摘 要

随着后PC时代的到来，嵌入式系统的设计已经成为现代计算机应用设计的一大领域和方向，在自动控制、汽车、手持设备、航空航天、武器装备等领域以及生活中的各方面，嵌入式系统都有着非常广泛的应用。为了应对越来越多样化和复杂化的应用，在嵌入式系统中使用嵌入式操作系统已经成为未来嵌入式系统发展的一个方向，嵌入式操作系统的优劣关系到整个嵌入式系统的性能。

本论文首先回顾了嵌入式系统的发展历史，介绍了嵌入式操作系统的基本原理，比较了国内外嵌入式操作系统的发展和现状，然后对嵌入式操作系统特别是微内核结构的嵌入式操作系统作了深入的研究。通过对嵌入式操作系统源代码的分析，如 small rtos51、UC/OSII、uCLinux 等，吸取和借鉴了这些操作系统的设计思想和解决方案，设计了一个嵌入式操作系统微内核，该内核由任务管理、任务通信、时钟管理、中断管理、内存管理等模块组成。最后介绍了 ARM 处理器及 EL-ARM-830 硬件平台，给出了内核在平台上的实现。内核实现了多任务的调度、任务间的通信和同步等基本的微内核功能，通过对互斥信号量的改进，很好的解决了抢占式嵌入式操作系统中普遍存在的优先级反转问题。

关键字：嵌入式操作系统；微内核；ARM；任务调度

Abstract

As the post-PC era has come, the embedded system design has become a major field of modern computer application and the direction of modern computer development. Embedded system was used in a lot of domain, such as automation field, automobile field, portable equipment, aerospace, weapons equipment as well as other aspects in life, so the embedded systems have a good prospect of application. In order to adapt the increasing diversity and complexity of application , using embedded operating system in the embedded system has become a direction for the future development of embedded systems. The embedded operating system take an important role in the embedded system, its performance determines the embedded system's performance.

In this paper, first reviewed the history of the development of embedded system, and introduce the basic principle of embedded operation system, compared the development and actuality in our country and oversea. then make a Thorough research on the embedded operating system especial the embedded operating that use micro kernel structure.. By analysing the source code of embedded operating system, such as small rtos51, uC/OS II, uClinux and so on, draw the design ideas and solutions of these operating system, and design a micro kernel operating system. The system kernel consist of task management module, task communication module, clock management module, interrupt management module and memory management module and so on. At last introduced the ARM processor and EL-ARM-830 hardware platform, and the implemention of the system on this platform. The system implements the functions of a micro kernel, such as multi-task schedule function, task communication function, task Synchronization function and so on. By making implovements to the semaphor used in the system, it solves the priority inversion problem that was very common in the preemptive system which was based on priority.

Key words: embedded operating system; micro kernel; ARM; task schedule

研究生学位论文独创性声明和版权使用授权说明

独创性声明

本人声明：所呈交的论文是我个人在程小辉教授指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含他人已经发表或撰写过的研究成果，也不包含为获得桂林工学院或其它教育机构的学位或证书而使用过的材料。对论文的完成提供过帮助的有关人员已在论文中作了明确的说明并致以了谢意。

学位论文作者（签字）：

签字日期：2007.6.11

版权使用授权说明

本人完全了解桂林工学院关于收集、保存、使用学位论文的规定，即：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务；学校可以采用影印、缩印、数字化或其它复制手段保存论文；在不以赢利为目的前提下，学校可以公布论文的部分或全部内容。(保密论文在解密后遵守此规定)

学位论文作者（签字）：

指导教师签字：

签字日期：2007.6.11

第1章 绪论

1.1 嵌入式系统的概述

嵌入式系统是以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统，如手机、PDA、车载GPS、洗衣机、冰箱等。

早在20世纪60年代就开始有嵌入式系统了，但那时的系统设计得比较简单，基本上就是一些简单的监控程序，而且几乎整个系统都是采用汇编语言来编写，这样，系统的移植性、可读性非常差，当更换处理器时不得不重新改写整个系统。随着计算机技术的发展，后来出现了高级语言，如C语言，这样，软件开发者就可以用高级语言来进行程序设计，从而大大的提高了系统的开发速度，程序的可读性和移植性也都得到了很大的提高，遗憾的是这些程序往往都是针对特定应用的专用程序，这些程序运行时也没有用到真正意义上的操作系统来进行调度，而是采用前/后台系统的设计方式。在前/后台方式的系统中整个应用程序就是一个超循环，在循环中调用相应的函数来完成相应的操作，这部分可以看作是后台行为，然后用中断服务程序来处理异步事件，这部分可以看作是前台行为。后台也称之为任务级，前台也称之为中断级，那些时间相关性强的工作一般由中断服务程序来处理。但是，在前/后台方式的系统中，中断服务程序提供的信息必须在等到后台程序运行到该处理这个信息这一步时，才会调用相应的函数对这一事件进行处理，因此这种系统的响应时间长，而且与系统的整个循环有关。当程序修改后，响应时间也会受到影响，因此响应时间具有不确定性。对于早期那些功能简单、要控制的对象少、硬件资源少的应用来说前/后台方式已经能满足要求，但随着电子技术的发展，处理器的处理能力和单位面积上能集成的存储器容量都得到了很大的提高，用户需要处理的任务和控制的对象也变得越来越复杂，传统的嵌入式应用系统设计方式逐渐难以满足这样的要求，因为采用传统嵌入式系统设计方式开发出来的软件很难维护。嵌入式领域的应用种类繁多，为了适应这种复杂性和多样性特点，缩短产品的开发周期，这就需要彻底的改变传统嵌入式系统的设计方式，在设计中引入操作系统。通过操作系统来有效的管理越来越复杂的系统资源，更合理的利用CPU资源，从而也能简化应用程序的设计，更好的保证系统的实时性和可靠性。操作系统通过把硬件虚拟化，使得开发人员能够从繁忙的驱动程序移植和维护中解脱出来，系统以提供库函数、标准设备驱动程序以及开发工具的形式来提高软件的复用性、系统的实时性等。

概括的说，嵌入式软件系统的发展主要经过了如下的四个发展阶段：

（1）无操作系统阶段：以单芯片为核心的可编程控制器形式的系统，同时具有与监测、伺服、指示设备相配合的功能，如工业控制系统等，一般没有操作系统的支持，通过汇编语言编程对系统进行直接控制，运行结束后清除内存。这一阶段系统的主要特点是：系统结构和功能都相对单一，处理效率较低，存储器容量较小，几乎没有用户接口。由于这种嵌入式系统使用简便、价格低廉，以前在国内工业领域应用较为普遍，但是已经远远不能适应高效的、需要大容量存储介质的现代化工业控制和新兴的信息家电等领域的需求。

（2）简单监控式的实时操作系统阶段：以嵌入式 CPU 为基础、以专用简单操作系统为核心的嵌入式系统[9]。这一阶段的嵌入式系统 CPU 种类繁多，通用性比较弱、系统开销小，效率高。操作系统具有一定的兼容性和扩展性，应用软件较专业，用户界面不够友好，系统主要用来控制系统负载以及监控应用程序运行。

（3）通用的嵌入式实时操作系统阶段：以通用的嵌入式操作系统为标志的嵌入式系统，如 VxWorks、PSOS、Windows CE 等。这一阶段系统的主要特点是：能运行在各种不同的微处理器上；具有强大的通用操作系统的功能，如具备了文件和目录管理、多任务管理、设备驱动支持、网络支持、图形窗口以及用户界面等功能；具有丰富的 API 和嵌入式应用软件。

（4）以 Internet 为标志的嵌入式系统：随着通用型嵌入式实时操作系统的发展，面向 Internet 网络和特定应用的嵌入式操作系统正日益引起人们的重视，成为重要的发展方向。嵌入式系统与 Internet 的真正结合、嵌入式操作系统与应用设备的无缝结合代表着嵌入式操作系统发展的未来。

1.2 嵌入式系统的主要特点

嵌入式系统主要由嵌入式处理器、相关支撑硬件和嵌入式软件系统构成，是集软硬件于一体的可独立工作的系统。嵌入式处理器可以是一个单片机或一个微控制器(MCU)，随着电子技术的发展，一些高性能 32 位处理器，如 ARM 处理器等也正被逐步的应用到嵌入式领域中。相关支撑硬件包括显示卡、存储介质(ROM 和 RAM 等)、通讯设备、IC 卡或信用卡的读取设备等。嵌入式软件包括与硬件相关的底层驱动、操作系统、图形界面、通讯协议、数据库系统、标准化浏览器和应用软件等。

嵌入式系统应用前景广阔，可应用于人类工作与生活的各个领域，如智能工控设备、IC 卡、机顶盒、数字电视、WebTV 等众多消费类和医疗保健类电子设备。此外，还有各类的多媒体手机、袖珍电脑、掌上电脑、车载导航器等。总之，在生活里随处都能看到嵌入式系统的影子，嵌入式系统已经深入到我们生活的方方面面，人们把嵌入式设备

被广泛使用的时代称为“后 PC 时代”。

嵌入式系统诞生于微型机时代，其核心硬件——处理器的发展也经历了三个阶段：SCM 阶段、MCU 阶段、SOC 阶段。

SCM（Single Chip Microcomputer）即单片微型计算机阶段，主要是寻求最佳的单片形态嵌入式系统的最佳体系结构，它与通用计算机朝着完全不同的道路发展。

MCU（Micro Controller Unit）即微控制器阶段，通过不断扩展来满足嵌入式对象系统要求的各种外围电路与接口电路的应用，加强对对象的智能化控制能力。在发展 MCU 方面，最著名的厂家当数 Philips 公司，它将 MCS-51 从单片微型计算机迅速发展到了微控制器。

SOC（System on chip）即片上系统阶段，随着半导体工艺技术的发展，IC 设计者能够将愈来愈复杂的功能集成到单硅片上，SoC 正是在集成电路（IC）向集成系统（IS）转变的大方向下产生的。单片机是嵌入式系统的独立发展之路，向 MCU 阶段发展的重要因素，就是寻求应用系统在芯片上的最大化解决，而专用单片机的发展自然形成了 SoC 化趋势。

嵌入式系统是基于计算机技术的，但从前面对嵌入式系统的定义可以知道，嵌入式系统与通用的计算机系统又是不同的，它与通用计算机相比具有以下的几个特点：

（1）嵌入式系统与通用计算机系统最大的不同是它面向特定的应用对象，为特定用户群设计的，针对性强，具有低功耗、体积小、集成度高等特点，能够把通用 CPU 结构的系统中许多由板卡来完成的功能集成在芯片内部，从而有利于嵌入式系统设计趋于小型化，移动能力大大增强。

（2）嵌入式系统是将先进的计算机技术、半导体技术以及电子技术与各个行业的具体应用相结合后的产物，这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

（3）嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用中对处理器的选择更具有竞争力。

（4）为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，它的升级换代也是和具体产品同步进行，因此嵌入式系统产品一旦进入市场，具有较长的生命周期。

（5）嵌入式系统本身不具备自举开发能力，产品的开发需要借助其它的平台来进行开发，如采用宿主——目标机的方式等，通过宿主机生成目标机能运行的代码，然后再

把它固化到目标机中，这样产品在销售以后用户通常也是不能对其中的程序功能进行修改的。

1.3 嵌入式操作系统

作为嵌入式系统灵魂的嵌入式操作系统是随着嵌入式系统的发展而出现的，是嵌入式系统发展到一定阶段的产物。为了面对越来越复杂的应用，提高系统的可靠性、提高产品的开发效率和缩短产品的开发周期，越来越多的嵌入式产品厂家都在其产品中加入了操作系统。嵌入式操作系统的使用能为嵌入式系统的开发减少工作量，增强嵌入式系统的可移植性和适应性。在一些控制系统中，出于安全方面的考虑，要求系统不能崩溃，当出现小错误的时候要有自愈能力，这不仅要求在硬件设计方面提高系统的可靠性和抗干扰性，而且也应在软件设计方面提高系统的抗干扰性，尽可能地减少安全漏洞和不可靠的隐患，这往往需要使用嵌入式操作系统。那些采用前/后方式设计的嵌入式系统在遇到强干扰时，容易产生异常、出错、跑飞，甚至死循环，造成了系统的崩溃。如果有操作系统，则可以通过操作系统运行的系统监控进程对其进行修复，采取一些利于系统稳定、可靠的措施，如把有问题的任务清除掉，使系统重新恢复正常。而且在有嵌入式操作系统支持的环境下开发应用程序，能够把一个复杂的应用程序，按照软件工程中的解耦原则将整个应用分解为多个任务模块，这样，每个任务模块的调试、修改几乎都不影响其他模块，开发人员就可以对几个模块同时进行开发，提高产品开发的速度。

