮助读者以最快的方式学习这一部分。如果读者对这些概念很熟悉，可以跳过。如果在阅读的过程中感觉这一部分太抽象，也没有问题，可以快速读过。学习是一个循序渐进、反复认证的过程，通过第1章的学习为后面章节打一个基础。在学习完后面章节的内容后，第1章的内容也就融会贯通了。

实时操作系统一般用于嵌入式的开发平台，如STM32、ARM、DSP、基于软核的FPGA，甚至是51单片机。但是本着学习的目的，先在普通PC的Windows XP环境下对操作系统进行代码移植，通过VC++ 对整个操作系统工程进行编译，编译后的可执行代码能在Windows平台下仿真运行，并可编译成可调试的代码以便单步调试和跟踪，这对于研究操作系统的代码、编写和验证例子程序、加深学习效果有极大的帮助。因此本书的前几章所给出的例子都是在这个环境下完成的，并试验通过。通过本书前言提供的网站可以下载到所有的源代码。

另外，嵌入式操作系统最终是要在嵌入式环境下使用的，因此笔者在实践部分给出了STM32下的移植和代码，更多的代码可以在本书前言中提到的论坛获得。

1.1　操作系统概述

本节主要介绍操作系统的一些最基本的概念。

1.1.1　什么是操作系统

操作系统（Operating System，OS）是裸机上的第一层软件。操作系统是计算机系统中最重要的系统软件，是硬件的第一层封装与抽象，在计算机系统中占据着重要的地位，其他所有的系统软件与应用软件都依赖于操作系统的支持与服务。除提供编程接口外，操作系统还承担着任务管理、事件管理和消息通信、CPU管理、内存管理、I/O管理等核心功能。

如图1.1所示，在有操作系统的系统中，应用程序（Application）并不直接和硬件打交道，而是通过操作系统和硬件打交道。操作系统直接运行在硬件平台之上，是裸机上的第一层软件，提供给用户编程接口。应用程序编程接口（API）可以解释为是一些预先定义的函数，开发者通过调用这些函数可以实现比较复杂的功能。例如，在μC/OS操作系统下，要做延时，只需要调用函数OSTimeDly即可，这个函数就是一个API。