与 windows 和桌面 Linux 等 PC 操作系统不同，嵌入式操作系统既有普通操作系统的特点，如负责整个系统的软件和硬件资源的分配、对任务进行调度、管理外设等，同时它运行的环境不同于台式机，如没有很大的硬盘，通常只有少数的 Flash、Eprom 等来存放程序和数据，系统的各种应用程序和功能都是事先确定的等，使得嵌入式操作系统有着自身的特点：①小巧和可定制性。由于嵌入式系统上硬件资源的限制，因此嵌入式操作系统必须设计得小巧紧凑，同时要能够满足不同用户的要求，能够根据不同的用户进行定制，去掉用户不需要的功能模块，以节省资源。② 实时性。通用操作系统追求的是软硬件资源的最大利用率，而嵌入式操作系统则不同，嵌入式系统多是对一些设备进行监测和控制，如，航天飞机的方向调节系统、导弹导航系统、核反应堆控制系统等，对这些设备的控制不仅要求程序逻辑上正确，而且在时间上也必须满足要求，如果不能在系统规定的时限内完成任务，就会造成严重的后果，这要求嵌入式系统有实时性的功能。③能够固化。在嵌入式系统里，应用软件和操作系统都是被固化在 Eprom 或 Flash 等存储器里面的，而像硬盘这样的辅助存储器基本上不用。④稳定性和弱交互性[2]。很多嵌入式系统都用在一些性命攸关的设备上，因此其稳定性是至关重要的，不允许系统在运行过程中出现什么故障。同时，嵌入式系统运行起来后也不再需要人进行干预，因

此不需要很强的交互性。

在嵌入式操作系统领域，到了20世纪80年代呈现出一种百家争鸣的局面，大量的商用系统相继问世，如pSOS，VxWorks， Palm OS，Windows CE，QNX等，它们作为商用的操作系统具有良好的性能。

PSOS：是美国加州的集成系统公司（Integrated Systems Inc）推出的实时操作系统，它专门为嵌入式微处理器而设计，能提供多任务处理环境，稳定性好，而且使用方便，模块化的体系结构使得系统具有很高的可扩展性，可以根据不同的应用需求来定制操作系统的功能和需要的内存。整个系统分为内核层、系统服务层、用户层，由内核层负责任务的管理和调度、任务间的通信、内存管理、实钟管理、中断服务等。系统服务层包括PNA+、PRPC+、PHILE+等组件，PNA+实现了完整的基于流的TCP/IP协议集，PRPC+提供了远程调用库,支持用户建立一个分布式应用系统，PHILE+提供了文件系统管理和对块存储设备的管理。用户层指的是用户编写的应用程序，它们以任务的形式出现。PSOS在手机等设备中应用比较广泛。

Palm OS：是由3Com公司开发的嵌入式操作系统，在PDA市场上占有很大的市场份额，它提供了串行通信接口和红外线传输接口，可以方便地与其它外部设备通信、传输数据，利用Palm OS提供的应用程序接口，开发商可根据需要自行开发所需的应用程序，目前大约有数千种专为Palm OS编写的应用程序，小到游戏，大到行业解决方案，Palm OS无所不包。

Windows CE：是Microsoft公司开发的一个开放的、可升级的32位嵌入式操作系统。它是从整体上为有限资源的平台设计的多线程、基于优先权、多任务的操作系统，其模块化设计使得它可方便的进行移植，该操作系统的基本内核需要至少 200K 的 ROM。Windows CE具有模块化及可伸缩性，实时性能好，通信能力强大，支持多种CPU，它的设计可以满足多种设备的需要，包括工业控制器、通信集线器以及销售终端之类的企业设备，还有照相机、电话和家用娱乐器材之类的消费产品。

1.4 课题背景

传统嵌入式系统的开发，通常都是针对某一应用，画程序流程图，然后编制应用程序，一般称这种程序为线性程序，采用这种方式来开发系统，软件和硬件的调试几乎占了系统开发的大部分时间，当程序产生错误或系统受到干扰时，只能通过看门狗复位系统，这严重的影响了系统的稳定性和程序的可读性。随着系统资源的增加和应用程序越来越复杂，采用线性程序设计的方法难以满足应用的要求。而在产品的开发中采用支持多任务的嵌入式操作系统，能够为开发提供一个稳定的平台，从而使得开发者能把精力集中到提高系统的性能上。基于操作系统的编程能使应用程序更便于移植，提高软件的

可复用性。

微内核结构的操作系统是相对于传统的宏内核结构即单内核结构来说的，采用宏内核结构设计的操作系统，整个系统是一个运行在核心态的大进程，包括了进程管理、内存管理、文件系统以及其它的功能，模块间的通信通过直接调用其它模块中的函数来实现，而不是消息传递。微内核的设计思想是只把系统最关键的功能，如进程调度、时钟管理、中断管理等在内核中实现，其它的功能如 I/O 管理、文件管理等放到内核之外，作为独立的服务进程来实现。进程间通过消息来进行通信，而内核就是一个消息的转发站，比如，系统调用模块要给文件系统管理模块发消息时，消息就通过内核来进行转发。采用这种结构设计的内核，能够实现模块间的隔离，在不影响其它模块的情况下可以方便的用一个新的实现来替代原来的模块，这样整个系统方便灵活，又能保证内核微小，便于移植。

1.5 论文所做的工作及论文的结构

本文首先介绍了嵌入式系统以及嵌入式操作系统的历史、现状和发展前景，分析和比较了目前比较流行并且代码公开的几款嵌入式操作系统，通过对这些系统的分析和研究，笔者对操作系统的理论及设计方法有了更深刻的理解，吸取了它们的优秀之处，运用所学的操作系统知识结合微内核思想，设计开发了一个小型的嵌入式操作系统。系统采用模块化的设计方法，分为任务管理、任务间通信、时间管理、中断管理、内存管理等多个模块，并给出了各模块的详细设计。

嵌入式操作系统的设计要求开发人员对底层的硬件有较深的了解，特别是编写与硬件相关部分的代码时更是如此。本系统在设计的时候就考虑了系统的移植性问题，为了便于移植，那些硬件无关部分的代码都采用 C 语言来编写，而与底层硬件相关部分的代码用汇编语言来编写，这样在移植的时候，只要修改或重新编写这部分代码即可。全文分为 5 章，其内容安排如下：

第 1 章 绪论。简单的介绍了嵌入式系统的发展历史，课题来源、背景和研究意义，以及论文的主要内容及章节安排。

第 2 章 微内核体系结构的研究。对微内核结构的操作系统进行了深入的分析和研究，包括任务的划分、任务的调度、任务间的通信以及优先级反转问题的处理等。

第 3 章 操作系统内核的设计。采用模块化的设计方法对操作系统的各功能模块进行了详细的设计，完成了一个微内核结构的嵌入式操作系统内核。

第 4 章 系统在 ARM 平台上的实现。最后给出了系统在 ARM 硬件平台上的实现，为系统在其它平台上的运行提供了一个参考。

第 5 章 总结与展望。对本文的工作进行了总结并展望下一步的研究工作。

第 2 章 微内核体系结构的研究

2.1 操作系统体系结构的发展

嵌入式操作系统 EOS(Embedded Operating System)是嵌入式系统中的灵魂，是为嵌入式系统特别设计的一段代码，也可称为系统的内核，它具有非常高的可靠性，负责管理系统的任务、中断、时钟、I/O、设备等各种资源，能根据不同的调度策略为各任务合理的分配 CPU 时间，通过为用户提供 API 接口的方式，方便用户应用程序的编写。基于操作系统的嵌入式软件开发，在设计某个任务的时候可不必关心其它任务的状态，任务之间的通信和同步都由操作系统来完成。优秀的操作系统通过采用各种调度算法和调度策略来提高系统的性能，保证系统行为的可预知性。

操作系统内核是操作系统最核心最基础的部分，其结构往往对操作系统的外部特性以及应用领域有着一定程度的影响。操作系统按内核的结构来分大致可以把内核分为单内核(monolithic kernel)、微内核(micro kernel)两种。单内核结构是操作系统中各核心模块（任务调度、文件管理、内存管理、I/O 管理等）杂然混居的形态，该结构产生于 1960 年代，历史最长，是操作系统内核与外围分离前的最初形态，很多的操作系统采用的都是这种结构方式，如 Unix、Linux、Windows 等。随着操作系统体系结构理论和实践的不断演进和发展，操作系统高层特性与内核结构之间的耦合度日趋缩小，逐渐把结构性部件与功能性部件分离开来，到 1980 年代就出现了微内核结构。微内核是内核的精简版本，只提供操作系统的核心功能，如，任务调度、时钟管理、中断管理、信号量管理等，把其它功能以系统服务器的方式提供给用户（如文件管理、网络支持等），用户可以根据需要对其进行裁减。采用微内核结构的操作系统占用的内存小，可移植性强，同时提供模块化的设计，能方便用户安装不同的接口，更好的满足各类用户群体的要求。传统单内核操作系统和微内核操作系统的结构分别如下图 2.1 和图 2.2 所示：

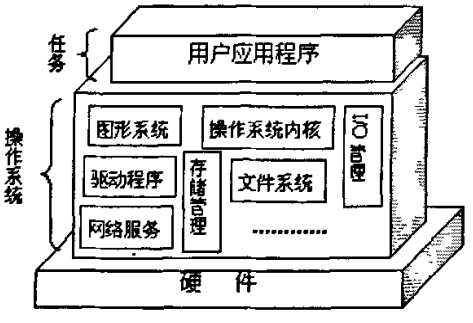


图 2.1 单内核操作系统结构

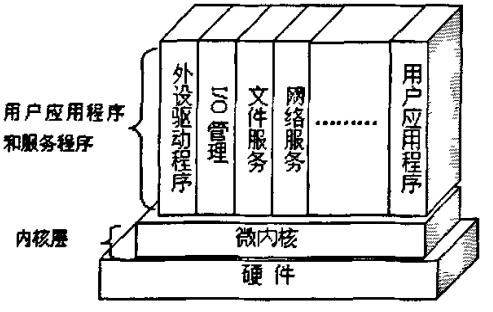


图 2.2 微内核操作系统结构

在操作系统结构的发展中，还有过一种分层结构的内核[3]，但是分层结构的内核存在很多的问题，因为在分层结构的内核中每一层提供的功能相当多，相邻层之间相互影响，如果一层的功能发生了变化那么与它相连的层将很难修改，在嵌入式系统中这种结构很少使用，而微内核结构能更好的满足嵌入式系统应用多样性，存储空间小的特点，因此越来越多的嵌入式操作系统采用的都是这种内核结构。

2.2 微内核结构的操作系统

2.2.1 微内核结构操作系统的特点

随着计算机体系结构的不断发展和变化，用户的需求也不断的扩大，操作系统必须具有很好的可移植性、可扩充性、可靠性、适应性、兼容性以及很高的效率等特点才能满足各种应用要求，而传统的单内核结构操作系统随着系统功能的增加使得整个内核变得越来越庞大、混乱，因此，后来就有人提出了微内核结构的操作系统。

微内核设计与单内核设计不同，其基本思想是将原来属于传统操作系统内核的一些功能和服务从内核中分出，以子系统的形式与内核或其它子系统相互作用，组件功能以服务进程的方式通过消息与微内核和其它组件相互通信来完成。微内核的主要功能就是消息交换：确认消息、在组件间传送消息以及对硬件的操作等。微内核通常只保留进程间通信(IPC)、进程调度、时钟管理等几项最基本、最简单的功能，而将诸如文件管理等其它传统操作系统服务作为用户层的服务进程驻留在微内核外。它依据客户-服务者模型的概念，把所有其它的操作系统功能都变成用户态的服务器，而用户进程则被当作客户。客户要用到操作系统时，其实就是通过微内核与服务器进程通信而已。微内核仅仅作为一个传递消息的工具，验证消息的有效性，在客户和服务器之间传递消息，并核准对硬件的存取。微内核结构的操作系统与单内核结构的操作系统相比具有以下特点：

(1)功能单一、体积小巧，可用于对实时性有要求的场合，同时那些与处理器相关的硬件方面的细节都包含在内核中来处理，便于系统在不同平台之间移植。

(2)具有很高的效率，系统响应快，上下文的切换速度高于一般的多任务系统，这样可以在低速的 CPU 上完成复杂的抢占式实时多任务功能。

(3)微内核采用客户-服务器的工作方式，如果要扩展系统的功能，只需要增加一个服务器即可，使得系统更易于扩展。

(4)微内核各服务器模块都是相互独立的，通信只是通过消息机制来进行，一个模块的出错不会导致整个系统的崩溃，提高了系统的稳定性。

(5)微内核系统和运行在其上的应用软件一般都固化在存储器芯片或单片机本身中，可靠

性高，而且微内核系统本身一般不具备自举开发能力，在设计完成以后用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发。

微内核的特点决定了其固有的模块化特性，而模块化反过来又为微内核操作系统提供了许多优良的特性，包括容易维护、修改方便、可移植性强、操作灵活、支持多处理器、易于实现分布式系统等功能。但微内核结构在带来操作系统高度灵活性和扩展性的同时，也使得操作系统的整体性能有所降低，因此必须对微内核结构系统的一些模块设计进行改进，使其在性能上能更好的提高。

2.2.2 微内核结构操作系统的不足及改进方法

微内核结构的操作系统较传统单内核结构操作系统更具灵活性和可扩展性，但在性能上也有所下降，因为在微内核结构中只保留了进程管理、进程调度等几项最基本的功能，其它的功能都是以内核服务进程的形式放在内核之外。这样，应用程序要获得内核的服务就必须先发消息给内核，然后内核再发消息给服务进程，执行得的结果按相反的顺序返回给调用的应用程序。基于消息机制的体系结构导致了频繁的进程切换和消息数据的拷贝，从而增加了系统的额外开销。应用程序申请内核服务的过程可用图 2.3 表示。

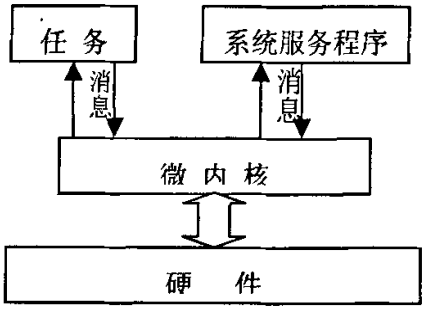


图 2.3 任务与内核服务的消息传递

采用消息的方式在内核服务器模块间以及内核与各服务器模块间进行消息传递，能使系统的内核保持微小的结构，满足嵌入式的应用，同时这种结构方式的系统其健壮性和可扩展性也是很高的，要增加系统的功能，只需增加一个新的内核服务程序模块就行了，不必修改其它的模块。同时，如果系统里某一个模块出了问题也不会影响其它模块，因此其可靠性也是很好的。但频繁的消息传递必然会增大系统的开销，因此，提高微内核结构操作系统性能的一个重要途径就是对其中的消息结构进行改进。

2.2.3 微内核消息机制的改进

单内核结构的操作系统，系统各模块之间可以相互直接调用，而微内核结构的系统

是基于消息机制的，因而系统各模块间的消息传递对系统的性能有着重要的影响，对消息机制的改进能够很好的改善系统的性能。通常可以采用下面两种方法来对其进行改进：

①减少要传递的消息的长度，这样既能提高消息传递的速度又能减少系统的开销。如果消息足够短的话还可以用寄存器来进行传递，寄存器的速度要比存储器的速度快。

②接受和发送模块直接从已为消息分配的缓冲区中对消息进行读写，而不必再为要传递的消息分配缓冲区和把消息移到新分配的缓冲区[4]。

消息机制是用于在系统的各模块间进行信息传递的，也可以用来改变系统的任务调度策略从而提高系统的实时性能。当低优先级任务在使用系统服务时，如果有比它优先级更高的任务产生，并需要使用这项服务，则高优先级任务就发一个请求给这个服务程序，服务程序就提高自己的优先级，使低优先级的任务放弃对它的请求，从而高优先级的任务就能获得它的服务。

2.3 采用微内核结构的嵌入式操作系统

2.3.1 QNX

QNX[5] [6]是 QNX 软件系统公司开发的一个实时、可扩充的操作系统，它遵循 POSIX.1(程序接口)和 POSIX.2(Shell 和工具)、部分遵循 POSIX.1b(实时扩展)。QNX 采用的是微内核结构，有地址空间保护机制，稳定可靠，被业界公认为是在 X86 平台上最好的嵌入式实时操作系统，目前已被移植到 PowerPC、MIPS、ARM 等内核的处理器上。QNX 的内核非常小，只有 8K，在内核中只提供了三项基本的服务：进程调度、进程间通信、多分部消息。①QNX 的各进程可以投递自己的请求消息，服务器在收到请求消息后根据请求进程优先级的高低来安排进程的运行，而当低优先级的进程在使用服务时，如果有高优先级的进程请求消息到来，则低优先级进程被抢占。②内核通过三个系统调用 send()、receive()和 reply()来实现进程间的相互通信。源进程向目标进程作 Send()调用后，它就被阻塞直到目标进程作了 Receive()调用，处理了消息，又作了 Reply()调用之后，源进程才能恢复执行。如果一个执行 Receive()调用的进程并没有消息悬置着等它，它就阻塞，直到别的进程对它作 Send()调用。由于消息传递机制对发送者或接收者的阻塞，它就在进行通信的进程之间起了同步作用。③多分部消息。多分部消息传输机制使得从一个进程传向另一个进程的消息不必在内存里占据单一的、相连的区域，发送进程和接收进程可以各自指定一个 MX 表，在表中指明发送进程和接收进程的消息碎片在内存中的位置。

2.3.2 uC/OS II

uC/OS II 是 Jean J.Labrosse 设计的源代码公开的嵌入式实时操作系统，已被广泛应用到 8 位、16 位、32 位的单片机上，系统经过了非常严格的测试，并且得到了美国航空管理局的认证，可用于与人性命攸关的系统和飞行器等场合，足见其可靠性和稳定性。系统只提供了一个非常微小的内核，负责任务的创建和调度、任务间的通信、服务模块间的通信等系统功能，在后续的版本中，作者又加进了内存管理模块和网络服务模块等。在调度策略上，系统采用基于优先级的抢占式调度方式，每个优先级对应着一个任务，任务的优先级也可作为任务的 ID 号，通过一张任务就绪表来管理系统中的就绪任务，调度速度快，效率高，任务调度的时间与当前的任务数无关，调度的时间复杂度为 O(1)。

2.3.3 Small RTOS51

Small RTOS51[7]是为 51 处理器设计的一个实时操作系统，其采用的调度策略与 uC/OS II 相似，基于优先级和用查表法的方式来确定下一个要运行的任务，能很好的满足系统的实时性和任务调度的可确定性。系统中通过维护着一张就绪任务的就绪表，能快速的定位到下一个将要被调用的任务，其时间复杂度是 O(1)，比起遍历就绪任务队列（时间复杂度 O(n)）性能要优越得多，但要付出的代价是就绪表需要占用一定的存储空间，是用空间来换取时间的解决方案，但在那些实时性要求高的嵌入式应用系统中，这样的开销是值得的。由于 8051 的资源少，因此 Small RTOS51 在设计时就没有考虑支持太多的任务，最大有 8 个任务和 16 个任务两种选项，根据具体的需要进行配置，可节省系统存储资源。

2.3.4 VxWorks

VxWorks 是 WindRiver 公司开发的嵌入式实时操作系统，具有良好的可靠性和实时性，易于裁减，支持 POSIX1003.1b 用户接口扩展，有一套自己的集成开发环境 Tornado，在嵌入式实时操作系统领域占有一席之地，主要用在军事、航空等领域。系统采用了微内核结构，在内核中只提供了多任务环境、任务间通信和同步等基本的功能。在调度机制上采用抢占式调度方式，有从 0 到 255 共 256 个优先级，任务在创建的时候被指定一个优先级，并且在任务运行过程中可以动态的改变优先级。在 VxWorks 上还提供了大量的服务功能，包括内存管理、TCP/IP 协议栈、网络文件系统（NFS）、远程过程调用（RPC）、C 语言解释界面、调试工具、信号和套接字、I/O 管理和文件系统等。VxWorks 作为一个商业的嵌入式操作系统，易于移植，功能强大，有着丰富的资源的良好的售后服务，但价格昂贵，一般的公司企业难以承担，并且不提供源代码。

2.4 微内核的任务管理

2.4.1 任务管理的基本原理

任务是指可并发执行的程序在一个数据集合上的运行过程，具有动态性、并行性和异步性[8]。任务管理是嵌入式操作系统的一项基本功能，也是整个系统里最核心的一部分，操作系统的性能很大程度上取决于内核的任务调度机制。按任务在运行过程中能否被抢占可把操作系统内核分为抢占式内核和非抢占式内核两种。抢占式内核的系统在运行过程中，高优先级的任务一旦就绪就能剥夺正在执行的优先级比它低的任务的 CPU 使用权，采用这种方式能够保证高优先级的任务得到优先的处理，从而提高系统的实时响应能力，如图 2.4 可抢占内核任务调度所示。在实时性要求很高的系统中采用的都是可抢占式的内核。基于优先级的非抢占式系统，中断服务程序可以使一个高优先级的任务就绪，但高优先级的就绪任务并不马上抢占低优先级任务的 CPU 使用权，中断完成后仍返回到被中断了的低优先级的任务，直到低优先级任务主动放弃处理器资源，系统重新进行任务调度时那个高优先级的任务才能得到执行，如图 2.5 非抢占内核任务调度所示，所以其实时性并没抢占式系统好，但仍比前/后台方式的要高。

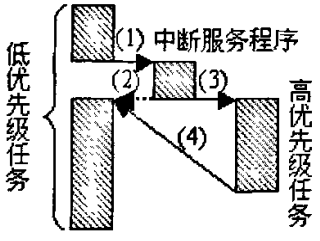


图 2.4 可抢占内核任务调度

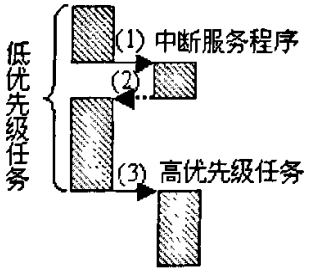


图 2.5 非抢占内核任务调度

可抢占内核执行的流程为：

（1） 低优先级的任务被中断服务程序中断。

（2） 中断服务程序没有使比被中断任务优先级更高的任务就绪，返回到被中断了的任务。

（3） 中断服务程序使优先级比被中断任务优先级更高的任务就绪，把被中断了的任务挂起，去执行高优先级的任务。

（4） 高优先级的任务完成，返回到被中断了的任务。

高优先级的任务在执行时也可能会使更高优先级的任务进入就绪状态，从而产生新的任务调度和任务切换。

非抢占内核执行的流程为：

（1） 任务被中断服务程序中断，系统转到执行中断服务程序。

（2） 中断服务程序完成后，返回到被中断了的任务。

（3） 被中断的任务执行完成，系统重新从就绪的任务队列中选择一个优先级最高的任务运行。

虽然非抢占式内核的性能要比传统前/后台方式的要好，但其时间响应能力仍是任务级的，在一些实时性要求高的场合很不适用。而抢占式的系统内核则能达到中断级的响应速度。

2.4.2 任务的划分

在多任务系统中，对任务的管理是操作系统的主要功能之一，系统中的每个任务通过时间或事件来进行驱动，操作系统根据任务的重要性和对时间要求的严格程度，为任务设定优先级，在需要进行任务调度时根据任务的优先级和系统的调度策略对系统中的任务进行仲裁，选择一个合适的任务投入运行。在微内核结构的系统中，任务是具有一定功能的程序模块，如数据采集、时钟显示、键盘输入等都可以设计成一个任务，然而，在把一个大的应用划分为多个任务时则很讲究，如果所设计的单个任务太大、功能过多，系统的实时性就会受到影响。如果任务划分得太小，虽然任务的响应速度可以提高，但任务切换太频繁，会加重系统的负载，使得吞吐量下降。对任务的划分一般应遵循以下一些原则：

（1）每个任务的运行时间都要小于系统对输入做出响应的最大允许时间，从而保证系统的实时性[1]。

（2）各任务在功能上应尽量相互独立，以方便任务的调试和维护。对需要互斥处理的一些功能和操作则应放在一个任务里，或用一个任务来控制它们所要使用的资源，那些需要使用这些资源的互斥任务就通过调用这个任务，完成它们之间的通信。

(3)任务尽量不要用中断服务程序来实现，因为如果中断服务程序执行的时间过长，就会使得其它的一些异步信号得不到响应，从而影响系统的实时性。

2.4.3 任务的调度策略

决定什么时候以什么样的方式从就绪队列中挑选一个任务投入运行的一组规则称为调度策略，调度器裁决的过程称为调度。嵌入式操作系统的调度策略一直都是嵌入式操作系统研究的一个热点，是嵌入式操作系统内核的关键部分，如何调度任务，才能使系

统满足应用要求也是嵌入式操作系统的重要研究内容。一个经过精心设计的调度策略可以建立一种所有进程都在同时被执行的假象，而一个糟糕的调度策略则会使用户的响应时间加长，系统的吞吐率下降，整个系统变得缓慢[1]。因此调度器必需经过高度优化，选择合适的调度策略，才能合理的调度多任务并快速的切换。从调度策略的定义知道，一个调度策略包含两个要素：选择函数和决策方式。