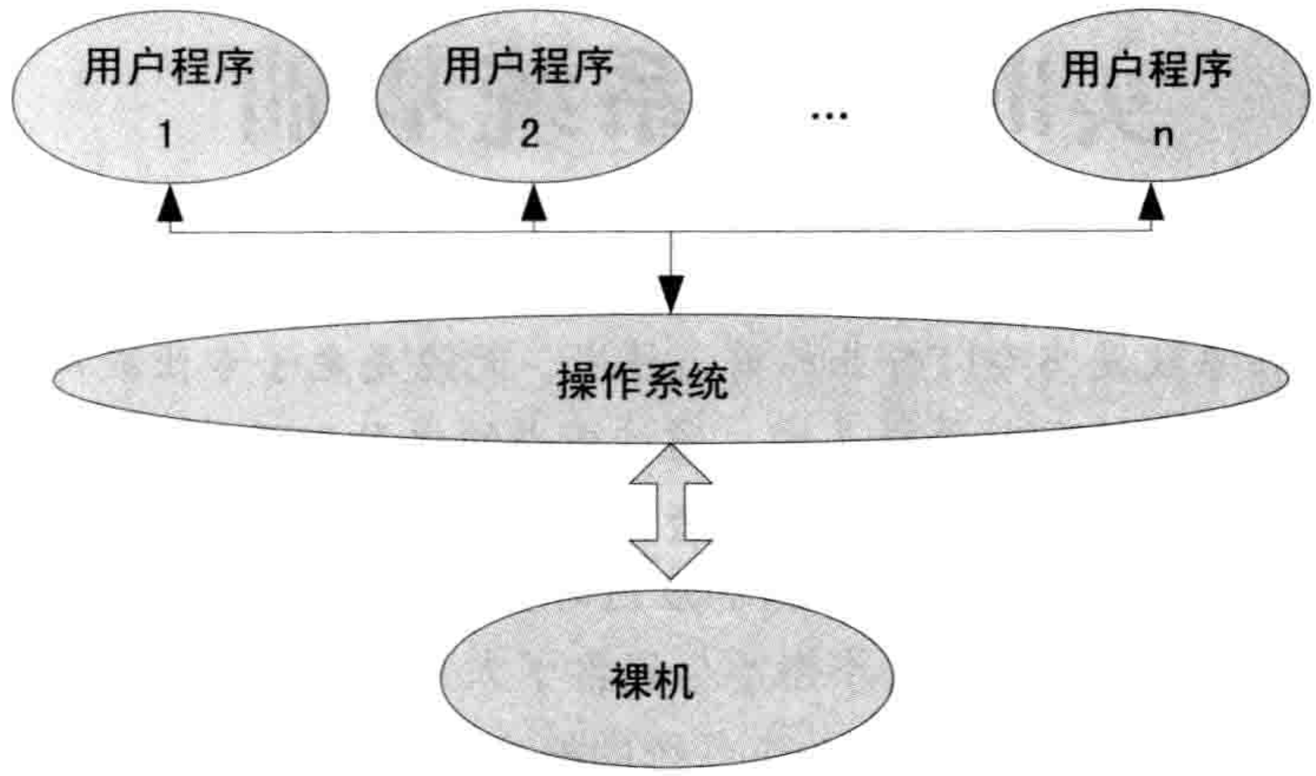

图1.1 操作系统的位置

因此，从开发人员的角度讲，操作系统提供了应用程序的接口API，我们通过调用该API来管理任务，进行消息通信及访问硬件，如打印机、显示器、硬盘等。访问硬件不仅包括写操作，即给硬件发送指令、发送数据等，也包含读操作，例如，读取打印机是否空闲，如空闲才能发送进一步的指令等。

如程序1.1所示为最简单的C语言代码，其含义是通过操作系统访问硬件。

程序1.1 屏幕输出Hello World!的C语言程序

```
int main(int argc, char* argv[])
{
    printf("Hello World!\n");                                    (1)
    return 0;
}
```

代码中（1）调用printf实现屏幕打印，不需要关心打印的细节。但是如果离开了操作系统，完成屏幕打印是比较困难和复杂的事情，首先要掌握硬件，然后编写底层程序驱动硬件。所以，本例说明操作系统提供了应用程序的接口，有了操作系统，就不用为这些底层的编程烦恼了。

换一个角度，从系统的使用人员，即最终用户的角度来看，操作系统是一台虚拟机器，在其上运行和操作用户软件。例如，一个财务人员做Excel财务报表，就是在一台虚拟机器上运行Excel软件。财务人员专注的目标是Excel中的报表，其他的底层细节与他完全无关。

1.1.2 操作系统基本功能

操作系统包含如图1.2所示的基本功能，分为5个主要组成部分。

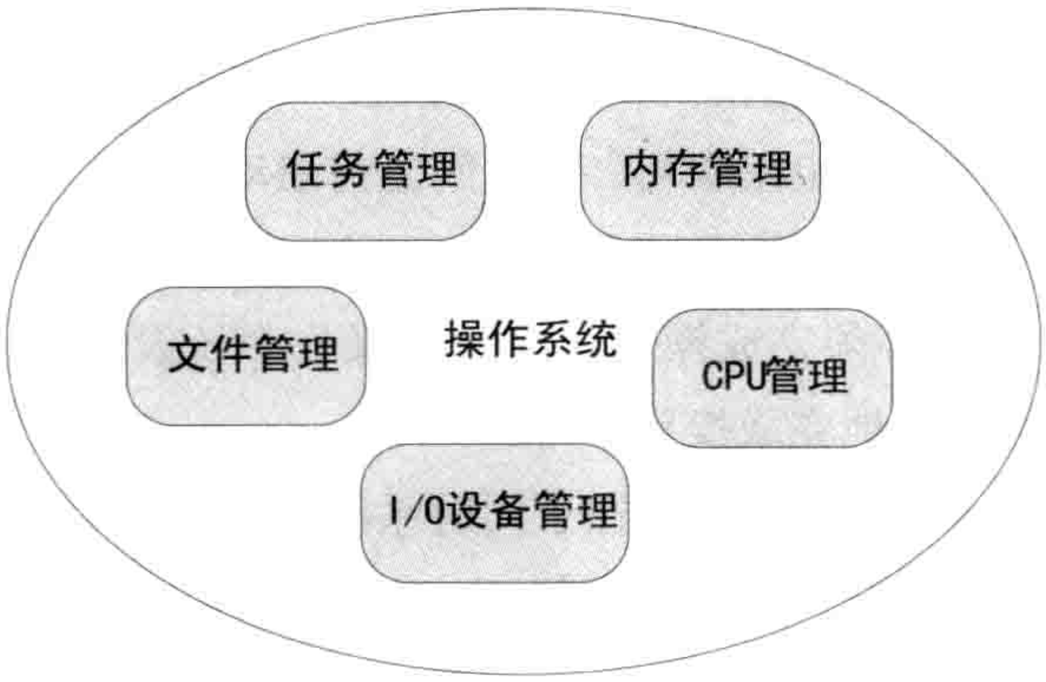

图1.2　操作系统的功能组成

1．任务管理

任务是程序的一次执行。任务可以分为系统任务和用户任务。系统任务是操作系统本身的任务，如操作系统的主程序、时钟中断服务程序，以及后面要讲到的空闲任务和统计任务等。用户任务是用户应用程序的运行，如用户设计的计算器软件的一次执行或Word软件的运行、本书中给出的一些用户任务。这些任务都需要任务管理部分来管理。

2．CPU管理

CPU管理的含义在于多任务OS对CPU的分配，也就是分配对CPU的所有权，即哪个软件正在运行，占有CPU。可以把CPU管理归入任务管理。

3．内存管理

内存是任务的生存空间。内存管理用于给任务分配内存空间，相应的，在任务结束后释放内存空间。

4．文件管理

文件管理实现对文件的统一管理，是对文件存储器的存储空间进行组织、分配和回收，负责文件的存储、检索、共享和保护。从用户角度来看，文件管理主要是实现“按名取存”，用户只要知道文件的文件名，就可存取文件中的信息，而无须知道这些文件究竟存放在什么地方。

5．I/O设备管理

I/O设备即管理系统中的各种硬件设备，如打印机、显示器、硬盘等。很明显，用户应用程序应该调用I/O设备管理模块提供的API来对设备进行操作，而不是直接读/写硬件。

1.2　实时操作系统概述

1.2.1　什么是实时操作系统

实时操作系统（Real Time Operating System，RTOS）是指当外界事件或数据产生时，能

够接收并以足够快的速度予以处理，其处理的结果又能在规定的时间内来控制生产过程或对处理系统做出快速响应，并控制所有实时任务协调、一致运行的操作系统。

所谓足够快，就是要使任务能在最晚启动时间之前启动，能在最晚结束时间之前完成。

因而，提供及时响应和高可靠性是其主要特点。实时操作系统有硬实时和软实时之分，硬实时要求在规定的时间内必须完成操作，这是在操作系统设计时保证的；软实时则只要按照任务的优先级，尽可能快地完成操作即可。

实时系统与非实时系统的本质区别就在于实时系统中的任务有时间限制。实时操作系统可以用于不需要实时特性的场合，反之则不行。

1.2.2 实时操作系统的基本特征

实时操作系统具有以下基本特征。

1. 实时操作系统首先是多任务操作系统

实时操作系统是一个多任务的操作系统，即在多任务的基础上，任务的调度时间固定、中断的响应及时、能在规定的时间内完成操作的满足实时性要求的操作系统。所谓多任务，是指允许系统中多个任务同时运行，而CPU只有一个，在某一个时刻，只有一个任务占有CPU。因此，多任务操作系统的核心任务之一就是任务调度，为任务分配CPU时间。任务调度就是微核的实时操作系统μC/OS的最核心的功能。另外，实时两个字完全不像读起来那么简单，要做到实时性，中断服务程序的长度就要务必短，因为长的中断服务程序会使低优先级的中断得不到响应。因此，实时操作系统μC/OS是利用了信号量等事件处理机制让系统迅速离开中断服务程序而巧妙地进入本来已经失去CPU的任务，相应的处理任务会完成处理的。例如，串口中断发生，串口中断服务程序发信号量给串口数据处理任务，然后就迅速离开中断服务程序以保证系统的实时性。而这个发信号量会使等待这个信号量的串口数据处理任务就绪，在离开中断时进行的任务切换过程中，这个处理任务由于优先级高会优先运行，待其处理完串口数据会继续等待信号量，原来被中断打断的任务才会继续运行。