选择函数是用来确定如何在就绪队列的多个任务中选择一个任务来运行的某种规则，也叫做调度算法。在操作系统的设计中可用的调度算法很多，如先来先服务算法、基于优先级算法、短作业优先算法、时间片轮转算法、基于时间驱动的调度算法、基于比例共享的调度算法等。

（1）先来先服务算法中把所有的就绪任务设置成一个链表结构，当一个新的任务进入就绪队列时把它加在链表的尾部，而要调度时就从链表的头部取一个任务来运行，这种调度算法实现起来容易并且任务切换开销少，但系统的响应时间长，对短任务不利。

（2）基于优先级的调度算法根据任务的轻重缓急来为任务分配不同的优先级，在调度时，优先级高的任务优先得到服务，这种调度算法能提高系统的响应速度，使紧急的任务优先得到处理，但在一些极端的情况下，这种调度算法会引起低优先级任务出现饥饿。基于优先级的调度算法又可分为两种类型：静态优先级调度算法和动态优先级调度算法。静态优先级调度算法：该算法给系统中所有任务都静态的分配一个优先级，静态优先级的分配可以根据应用的属性来进行，例如任务的周期、用户优先级或者其他预先确定的策略，RM（RateMonotonic）是一种典型的静态优先级调度算法，它根据任务执行周期的长短来决定调度优先级，执行周期小的任务具有较高的优先级。动态优先级调度算法：该算法根据任务的资源需求来动态的分配任务的优先级。EDF（Earliest Deadline First）算法是一种典型的动态优先级调度算法[10]，该算法根据就绪队列中各个任务的截止期限来分配优先级，具有最近截止期限的任务的优先级最高，采用这种方式的调度器，如果系统的负载很轻，则在抢占式的系统和非抢占式系统中都能表现出很好的性能[11]。然而当系统负载过重时，EDF 调度算法的性能将急剧下降，一些学者已经对此作了一些改进，改进的调度算法有 SCAN-EDF[12]、gEDF[13]等。SCAN-EDF 是短作业优先（SJF）和最早时限优先（EDF）两种调度算法的结合，在进行调度时如果两个任务的结束时限相同，则运行时间需要少的那个任务能优先得到执行。gEDF 算法把结束时限相近的任务归到一组里，第 i 个任务可以这样描述 $T_i=(R_i,E_i,D_i,P_i)$，其中，R_i 是任务到达的时间，E_i 是任务运行所需要的时间，D_i 是任务结束时限，P_i 为任务运行的周期或用户指定的其它值。用 $d_j=R_i+E_i$ 表示任务的动态结束时限，则在 gEDF 算法中，任务 i 和 j 如果有 $d_i \leqslant d_j \leqslant (d_i+Gr(d_i-t))$这样的关系，则它们属于同一组，Gr 为一个给定的分组参数，t 为当前时间。

（3）时间片轮转调度算法为每个就绪的任务都分配一个称为“时间片”的相等的时间，这个时间表示任务一次允许运行的时间。在进行任务调度时，从就绪任务里取一个任务来运行，如果这个任务的时间片用完了而进程还没结束就把它挂到就绪队列尾部并重新设置时间片等待下一次调度，然后再从就绪链表里取下一个任务来运行。时间片轮转的调度算法容易实现，对短任务有较好的响应时间，对所有的任务进公平对待，任务切换开销小，吞吐量由时间片的大小来决定。

（4）基于比例共享的调度算法[12]：该算法的基本思想就是按照一定的权重（CPU 使用的比例）对一组需要调度的任务进行调度，使其执行时间与权重完全成正比。可以通过两种方法来实现比例共享调度算法：①调节各个就绪进程出现在当前调度队列首的频率，并调度队列首的进程执行；②逐次调度就绪队列中的各个进程投入运行，但根据分配的权重调节分配给每个进程的运行时间片。

决策方式是系统调用选择函数调度任务时所采用的一种仲裁方式。因为在单处理机的系统中某一时刻只能有一个任务是处于运行状态的，其它的任务或者被阻塞或者处于就绪队列中。决策方式负责确定什么时候把选择函数从就绪队列里选出的任务投入运行，也就是一个调度时机的问题[14]。而在一个实际的系统中，采用的调度算法往往不只一种，而是把几种算法结合起来，这样才能使系统适应各种情况，性能达到最优。

2.5 微内核的通信机制

在微内核结构的系统中，任务和内核服务器都是内核之外的独立模块，因此任务与任务之间、任务与服务器之间以及服务器之间要进行相互的协作，只能通过内核来进行沟通，而内核采用得较多的是消息机制来进行沟通，内核作为消息的一个中转站，负责接收请求模块发来的消息，然后发送到目标模块处理，再把执行情况从处理消息的目标模块转发到请求模块。微内核采用的通信方式有信号量、邮箱、消息队列等多种方式。

信号量是一种协议机制，可以用于控制对共享资源的互斥访问、任务间的同步和标志事件的发生。信号量在对共享资源的互斥保护中相当于一把钥匙，任务如果要对共享资源进行访问，首先必须得到这把钥匙。当这把钥匙分配给一个任务 taskA 后，如果另一个任务 taskB 再试图访问共享资源就会因得不到这把钥匙而必需进入等待状态或被挂起，直到任务 taskA 释放这个信号量后 taskB 才能继续执行。

邮箱由一个指向消息的指针和等待该邮箱消息的任务列表组成，通过内核服务，一个任务或一个中断服务程序可以把一条消息（也就是指向该消息的指针）放到邮箱里，同样任务也可以通过内核服务程序接收这条消息，如果任务要从邮箱中取消息，而此时邮箱中消息为空，就把任务放到等待列表里，当等待的消息到来时再唤醒任务，使任务

就绪，当然如果定义了超时等待，在等待时间到时如果仍没有消息到来，则也会把任务就绪，但要返回出错信息。

消息队列可以看成是多个邮箱的结合。一个邮箱只能存放一条信息，如果要容纳多条消息，则可以采用消息队列的方式，一个消息队列就相当于一个消息的缓冲池，如果队列未满则任务或中断服务程序就可以向队列中放消息，同样，当队列非空时任务或中断服务程序可以从队列中取得消息。消息队列的一般结构形式如图 2.6 所示。

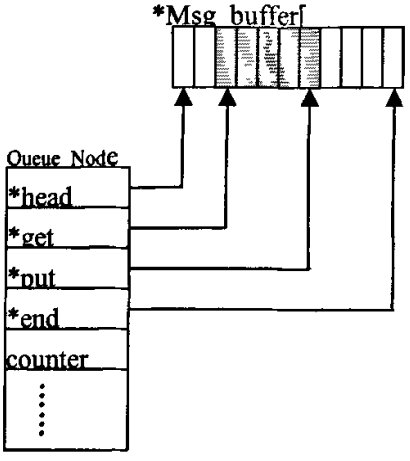


图 2.6 消息队列结构

一个好的嵌入式操作系统能够支持任务间的相互通信和很短的延迟时间，并且这种通信是很健壮的，能够防止数据损失的情况出现，如果通信不能完成，系统应该能检测得到并给出错误信息。

2.6 优先级反转问题及其解决方法

2.6.1 优先级反转问题的产生

在基于优先级的抢占式系统中，由于控制临界资源访问的互斥信号量的使用会使得系统存在优先级反转的问题。所谓优先级反转是指多任务共享资源，一个任务等待优先级比它低的任务释放资源而被阻塞，反而使得中等优先级的任务先于高优先级的任务得到执行，如果这时有多个中等优先级的任务就绪，阻塞就会进一步恶化，甚至会使得高优先级的任务超过了最后期限而引起灾难性的后果[15]。因为在实时性要求高的系统中，不仅要求系统的逻辑正确，而且还要求在规定的时间内完成某项任务，如果超出了这个规定的时间将会导致致命性的错误。

为了解决优先级反转问题，先研究一下优先级反转问题是如何产生的。假设系统中有 TASK1 TASK2 TASK3 三个任务，其优先级为 TASK1>TASK2>TASK3，（为方便，假设一个优先级只对应一个任务，一个优先级对应多个任务的情况与此相同）。TASK1 和 TASK3 共享一个资源 A，使用资源 A 前要先得到信号量 S。某一时刻 t0，任务 TASK3 在运行，需要使用资源 A，并得到了信号量 S，此时任务 TASK1 和 TASK2 在等待事件的发生，处于挂起状态。当运行到 t1 时刻时，任务 TASK1 等待的事件到来，从而抢占了任务 TASK3 的 CPU 使用权。TASK1 运行到 t2 时刻时，需要使用共享资源 A，而此时共享资源 A 被 TASK3 占有，TASK1 因无法获得信号量 S，只能挂起，等待 TASK3 释放信号量 S，并把 CPU 使用权交给 TASK3，让 TASK3 继续运行并希望 TASK3 尽快的执行完毕，以释放共享资源 A，而当 TASK3 运行到 t3 时刻时，TASK2 等待的事件到来，TASK2 又抢占了 TASK3 的 CPU 使用权。直到在 t4 时刻 TASK2 主动放弃 CPU 使用权后，TASK3 才得以继续运动，并在 t5 时刻完成对共享资源 A 的使用，释放信号量 S，此时任务 TASK1 因能得到信号量 S，才得以继续运行。TASK1 TASK2 TASK3 执行的时间关系如图 2.7 所示。从图上可以看出任务 TASK1 因任务 TASK2 的加入，被延时了 TASK2 执行的时间。出现这种情况，使得 TASK1 的优先级降低到了 TASK3 优先级的水平，而使得 TASK2 都能抢先 TASK1 得以运行了，这就是优先级反转的问题。

从图 2.7 可以看出，之所以会出现优先级反转，是因为任务 TASK1 和任务 TASK3 都要使用一个共享资源 A，而 TASK3 在还没完成对资源 A 的使用时又被比它优先级高的 TASK2 抢占了，这样就变成了 TASK2 先于 TASK1 优先得到完成，这就是优先级反转问题。如果使用同一共享资源的两个任务优先级相差很大，则情况将会更糟糕，这样，高优先级的那个任务将会因等待时间过长而错过了最后的执行期限。

2.6.2 解决优先级反转问题的两种方法

在嵌入式系统中处理优先级反转问题时采用得较多的是优先级置顶和优先级继承[16][17]两种方式。

优先级继承是当任务 B 申请共享资源 S 时，如果 S 正在被任务 A 使用，通过比较任务 A 与自身的优先级，如发现任务 A 的优先级小于自身的优先级，则将任务 A 的优先级提升到自身的优先级，以避免优先级介于任务 A 和任务 B 之间的任务抢占任务 A 的执行，任务 A 释放资源 S 后，再恢复任务 A 到原来的优先级，优先级继承的关系如图 2.8 所示。

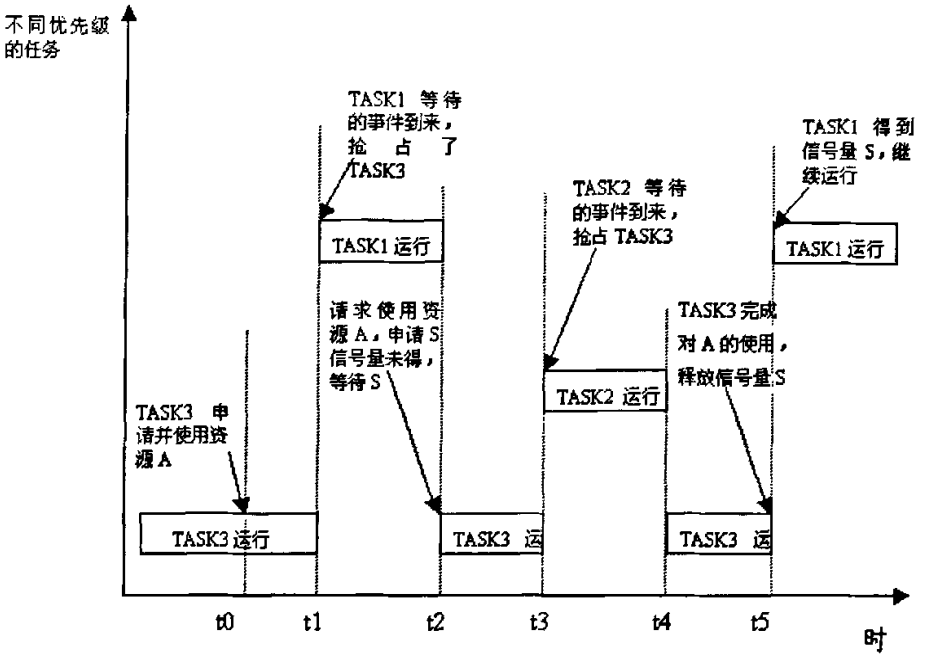


图 2.7 优先级反转情况

在图 2.8 中，TASK_A、TASK_B、TASK_C 三个任务共享资源 D，优先级关系为：TASK_B > TASK_A >TASK_C，图中有阴影部分表示任务在运行。在 T0 时刻 TASK_A 申请共享资源 D，因为系统中没有其它的任务在使用资源 D，所以 TASK_A 得到了资源 D 并继续运行，在 T1 时刻高优先级的任务 TASK_B 申请共享资源 D，因为此时 TASK_A 还没完成对共享资源 D 的使用，所以 TASK_B 只能挂起等待，同时系统把 TASK_A 的优先级提高到和 TASK_B 相同的水平。在 T1 到 T2 这段时间内又有 TASK_C 对共享资源提出申请，但其优先级比占有共享资源的 TASK_A 低，所以只是简单的把它挂起。T2 时刻 TASK_A 完成对共享资源的访问，释放了资源 D，并把其优先级降低到原来的水平。此时等待资源 D 的有 TASK_B 和 TASK_C，因为 TASKB 的优先级比 TASK_C 的优先级高，所以 TASK_B 获得共享资源继续运行，到 T3 时间 TASK_B 主动放弃处理器，TASK_C 获得共享资源继续运行。