2. 多级中断机制

一个实时系统通常需要处理多种外部信息或事件，如串行通信、网络通信或者事件报警，例如温度超高。但处理的紧迫程度有轻重缓急之分，很明显，温度超高的报警事件是最急切的，必须立即做出响应，而通信可以延后处理，并不会使整个系统出现问题。因此，建立多级中断嵌套处理机制，以确保对紧迫程度较高的实时事件进行及时响应和处理是实时操作系统必须具备的功能。对应来看，如果给串行通信、网络通信或者事件报警都创建一个处理任务，那么无疑报警事件的处理任务应该优先级最高。

3. 优先级调度机制

为做到实时，任务必须分优先级，也就是越急迫的任务优先级越高，一般短的任务也尽量优先级提高。任务管理模块必须能根据优先级调度任务，而又能保证任务在切换的过程中不被破坏。通过该机制，操作系统应能保证优先级高的任务更多地获得CPU，而优先级

较低的任务也不至于因为得不到运行而被“饿死”。因此，在编程的时候，当任务无事可做的时候一定要延时阻塞，如果需要等待某些事件的发生一定要等待（PEND）信号量等事件！这些操作都会让任务放弃CPU,低优先级的任务就有运行的机会了！换而言之，如果设计的系统出现低优先级的任务没有机会运行的局面，那不能怪操作系统，是属于自己设计上的问题，对操作系统没有吃透是主要原因。

1.3　任务

1.3.1　任务简介

任务是程序的动态表现，在操作系统中体现为线程，是程序的一次执行过程。程序是静止的，存在于ROM、硬盘等设备中。任务是运动的，存在于内存或闪存中，有睡眠、就绪、运行、阻塞、挂起等多种状态。相同程序的多次执行是允许的，这样就形成了多个优先级不同的任务，每一个都是独立的。

在实时系统中，把应用程序的设计过程分割为多个任务，每个任务都有自己的优先级，在操作系统的调度下协调运行。

典型的任务运行方式是循环。

程序1.2所列出的代码是一个典型的任务代码usertask，该任务就是一个循环结构。

程序1.2　一个典型的任务代码usertask

```
void usertask(void *pParam)
{
int i=*((int *)pParam);
        for(;;){                                                    (1)
          printf("\n\r%d\n",i++);                                   (2)
          OSTimeDly(OS_TICKS_PER_SEC);                              (3)
        }
}
```

在程序1.2中，函数usertask在（1）处进入无限循环，在循环体（2）处在屏幕上打印输出当前的计数值i，在（3）处调用操作系统的延时函数，延时1秒。1秒后又可获得运行，循环继续。这就是一个标准的任务结构。

我们对代码进行一下分析。usertask有一个参数是指针类型的，名为pParam。首先将指针强制转换为指向整数类型的指针，然后取该指针所指的内容，将它赋值给一个局部变量i。在循环体内，首先打印出i的值，然后将i加1，接着调用延时函数OSTimeDly，延时1秒，之后又继续循环。OSTimeDly首先将本任务阻塞掉，这样，即使该任务优先级最高，也可以让低优先级的任务获得运行的机会。实际上，操作系统的时钟滴答服务会在1秒后使这个任务重新就绪，重新得到运行。

在μC/OS中，一般的任务都是以这种循环方式运行的，非循环的任务是不允许返回的，而是采取删除自己的方式结束。

1.3.2 多任务

实时操作系统是多任务的操作系统，系统中必然有多个任务在执行。其中有用户任务，如前面讲到的usertask，也有操作系统的系统任务，如空闲任务和统计任务。多任务的运行相对于其他的系统，其优点是可以大大提高CPU的利用率，又必然使应用程序分成多个程序模块，实现模块化，应用程序更易于设计和维护。在一个ARM采集处理系统中，同时采集16路信号，又同时对多信号进行处理和传输，可以创建16个任务，负责16路信号的采集，创建一个任务对信号进行处理，再创建一个任务负责数据的传输。

为了使读者更好地理解多任务，下面给出简单的示例性代码，如程序1.3所示。

程序1.3 启动两个用户任务，任务代码都为usertask

```
int main(int argc, char **argv)
{
    int p[2];
    p[0]=0;
    p[1]=100;
    OSInit();                                                          (1)
    OSTaskCreate(TaskStart,0,&TaskStk[0][TASK_STK_SIZE-1],TaskStart_Prio);   (2)
    OSTaskCreate(usertask, p, &TaskStk[2][TASK_STK_SIZE-1], 5);        (3)
    OSTaskCreate(usertask, p+1, &TaskStk[3][TASK_STK_SIZE-1], 6);      (4)
    OSStart();                                                         (5)
    return 0;
}
```

代码中其他部分现在暂时不做深入研究，在相关章节还要详细论述，这里仅简单介绍以下几个方面：

（1）对μC/OS-II内核进行初始化。

（2）创建一个设置时钟中断的任务。

（3）创建用户任务，这个任务代码是前面已经提到的程序1.2中的usertask，参数是p，即指向p[0]的指针，优先级是5。

（4）创建用户任务，这个任务代码也是usertask，参数是p+1，即指向p[1]的指针，优先级是6。

（5）开始启动多任务。

因为两个用户任务虽然优先级不同，但是都执行延时操作，也就是说都主动放弃CPU，因此可以轮流运行。如果没有执行延时操作，把自己阻塞起来，在高优先级任务结束前，低优先级的任务是不能得到运行的。

如图1.3所示为程序清单1.3运行的结果。

图1.3　两个任务运行的结果

第一个任务的参数是0，从0开始计数；第二个任务的参数是100，从100开始计数。于是得到如图1.3所示的结果。我们看到，两个任务轮流输出结果到屏幕，多任务被调度运行了。

1.3.3　任务状态

任务是活的，有多种状态，如图1.4所示是任务状态图，说明任务的状态和它们之间的关系。

```
task1 call add2(1,2) solution is 300
task2 call add2(100,200) solution is 3
task1 call add2(1,2) solution is 300
task2 call add2(100,200) solution is 3
```

图1.4　任务状态及相互关系

为了管理和调度任务，必须为任务设置多个状态。在μC/OS中，任务具有5种状态。

1．睡眠态

任务已经被装入内存了，可是并没有准备好运行。例如，上面给出的usertask代码，以代码的形式存在于内存中，在调用OSTaskCreate（任务创建函数）创建之前，处于睡眠态。睡眠态的任务是不会得到运行的，操作系统也不会给其设置为运行而准备的数据结构。经过后面的学习我们将知道，没有给其配置任务控制块。

2．就绪态

当操作系统调用OSTaskCreate创建一个任务后，任务就进入就绪态。从图1.4中可以看出，任务也可以从其他状态转到就绪态。处于就绪态的任务，操作系统已经为其运行配置好了任务控制块等数据结构，当没有比其优先级更高的任务，或比其优先级更高的任务处于阻塞态的时候，就能被操作系统调度而进入运行态。从就绪态到运行态，操作系统是调用任务切换函数完成的。

3．运行态

运行态是任务真正占有CPU，得到运行。这时运行的代码就是任务的代码，如usertask。处于运行态的任务如果运行完成，就会转为睡眠态。如果有更高优先级的任务抢占了CPU，就会转到就绪态。如果因为等待某一事件，例如等待1秒的时间，如OSTimeDly

(OS_TICKS_PER_SEC)，需要暂时放弃CPU的使用权而让其他任务得以运行，就进入了阻塞态。当由于中断的到来而使CPU进入中断服务程序（ISR），必然使正在运行的任务放弃CPU而转入中断服务程序，这时被中断的程序就被挂起而进入挂起态。

总之，任务要得到运行就必须进入运行态，CPU只有一个，不能让每个任务同时进入运行态，进入运行态的任务有且只有一个。

4. 阻塞态

阻塞对于操作系统的调度、任务的协调运行是非常重要的。我们之所以能看到图1.3所示的运行结果，而不是只有一个高优先级的任务得到运行，就是因为usertask在没有事情可做，等待1秒的时候，不是强行运行代码，而是把自己阻塞起来，使操作系统可以调度其他的任务。