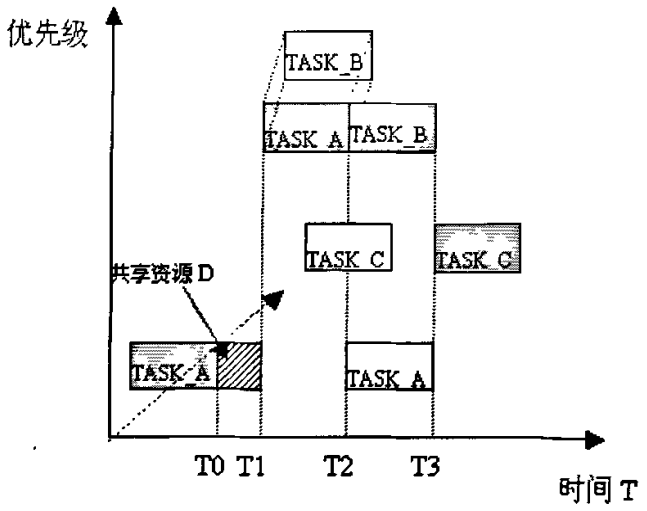


图 2.8 优先级继承关系图

优先级置顶的方式是在系统中设置一个顶层优先级和各个临界资源相关联，当任务需要使用临界资源时，系统就把这个顶层优先级传递给这个任务[18]，而使得这个任务的优先级成为所有可能使用此共享临界资源的任务中优先级最高的那个，这样就不会出现优先级反转的情况了，当任务完成对临界资源的使用后再把它恢复到原来的优先级。

优先级继承与优先级置顶的比较：

（1）优先级继承可能多次改变占有某临界资源的任务的优先级，从而引入更多的额外开销。而优先级置顶只需改变一次优先级，具有较高的效率。

（2）优先级继承的方式任务被阻塞的最长时间 Tmax= $\sum_{i=1}^{n} Ti$，而优先级置顶方式任务被阻塞的最长时间 Tmax=maxTi, 其中 Ti 是阻塞任务 T 在临界区执行的时间。从分析可以看出，优先级置顶方式能改善最坏的情况。

（3）优先级继承只有需要时才改变优先级，而优先级置顶不管是否需要都抬升优先级，因而对应用中预先设定的任务流程影响较大。

（4）优先级置顶法需要在系统中事先确定出试图获取某一共享资源的任务中优先级最高的任务，这在一个大的系统中是很困难的，而优先级继承法不需要这个先验信息，所以它的应用条件更为宽松。

2.6.3 优先级继承的一些基本特性

在优先级继承的系统中，任务具有以下的一些性质[19]：

（1）在优先级继承的情况下，如果当前可运行的任务中的最高优先级为 Pi，则运行任务的当前优先级为 P_i 。

（2）如果任务中有任务 $Task_0$ 和 n 个优先级比它低的任务 $Task_1$~Taskn 则对每一个临界区集 $\beta_{0.1}^*$，$1 \leqslant i \leqslant n$，$Task_0$ 被 $Task_i$ 阻塞的时间最多是一个临界区 Z_i 的执行时间，$Z_i \in \beta_{0.1}^*$ 。

（4） 如果有 n 个信号量可以阻塞任务 T，则 T 最多可以被阻塞 n 次。

第3章 操作系统内核的设计

3.1 内核整体结构

在设计操作系统时，可以采用对象、层次和模块三种方法来进行设计，用得比较多的是层次化设计和模块化设计，而对象的方法目前在操作系统的设计中用得并不是很多。采用层次化设计的系统便于移植，在移植时只需要改写最底层的代码即可，但层次结构的系统执行速度慢，高层的服务必须通过一系列低层的调用才能实现，如第N层要调用N-1层，N-1层又调用N-2层等等。模块化设计按功能把系统分成若干个模块，对外部只提供一个抽象的接口，而把模块的具体实现隐藏了起来。模块化设计的系统各模块之间可以直接调用或通过消息机制通信来协调工作，而层次化的系统，只能是上层调用相邻的下层。

本文设计的系统采用模块化结构，最多可支持64个优先级队列256个任务，同一优先级队列上可以有多个任务，对于不同优先级队列的任务采用抢占式的调度策略，对于同一优先级队列上的任务采用时间片轮转的分时调度策略，整个系统中总是就绪的优先级最高的那个队列的任务优先得到执行。整个系统内核分为任务调度、任务通信、中断管理、临界资源管理、时钟管理、内存管理等几个模块，系统内核的整体结构如图3.1所示。

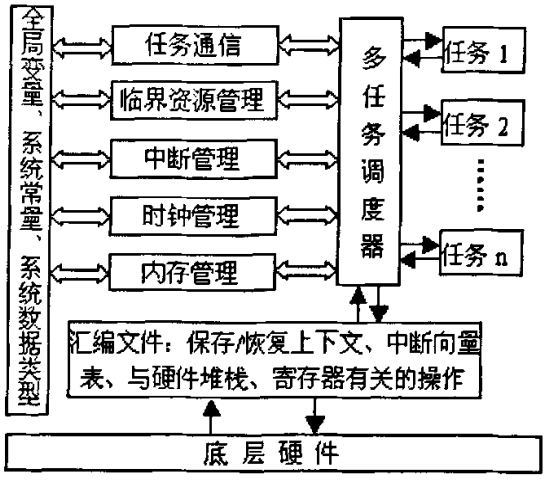


图3.1 系统内核结构

3.2 系统数据类型设计

为了便于系统统在不同硬件平台的移植，笔者在设计的系统中不直接采用 int、long 等数据类型，因为这些数据类型与处理器和编译器有关，而是采用更直观的数据类型表示方式，如 8 位无符号整数就表示成 UINT8，这样把系统移植到具体的硬件平台时，可根据具体的处理器和编译器来进行设定，如可以用 typedef unsigned char UINT8 来定义 UINT8。系统中用到的一些数据类型说明如下：

UINT8 ：8 位的无符号整数，值为 0~255

SINT8 ：8 位有符号整数，值为-128~128

UINT16 ：16 位无符号整数，值为 0~65535

SINT16 ：16 位有符号整数，值为-32768~32767

UINT32 ：32 无符号整数，值为 0~4294967295

SINT32 ：32 位有符号整数，值为-2147483648~2147483647

TASK_STACK：保存任务上下文时用到的堆栈宽度，如 8051 上堆栈宽度为 8 位，在 ARM 上堆栈宽度为 32 位

BOOLEAN：布尔类型

3.3 任务管理模块的设计

3.3.1 任务状态

任务的管理分为任务的创建、任务的删除、任务的挂起、任务的恢复、设定任务优先级、任务就绪、任务运行等。系统中的任务总是处于休眠、就绪、运行、挂起、被中断五种状态中的一种，它们之间的转换关系可用图 3.2 表示。

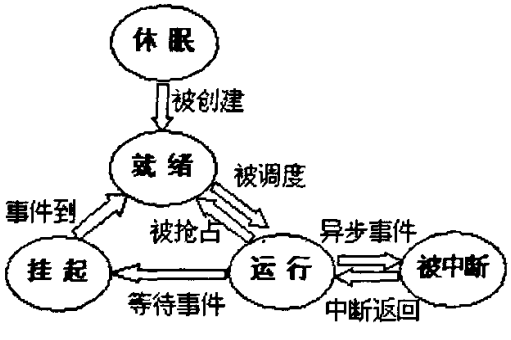


图 3.2 任务状态转换图

休眠状态的任务是驻留在存储器中，还未被内核创建使用的形态；就绪状态指的是任务已获得其运行所需要的除 CPU 外的所有资源的状态[20]；等待或挂起状态指的是等待除 CPU 之外的其它资源的状态，等待的可能是 I/O 操作、信号量等；运行态是指任务获得 CPU 资源，正在运行，在单处理器的系统中，某一时刻最多只能有一个任务处理于运行态；由于异步事件的发生，任务被中断服务程序中断，任务就处于被中断状态。

3.3.2 任务控制块的设计

任务也叫作进程，是一个具有独立功能的程序对某个数据集在处理机上的执行过程和分配资源的基本单位[21][40]。任务的执行是动态的，具有并行性、独立性和异步性的特点，为了描述和控制多任务的运行，在系统中为每个任务定义了一个任务控制块 TCB（Task Control Block）包含有与任务相关的所有信息，这些信息称之为任务的上下文，如任务执行时 CPU 的所有寄存器中的值、任务的状态以及任务堆栈的内容等，当任务被中断或等待资源被挂起而进行任务切换时把所有的上下文消息保存 TCB 中，任务再次运行时把 TCB 的内容进行恢复，这样任务就像未被中断过一样。本系统在处理寄存器和堆栈等上下文内容时，没有像 LINUX 等操作系统那样专门定义一个数据结构，而只是在任务控制块里用了一个指针 task_stk_ptr 指向这部分内容的存储空间，具体的大小用户可以自己指定，这样更灵活，便于移植[21][22]。当要创建一个任务时首先要从空闲任务控制块队列中取出一个任务控制块，根据任务的属性要求为任务设定优先级、堆栈位置等信息，然后把任务放到相应的就绪列表中。相反，当要删除一个任务时，就把任务控制块释放，挂到空任务控制块队列上。任务创建有静态创建和动态创建两种，静态创建是任务在多任务系统启动之前就已经被创建好，而动态创建则是由运行的任务或中断服务程序来创建。任务控制块 TCB 的结构如下：

```
typedef struct task_tcb{
    TASK_STACK   *task_stk_ptr; /*指向保存任务上下文堆栈的指针*/
    UINT8 task_id; /*任务号，必须唯一，*/
    UINT8 task_priority; /*任务的优先级，未提升时与 primitive_priority 同*/
    UINT8 primitive_priority; /*任务的原始优先级，正常*/
    UINT8   task_name[15]; /*保存任务名的字符数组*/
    struct task_tcb *task_previous; /*指向任务链表中的前一个任务控制块*/
    struct task_tcb *task_next; /*指向任务链表中的后一个任务控制块*/
    UINT16 total_time; /*时间片轮转调度时分配给任务的时间数*/
```

```
UINT16 remain_time; /*当被高优先级的任务剥夺时，如果时间片未用完
                      则保存在此，任务刚开始被调度时与 total_time 相同*/
UINT8 task_state;   /*任务状态*/
Void *task_wait;    /*指向任务等待的信号量、事件、或消息的指针*/
 }TASK_TCB;
```

3.3.3 优先级队列结点的设计

在优先级队列表里，每个元素控制着这一优先级队列的任务，系统支持的最大优先级数为 64 级[23]。在进行任务调度时首先通过 task_queue_grp 和 task_queue_tbl[ ]这两个变量的值来确定当前就绪的任务中优先级最高的任务所在的队列，然后让最高就绪优先级队列指针 queue_high_ready_pt 指向该队列，如果队列的优先级比当前优先级队列指针 queue_current_pt 所指向的优先级要高，则保存正在执行的任务的上下文，把当前任务队列指针指向最高就绪优先级队列指针所指的队列，实现运行任务队列的切换，然后从这个队列的队列头取一个任务来运行。在运行中系统总是对 queue_current_pt 指针所指向的队列采用时间片轮转的调度算法，当任务时间片用完后，重新设置时间片，并挂到队列尾，再从队列头取一个任务来运行。如果队列上只有一个任务时，系统重新设置任务时间片后，继续运行该任务。当任务在运行过程中有新的任务就绪了则需要通过优先级队列表重新定位优先级最高的任务队列，如果需要切换任务队列，但当前任务的时间片还没用完，则它并不挂到队列末尾，而是放在队列头，直到在下次调度中用完了剩余的时间片，才重新设置时间片的值，然后把任务挂到队列尾。任务队列结点 QUEUE 的数据结构如下：

```
typedef struct queue{
        UINT8 queue_priority; /*队列优先级*/
        UINT8 task_counter;   /*连在本队列上的任务数*/
        TASK_TCB *queue_head; /*队列头指针*/
        TASK_TCB *queue_end; /*队列尾指针*/
        }QUEUE;
```

就绪任务和任务队列的关系如图 3.3 所示。优先级表 Queue_Table[]中各元素存放的队列的优先级即为其下标的值，如元素 Queue_Table[1]存放的即是优先级为 1 的任务队列。

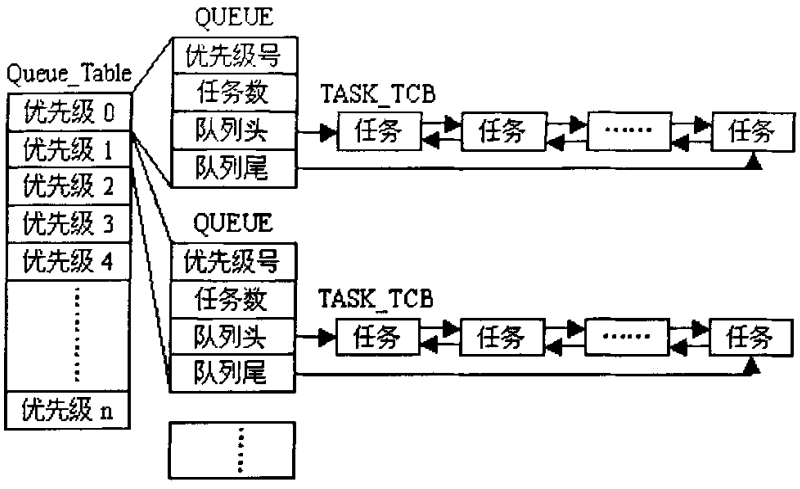


图 3.3 就绪任务和任务队列关系图

3.3.4 任务的创建

创建任务可以通过调用 tsk_create(void(*task)(void),TASK_STACK *ptos,UINT8 prio,UINT8 id,UINT8 *pname)函数来完成，通过参数把任务的各种信息，如任务代码的起始地址、优先级、任务 id 号、堆栈地址等填入到任务控制块 TCB 中，如果任务创建成功则返回一个指向任务的指针，以后对任务的所有操作，如任务的调度、删除、挂起等都通过这个指针所指向的任务 TCB 来进行。保存任务 CPU 各寄存器和堆栈等上下文的空间在系统里静态给出，在创建任务时把空间的起始地址传给 tsk_create( )函数。要创建一个新的任务，首先要看为任务指定的任务 ID 号是否已经被占用，因为任务的 ID 号在系统里必须是唯一的，如果已经被占用，则返回任务 ID 号被占用的错误信息。如果 ID 号未被占用，就看系统中是否还有空闲的任务控制块，如果空闲任务控制块已被用完，则返回任务控制块被用完的出错信息，否则就从空闲任务控制块队列中取出一个任务控制块，然后把任务的各种信息都填入任务控制块，再把此任务挂到相应优先级的就绪任务队列上。任务可以在系统启动多任务调度前被创建，也可以在系统启动多任务后由任务或中断服务程序来创建。如果系统已经处在多任务运行状态，即系统多任务运行标志 task_running 为 TRUE 则在任务创建结束或中断退出时调用 tsk_schedule()函数来重新调度任务，看是否有新的更高优先级的任务就绪。如果 task_running 为 FALSE 则任务创建结束后直接返回，创建任务的流程图如图 3.4 所示。任务创建成功系统任务记数器 task_counter_created 进行加 1，当删除一个任务时 task_counter_created 减 1，task_counter_created 指示系统中任务的个数。

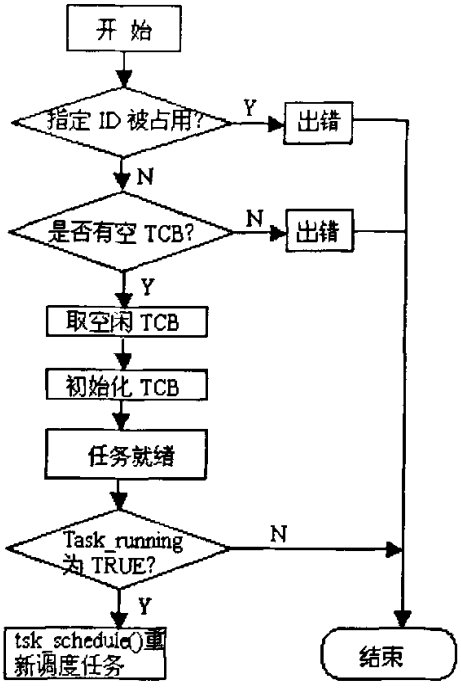


图 3.4 任务创建流程

在系统里设计了一个任务控制块类型的指针数组 TASK_TCB *tsk_array_pt[ ]来维护系统中所有的任务，数组里的元素指向 ID 号与其下标相同的任务，结构如图 3.5 所示，如果该 ID 的任务不存在则指针为空，这个指针在系统中起到任务句柄的作用，这个句柄是在任务创建时被返回的，以后对任务的很多操作，如任务间的通信、把挂起的任务就绪等都可以通过这个句柄来进行。在系统里只要任务不被删除，不管它是处于哪个状态，通过 tsk_array_pt[ ]指针数组都能找到任务。任务指针数组与任务的关系如图 3.5 所示。

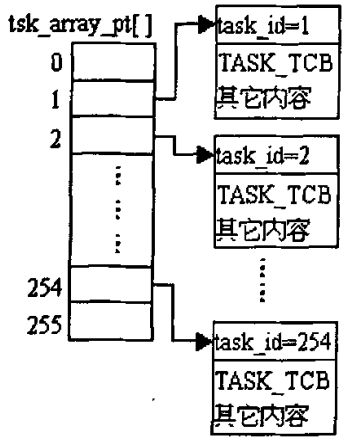


图 3.5 任务指针数组

3.3.5 任务的调度

多任务操作系统内核管理着多个任务，而系统只有一个处理器，所以系统在给定的时刻最多只能有一个任务处于运行状态，怎么合理的为这些任务分配处理器，提高系统的性能，满足实际应用的要求是操作系统要解决的一个重要问题。为了使就绪队列中的若干个任务都能及时的获得 CPU，在操作系统的内核里设置了专门的程序来裁决就绪任务的运行次序，这个程序称这为调度器。衡量一个调度器的好坏可从定性和定量两方面来进行衡量，定性指系统的安全性，也就是在一次任务调度中是否会引起数据结构的破坏；定量指的是系统的开销，这也是衡量一个调度器好坏的重要指标，由于调度器的执行会引起任务之间上下文的切换，如果任务切换得太频繁将占用过多的系统资源，增大系统开销，降低系统的整体性能[24]。在本文设计的系统里会引起任务重新调度的几种情况是：

（1）运行的任务因缺乏资源被阻塞。

（2）同一优先级上有多个任务，并且运行任务的时间片已用完。

（3）有新的任务就绪。

（4）中断处理程序退出时。

（5）任务运行结束

（6）任务被删除

在第二章任务调度策略部分已经对任务的各种调度算法进行了分析，本文设计的系统结合了基于优先级抢占式调度和基于时间片轮转的分时调度两种方式，通过对这两种方式的结合能更好的满足嵌入式系统的应用要求，从而能弥补单独使用这两种方式的不足。（1）基于优先级的抢占式调度使得系统能快速的处理紧急的任务，只要有比当前任务优先级高的任务就绪就能马上的剥夺 CPU 资源，从而能提高系统的实时性。（2）同优先级任务采用时间片轮转分时调度，使得实时系统中优先级相同的任务具有平等运行的权利[25]，从而方便编程，是系统在最高就绪优先级队列上有两个或多个任务时采用的一种调度策略。

基于优先级的抢占式内核能很好的满足系统的实时性要求，为何还要增加时间片轮转的调度方式呢？首先来看看时间片轮转的必要性。假设有两个任务，TASKA 和 TASKB，他们具有的重要程度相同，希望系统在任务调度时能够得到同等的对待，但如果系统中只有抢占的方式，每个优先级只能对应一个任务，就必须为这两个任务分配不同的优先级，假设 TASKA_PRIO>TASKB_PRIO，某一时刻 TASKB 在运行，TASKA 因等待的事件发生或延时结束变为了就绪态，由于系统是抢占的，TASKA 必然会抢占

TASKB，并且直到 TASKA 主动放弃处理器后 TASKB 才能得到运行，这样 TASKB 就有可能错过期最后的运行期限，而实际上这两个任务是同等重要的，在这种情况下发生抢占显然是不合理的，而基于时间片轮转的方式则能很好的解决这个问题。当正在运行的优先级队列上有多个任务时，从队列头取一个任务让它运行一小段时间，之后即使没有完成也必需停止，释放处理器，挂到队列尾，等待下一次调度，而让队列上的下一个任务运行。任务运行的这一小段时间，称之为时间片，时间片大小的选择很讲究，时间片设置得太大就失去轮转调度的意义了，变为先来先服务（FCFS）了，时间片设置得太小任务切换过于频繁，就会加重了系统的负载。如果当前任务所在优先级的队列上只有一个任务，则让任务一直运行下去，直到被更高优先级的任务抢占，或自动放弃处理器。

在本文设计的系统里，系统的任务优先级队列通过一张优先级队列就绪表[26]来进行管理，最多可管理 64 个优先级队列，256 个任务，每个优先级队列上可有多个任务。在进行应用系统开发时，如果所有的任务都定义在一个优先级上，则系统就完全变成了基于时间片轮转的系统了；如果每个优先级上最多只设置了一个任务则系统就是完全的抢占式系统。当一个任务运行结束或挂起、系统在运行过程中有新的任务被创建、有资源被释放使得新的任务就绪或任务提出资源申请得不到满足等都可能使调度器重新调度任务。

在系统里通过对优先级队列就绪表进行查表能快速的找到优先级最高的就绪队列，而且与系统中当前的就绪任务数无关，比遍历任务队列的方法性能要优越。当查找到下一个将要运行的优先级队列后，队列头的那个任务即为将要调度运行的任务，就绪任务和其对应的队列关系如前面的图 3.3 所示。在定位优先级队列时要用到两个量：优先级表 task_queue_tbl[]和优先级组变量 task_queue_grp。之所以把任务优先级队列设计成 64 个，是为了在提高系统性能的同时，能节省系统的存储空间，因为查找表随优先级数的增加，成倍的扩大，且 64 个优先级已能满足绝大部分的应用要求。在设计时把 64 个优先级分成了 8 组，每组 8 个，这样，查找表就不会过于庞大。优先级表是一个数组 task_queue_tbl[8]，有 8 个元素，每个元素都是一个 8 位的无符号数，每一位表示一个优先级，如果该优先级队列上有就绪任务则该位置 1，否则清 0，同理，如果某组上有任务则 task_queue_grp 里与该组相关的位置 1，否则清 0。task_queue_grp 与 task_queue_tbl[]的关系如图 3.6 所示。

在需要重新定位任务队列时，系统就根据 task_queue_grp 里的数值和 task_queue_tbl[]对应组里的数值查 search_table[]表得到最高就绪任务队列的优先级。search_table[]是一个事先设计好的常量数组，如图 3.7 所示，数组有 256 个元素，每个元素为一个 8 位的无符号整数，其内容通过计算来填好。如，task_queue_grp 里的数值为 00100100，即 0x24，用 task_y 来表示任务所在的组，通过查 search_table[]，有 task_y=search_table[0x24]=2，

可知就绪任务中优先级最高的任务必定在第 2 组里，即 task_queue_tbl[2]里（task_queue_tbl[0]~task_queue_tbl[7]共 8 组）。具体是第 2 组里的第几个元素呢？又通过对第 2 组里的数据查表即可得，在这里，第 2 组的数据 task_queue_tbl[2]=0x62，用 task_x 表示指定组里的第几个元素，则有 task_x=search_table[0x62]=1，即系统中优先级最高的任务是第 2 组的第 1 个元素（每组有 0~7 共 8 个元素）。用 highest_pri_rdy_num 来表示就绪表中优先级最高的优先级号，则有 highest_pri_rdy_num=2*8+1=17，即就绪表中，优先级为 17 的任务队列优先级最高。

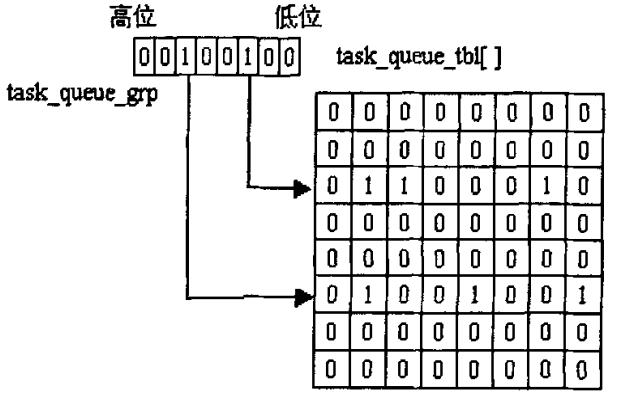


图 3.6 优先级组与优先级表关系

search_table[ ]