当任务在等待某些还没有被释放的资源或等待一定的时间的时候，要阻塞起来，等到条件满足的时候再重新回到就绪态，又能被操作系统调度以进入运行态，这是实时操作系统必须要实现的功能之一。

一些不理解操作系统的读者编程时，在等待的时候常常使用for循环，不停地执行代码而使CPU的利用率暴增，使系统的运行环境十分恶劣，甚至造成死机，这是不可取的。

5. 挂起态

当任务在运行时，因为中断的发生，如定时器中断每个时钟滴答（clock tick，指每个时钟周期）中断一次，被剥夺CPU的使用权而进入挂起态。在中断返回的时候，若该任务还是最高优先级的，则恢复运行，如果不是这样，只能回到就绪态。

任务在各种状态之间转换，有一个非常重要的词——Context Switch，即上下文切换或直接翻译为任务切换。

1.3.4 任务切换

从上一节的例子中读者可以看到多任务运行的结果，很明显，任务进行了切换，否则不可能产生如图1.3所示的运行结果。

任务切换的核心是上下文切换（Context Switch），是任务调度的重要部分。任务切换是暂停一个任务的运行，运行另一个处于就绪态的任务。暂停一个任务，以后又能恢复运行，必须考虑将这个任务运行的信息保存，而恢复运行的时候需要将这些信息恢复到运行环境。这种保存上下文的数据结构就是堆栈，是一种后进先出的数据结构。

于是，任务切换必须做环境的保存和恢复的操作。环境的保存和恢复与任务有关，也与任务运行的硬件环境直接相关。PC和ARM就有不同的CPU寄存器，很明显，其保存和恢复的内容就大不相同。所以，要实现内存切换就涉及了汇编语言实现的最底层的代码，需要一定的计算机原理或嵌入式系统的基本知识，即硬件的基本知识和汇编语言编程的基本知识。掌握最基本的知识，对于我们的操作系统学习就足够了。

在操作系统移植的时候，任务切换代码就是必须要实现的部分之一。

多任务的关键在于如何进行调度，μC/OS采用的是可剥夺优先级调度算法。多任务下，各任务还要按一定的次序运行，因此存在同步的问题，所以引入了信号量的概念来进行同步。任务间有互相通信的需求，因此操作系统需要有邮箱、消息等用于通信的数据结构，以便多任务通信。多个任务可能争夺有限的资源，如都要访问串口，因此操作系统还要管理各任务，尽可能使它们和平共处，能充分利用资源而不发生冲突，该排队的就要排队，于是又产生了互斥、死锁等概念。

1.3.5　可重入函数和不可重入函数

同一个代码可以运行为不同的任务，不同的任务又可以调用相同的函数。我们不能为每个任务都写一个程序，却需要每个程序都运行正确，无论系统中有多少个任务在执行。

可重入函数（Reentarut Function）是指一个函数可以被多个任务调用，而不需要担心在任务切换的过程中，代码的执行会产生错误的结果。如果可能产生错误的结果，就是不可重入函数了。在实时多任务操作系统中，任务应该调用可重入函数，可重入函数是任务中的一个重要概念。

如程序1.4所示的例子：启动两个用户任务，任务代码为usertask1和usertask2，两个任务都调用add2函数，该函数将返回两个参数相加的结果。其中使用了全局变量a和b。

程序1.4　使用不可重入函数得到错误结果

```
int a,b;                                                    (1)
int add2(int p1,int p2)                                     (2)
{
    a=p1;
    b=p2;
    OSTimeDly(OS_TICKS_PER_SEC);                            (3)
return(a+b);                                                (4)
}
void usertask1(void *pParam)
{
     int sum;
    for(;;){
            printf("\ntask%d call add2(1,2)\n",1);
            sum=add2(1,2);
            printf("\ntask%d call add2(1,2) solution is %d\n",1,sum);
    }
}
void usertask2(void *pParam)
{
     int sum;
    for(;;){
            printf("\n\rtask%d call add2(100,200)\n",2);
            sum=add2(100,200);
            printf("\ntask%d call add2(100,200) solution is %d\n",2,sum);
    }
}
```