0	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
1	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
2	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
3	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
4	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
5	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	1
6	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
7	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
8	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
9	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
10	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
11	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
12	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
13	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
14	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
15	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

图 3.7 优先级查找表

当被调度队列的优先级确定后，就能很容易找到将被切换进的任务了，然后看任务队列上是否有多个任务，如果有多个任务则把系统模式 sys_mode 置为 SYS_MODE_ROLL，系统在没有更高优先级任务产生时就轮流调度此队列上的任务，如

果只有一个任务则把 sys_mode 置为 SYS_MODE_PREM，然后进行任务切换。任务调度的部分代码如下：

```
void tsk_schedule(void)
{
  UINT8 i,priority,taskx,tasky;
  disable_int( );  /*关中断*/
  tasky=search_table[task_queue_grp];  /*就绪的最高优先级任务所在的组*/
  i=task_queue_tbl[tasky];
  taskx=search_table[i];          /*就绪的最高优先级任务所在的列*/
  priority=tasky*8+taskx;
  highest_pri_rdy_num=priority;  /*系统当前就绪的最高优先级*/
  queue_high_ready_pt=&queue_table[priority];  /*最高优先级队列*/
  if (queue_current_pt->queue_priority> highest_pri_rdy_num) /*是否需切换对列*/
 {
  queue_current_pt=queue_high_ready_pt;
  if(queue_current_pt->task_counter>1)   /*时间片轮转模式设置*/
     sys_mode=SYS_MODE_ROLL;
  else
     sys_mode=SYS_MODE_PREM;
  task_high_ready_pt=queue_high_ready_pt->queue_head;/*将被运行的任务*/
  queue_high_ready_pt->queue_head=queue_high_ready_pt->queue_head->task_next;
  context_sout( );  /*任务切换*/
  task_current_pt=task_high_ready_pt;  /*当前任务指针*/
  context_swin( );
}
  enable_int( );  /*开中断*/
}
```

3.4 任务间通信和同步机制的设计

3.4.1 任务间通信需求分析

微内核结构的嵌入式操作系统是建立在多任务间通信的概念上的，一个多任务环境可将实时应用程序构造为一种独立任务的集合，每个任务都有其单独的执行线程和自己的系统资源集合。任务间存在着各种复杂的逻辑关系，必须通过通信在任务之间进行信息交换，通过通信使得这些任务协调的工作。在嵌入式系统中，其通信机制应该是健壮的，要能够防止数据丢失情况的出现，对于不能完成的通信，系统可以检测出来[27]。嵌入式系统的任务间在进行通信时，使用消息传递机制会带来很多好处，特别是对那些需要高可用性的系统，如，一年只允许有不多于一秒钟的停机时间的系统，消息传递机制的使用显得尤为重要。消息通信有多种方式，如信号量、邮箱、消息队列等。信号量简单、快捷，但是用信号量也有它的缺点，例如，信号量容易造成任务死锁，比如当一个任务等待的信号量被另一任务获得，而此任务已被删除或由于种种原因不能释放信号量，这种情况下，信号量会永远丢失，等待的任务就要一直地等待下去。使用信号量还经常受优先级反转问题的困扰，最著名的一次优先级反转引发事故的是 1997 年 12 月的 RISKS-FORUM DIGEST 登载的火星探路者事件，系统由于一个低优先级的任务阻挡了高优先级的任务，且未发出任何警告和表示，而造成了灾难性的后果。系统在遇到优先级反转问题时，会引起不可预测的系统时延，从而降低实时性。

消息管理器用于提供任务间通信和同步的能力。在系统里消息是一个可变长度的缓冲区，用于存储任务间通信的信息，消息的长度和内容由用户定义，可以是真实的数据、指针或为空。消息队列也是消息机制中的一种，允许在任务或者中断服务程序ISR 之间传递信息，消息队列能包含零个或者多个消息。消息管理器中的信号量还能为任务提供异步通信的手段，任务、中断服务程序与信号量、消息之间的关系如图3.8所示。

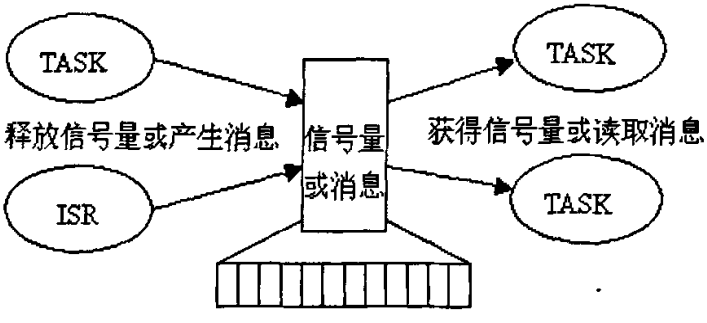


图 3.8 任务、中断服务程序与信号量、消息的关系

3.4.2 信号量

3.4.2.1 信号量数据结构的设计

信号量[28]在系统中主要用于保护系统的临界资源、指示事件的发生、同步任务等，可分为二值信号量与计数信号量两种。二值信号量在用于保护临界资源时，也可称作互斥信号量，它就好比是一把钥匙，要访问临界资源的任务必须先得到这把钥匙才能运行，如果此时信号量已被其它任务占有，则它只能挂起，直到信号量被释放。当信号量被释放时，如果有多个任务在等待它，就把他分配给等待的任务中优先级最高的那个任务，而在本文设计的系统中同一优先级可对应着多个任务，当多个同优先级的任务在等待同一信号量时就按先入先出的方式来分配信号量。信号量中至少要包含如下的一些元素：①信号量标志，其值为 0 或 1，指示资源是否可用或事件是否发生等。②等待此信号量的任务列表。当用信号量对临界资源进行控制时，如果任务申请信号量要进入临界区而无法得到满足时，则进入等待队列，交出处理器的控制权，系统调度其它的进程来运行，这样任务就不会进入盲等而降低了处理器的使用率[29]。③指向正在获得此信号量的指针。

计数信号量与二值信号量相似，计数值在信号量初始化时赋值为系统中该类资源可用的总数目。如果任务需要使用该类资源，就对信号量执行 P 操作，把信号量的计数值减 1，如果减 1 后计数值仍大于 0 或等于 0，就表示资源是可用的，任务可以访问该资源；如果减 1 后计数值小于 0，表示目前该类资源不可用，任务需要被挂起，然后把它放在该信号量的等待队列上等待。当任务完成对资源的使用后，对信号量执行 V 操作，使信号量的计数值加 1，如果加 1 后计数值小于 0 或等于 0，就表明目前有任务正在等待此信号量，然后激活等待队列上的一个任务，使任务使用资源。

为了便于对信号量进行管理，在系统中设计了一个信号量控制块 SCB(Semaphore Control Block)，其结构如下：

```
typedef struct scb{
        UINT8 semtype;   /* 信号量的类型，二值型和计数型*/
        struct scb      *pnext;   /*指向后一个 SCB*/
        UINT8   id;          /*信号量的 ID 号，作为信号量的内部标志，与信
                              号量数组的下标相同*/
        TASK_TCB   *ptskget;  /*指向获得信号量的任务，用于处理优先级
                              继承*/
        TASK_TCB   *ptskwait;  /*正在等待信号量的任务队列*/
```

```
UINT8   name[10];   /*信号量的名称*/
SINT8   maxcount;   /*最大计数值*/
SINT8   count;   /*当前计数值*/
} SCB;
```

3.4.2.2 对信号量的操作

信号量在系统中是一个全局量，通过一个 SCB 类型的数组来存放系统中的所有信号量控制块，数组的下标作为相应信号量控制块的 ID 号[30]，并在系统初始化时把所有的 SCB 连成一个空闲信号量控制块链表，当需要建立信号量时就从链表上取一个空闲的 SCB，同样当删除一个信号量时就把信号量归还到空闲链表上去。系统的信号量数组定义为：

```
SCB scb_array[MAX_SEM_NUM] ;
```

信号量数组初始化连成链表后的结构如图 3.9 所示：

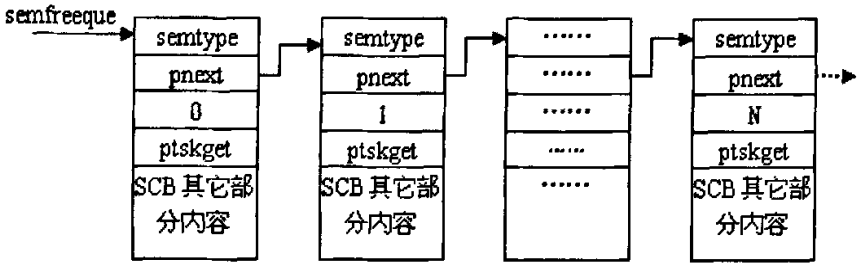


图 3.9 信号量控制块链表

在系统中，通过信号量控制块数组下标或信号量句柄都能找到所需要的信号量，查询到信号量的状态信息、等待该信号量的任务列表等。除在系统启动时把信号量控制块数组初始化成空闲控制块链表的函数 scb_init( )外，系统里还设计了 6 个对信号量进行操作的函数：

```
/*创建信号量，返回指向被创建信号量的指针作为该信号量的句柄*/
SCB *sem_create (UINT8 semtype, UINT8 *pname, UINT8 priority,UINT8 maxcount);
void   sem_wait(SCB *pscb);   /*等待信号量*/
void   sem_signal(SCB *pscb);   /*释放信号量*/
void   sem_delete(SCB *pscb); /*删除信号量*/
void sem_waits(SCB *pscb, … );  /*等待一组信号量*/
```

void sem_signals(SCB *pscb, …)；/*释放一组信号量*/

（1）信号量的建立

在使用一个信号量之前，首先要建立该信号量，对信号量的类型、初始值、等待任务列表等进行初始化，如果创建成功，在程序的最后返回指向该信号量的指针作为该信号量的句柄，以后对信号量的各种操作都通过该句柄进行。信号量建立函数的部分代码为：

```
SCB *sem_create(
    UINT8 semtype,      /*信号量的类型，二值信号量：SEM_BIN、计数信号量：
                          SEM_CNT、事件：SEM_ENT*/
    UINT8 *pname,       /*信号量名*/
    UINT8 maxcount)     /*最大计数值*/
  { SCB *p;
   disable_int( );
   p=semfreeque;   /*指向空闲队列*/
   if (p!=NULL)   /*如果还有空闲 SCB 就调整队列*/
     { semfreeque=semfreeque->pnext;
        enable_int( );}
    else { enable_int( );
          return （SCB *）0; }   /*返回空指针*/
     //初始化 SCB
    p->semtype=semtype;
    p-> ptskget=(TASK_TCB *)0 ;
    p->ptskwait=(TASK_TCB *)0 ;
    Strcpy(p->name,pname); /*把信号量名存放到字符串数组中去*/
    if(p->semtype= =SEM_ENT)
      {   p->maxcount=0;
          p->count=0; } /*为 1 表示事件发生*/
    else if(p->semtype= =SEM_BIN)
```

```
{   p->maxcount=1;
    p->count=1;  } /*表示临界资源可用*/
  else if(p->semtype= =SEM_CNT)
  {  p->maxcount=maxcount;  /*最大计数值*/
     p->count=maxcount;  }    /*当前计数值*/
  return  p;  /*返回指向已创建信号量的指针*/
}
```

（2） 等待和释放信号量

任务在等待一个信号量时，信号量等待函数通过对信号量类型的检测，能够识别信号量是一个互斥、同步信号量还是一个资源计数器。等待一个信号量就是看当前任务要申请的资源是否可用或事件是否发生，如果资源可用，则任务使用资源并把资源计数器减 1，如果资源不可用则把申请信号量的任务挂在等待队列上。等待一互斥信号量和一个计数信号量采用相同的方法，就是首先把信号量里的 count 减 1，减 1 后判断 count 是否小于 0，如果不小于 0 说明资源可用，任务能够继续运行，否则，资源不可用，把任务挂起。申请信号量的函数原型为 Void sem_wait(SCB *pscb) ，其流程图如图 3.10 所示。

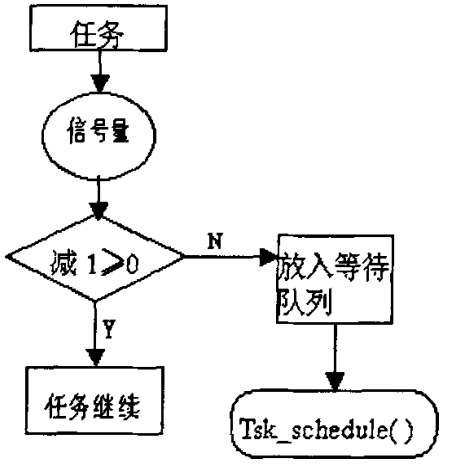


图 3.10 申请信号量流程图

释放信号量的过程与申请信号量的过程相反，在设计上把它们设计成一组相对的操作，其函数原型为 void sem_signal(SCB *pscb)。在释放信号的时候就对计数值执行加 1 的操作，如果信号量小于等于 0 则说明在该信号量的等待队列上有任务正在等待此信号

量，然后把一个任务就绪；如果计数值加 1 后，其值大于 0 则说明没有任务在等待信号量，只需简单的把信号量加 1 即可。信号量里的 maxcount 值能保证信号量的正确性，当加 1 后如果 count>maxcount，则说明肯定在对信号量操作的过程中出错了，给出系统的出错信息。

（3）多信号量的等待与释放

用信号量能够保护那些需要互斥访问的临界资源，但在有些应用场合，任务要运行必需同时获得两个或更多的资源后才能运行，如何让任务同时拥有对两个或两个以上临界资源的使用权呢，如果采用多个 sem_wait( )来逐个的申请互斥信号量，系统将很容易出现死锁的问题[31]，因此在内核里设计了两个对多信号量进行操作的函数 void sem_waits(SCB *pscb, …)和 void sem_signals(SCB *pscb, …)。为了不产生死锁，sem_waits( )在为任务申请多个信号量时，所需要的临界资源一次性全部分配，只要所需要的临界资源中有一个不可用，则其它能用的临界资源也不分配给这个任务，然后把任务挂在第一个不能使用的临界资源信号量的等待队列上。sem_waits( ) 与 sem_wait( )在设计上的不同之处在于，sem_wait( )在检测到信号量不可用时，直接把任务挂在该信号量的等待列表上然后重新调度任务就可以了。而 sem_waits( )在发现有资源不可用时，把任务挂起，当等待的资源可用时还需重新检测所有申请的资源，防止当不可用的资源变为可用时，其它原来可用的资源又不可用了，只有当所有申请的资源都可用时才一次性的给任务分配这些资源。

void sem_waits(SCB *pscb, …)和 void sem_signals(SCB *pscb, …)是两个多信号量等待函数，在调用这两个函数时传入多个信号量指针作为函数的实参[32]，两个函数实参的类型和个数要一致。

3.4.3 邮箱和消息队列

3.4.3.1 邮箱数据结构的设计

操作系统中的邮箱机制与生活中的邮政系统是有着相似之处的，在邮政系统中，一次信件的传递需要两个邮箱：公用邮箱和私用邮箱。发件人把信件投到公用邮箱里，然后邮递员负责把信件从公用邮箱转到收件人的私用邮箱里[33]。

在本文设计的系统里也用了邮箱来完成任务间消息的传递，但与邮政系统比起来它们有两个不同点：1）操作系统的邮箱只能存放一条消息，如果邮箱中已经有消息，在消息被取走前，任务不能向邮箱中存放新的消息，同样，如果邮箱中没有消息，想从邮箱中取消息的任务必需等待。2）在邮政系统里，邮递员只负责把信件从公用邮箱转到收件人的私用邮箱里，并不告诉收件人有信件到了，收件人必需自己去查看，这样收件人为

了收取一封信必需每天都去开邮箱，很浪费时间。操作系统中的邮箱设计成在消息到来时能主动通知等待消息方，当取不到消息时也能主动的通知消息的发送方。

邮箱机制可以使任务间的消息及时得到交换，通信速率非常高，还可以实现消息的暂存、同步和多点传播。本系统设计的邮箱只能存放一条消息，因此如果两个任务对一个邮箱进行操作会出现几种情况，当邮箱为空时发消息的任务就可以直接把消息发到邮箱里，如果邮箱里有消息时，取消息的任务也能马上从邮箱中把消息取走；但是如果邮箱满时发消息的任务就必须挂起等待，直到其它任务把消息取走，发消息的任务才能把消息发送成功，当邮箱为空时取消息的任务也必须挂起进行等待，直到邮箱中有消息才能取消息成功，并返回所取得的消息。邮箱的结构如下：

```
Typedef   Struct mail_box {
            UINT8 mtype;   /*类型*/
            Struct mailbox   *pnext_mbox   /*指向下一个邮箱*/
            UINT8 mboxid;  /*邮箱内部 ID 号，其值在初始化邮箱结点数组时赋
                             值为数组的下标。*/
            UINT8 mcnt;    /*消息数*/
            TASK_TCB    *pwrite; /*邮箱满时，发送消息的任务挂在此
                                   指针指向的队列上等待*/
            TASK_TCB    *pread;  /*邮箱空时，取消息的任务挂在此指针指向
                                   的队列上等待*/
            Void    *pmsg;     /*指向邮箱消息*/
          }MAIL_BOX
```

每次向邮箱写入消息时，如果邮箱为空则把把消息放到邮箱里，否则如果邮箱满就把任务挂在写等待队列上，然后重新调度，选择一个新的任务投入运行，当邮箱的消息被取走后再唤醒任务，把消息存到邮箱里；当任务从邮箱读取消息时，如果邮箱有消息则直接读取消息，否则如果邮箱为空则把任务挂在邮箱的读等待队列上，然后重新调度，选择一个新的任务投入运行，有消息到来时再唤醒任务，把消息读取。

3.4.3.2 消息队列

对消息队列的操作与邮箱相似，消息队列可以存放多条消息，就像一个存放消息的缓冲池，只要队列不空，任务就可以从队列中取消息，同样，只要队列不满，任务就可以向队列发消息。每条消息就是一个记录，消息所代表的含义由通信双方进行约定，通

信的任务可以对消息队列进行读取和写入，消息队列的大小在建立时根据需要进行设定。采用消息队列的通信机制，发送消息的任务不必等待接收任务收取消息就可以继续发送（消息队列未满），接收消息的任务，只要消息队列不空就可以从队列中取走消息，这样能在通信的任务间进行异步的信息传递，当需要交换的数据量很大时，采用这种方式能提高效率。对消息的存取采用 FIFO（先入先出）的方式，取数据的任务从队列头读取数据，发送消息的任务把数据存放在消息队列尾，整个消息队列是一个循环队列。消息队列结点的结构如下所示：

```
Typedef   Struct msg_queue {
                UINT8 msgtype;   /*类型*/
                Struct msgqueue    *pnext_msgqueue;   /*下一个消息队列*/
                UINT8 msgqid;   /*邮箱队列的内部 ID 号*/
                UINT8 queuesize;   /*消息队列最大能存放的消息数*/
                UINT8 msgcnt;    /*队列中的消息数*/
                TASK_TCB    *pwrite;   /*队列满时，写消息的任务挂在此指针指向
                                        的队列上等待*/
                TASK_TCB    *pread; */队列空时，读消息的任务挂在此指针指向的
                                        队列上进行等待*/
                Void    **phead;   /*指向消息队列头的指针*/
                Void    **pout;    /*指向下一条将要被读取的消息*/
                Void    **pin;     /*指向下一个存放消息的空间*/
                Void    **pend;   /*指向消息队列尾*/
              }MSG_QUEUE
```

系统的消息存储区为一个指针数组，数组里的每个指针可以指向一条消息，数组的大小根据需要进行指定，如有 Void *msgbuf[10] 则此消息队列最大可以存放 10 条消息，消息队列结点与消息存储区的关系如图 3.11 所示。

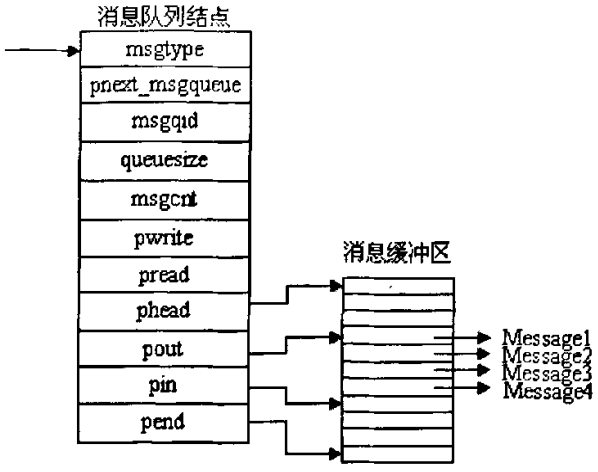


图 3.11 消息队列结点与消息存储区关系图

3.5 任务优先级继承的设计

3.5.1 优先级继承

在嵌入式系统的应用中，实时性是一个重要的指标，而优先级反转会影响系统的实时性，使得高优先级任务的延迟时间无法预测。通过前面优先级反转问题及其解决方法的研究知道，优先级继承和优先级置顶都可以解决优先级反转问题，他们各有优缺点。优先级置顶的方法只需改变一次优先级，效率高，对于任务可能被多次阻塞的情况性能会得到很好的改善，但要在一个大的系统中事先确定出试图获取某一共享资源的任务中的最高优先级往往很困难。优先级继承虽然有可能会多次改变任务的优先级，增加系统的开销，但它只有在需要时才改变，而不需要预先知道某共享资源的顶层优先级，这对于阻塞情况不是很多的任务也能收到很好的效果，而且实现起来相对比较容易。在本文设计的系统中采用的是优先级继承的方法。

优先级继承就是当一个高优先级的任务要访问已被低优先级任务占有的临界资源时，系统就会把占用临界资源的任务的优先级提高，使其尽快的释放临界资源，当它释放资源后再把其优先级还原到本来的水平。

3.5.2 对共享临界资源的控制

优先级反转问题的产生归根到底是因为多个不同优先级的任务需要对同一共享临界资源进行访问而引起的，因此，要从对共享临界资源的访问控制上来解决优先级反转问题。在对需要互斥访问的临界资源进行保护时可以采用关中断/开中断或信号量的方式来处理，开/关中断的方式不会引起优先级反转问题，但是，如果对临界资源的访问需要的时间很长则不宜采用关中断的方式，因为关中断的时候会阻止系统对外部事件的响应，

从而降低了系统的实时性，因此只能采用互斥信号量的方式来进行保护。

信号量控制块（SCB）的结构在任务间的通信与同步部分已给出，除了信号量的类型、信号量控制块的ID号、最大计数值、当前计数值、信号量名称等域外还有ptskwait和ptskget指针分别指向等待该信号量的任务队列和正在使用该信号量的任务。保护共享资源的二值信号量(semtype为SEM_BIN)与访问共享资源的任务的关系如图3.12所示。

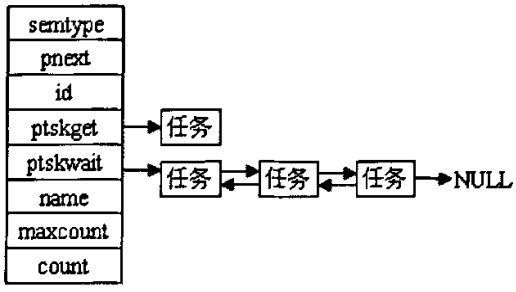


图3.12 二值信号量与任务队列

在使用信号量前，首先用 sem_create（）创建一个二值信号量，如果信号量创建成功则返回指向被创建信号量的指针作为该信号量的句柄。当访问临界资源时先对该句柄执行sem_wait( )操作，sem_wait( )操作判断信号量是否可用，如果信号量可用，即count减1后是否大于等于0 ，count 可能为1、0和负数三种情况，1表示任务可以获得信号量，其它表示任务不能马上获得信号量，需挂起等待，如果大于等于0则让ptskget指针指向此任务，否则说明信号量不可用，把任务挂在该信号量的等待队列上，同时把当前任务的优先级与已获得信号量的任务即 ptskget 所指向的任务的优先级进行比较，如果ptskget 所指向的任务的优先级比当前任务的优先级低，则把它提高到当前任务的优先级，重新调用tsk_schedule( )进行任务调度。sem_wait( )函数的部分代码如下：

```
void sem_wait( SCB *pscb)
   {  disable_int( ) ; //关中断
      if((- -pscb->count)>=0)   //任务获取信号量
       { pscb-> ptskget=task_current_pt;
         enable_int( );
        return 0; }
      else { if(pscb->ptskwait= =( TASK_TCB * ) 0)   /*等待队列*/
              pscb-> ptskwait = task_current_pt;
           else{   task_current_pt->task_next= pscb-> ptskwait; /*调整队列*/
                   pscb-> ptskwait->task_previous= task_current_pt;
```

```
            pscb-> ptskwait= task_current_pt;   }
        /*优先级值越大其优先级越小*/
        if (pscb->ptskget->task_priority > task_current_pt->task_priority)
            {   inheritprio(pscb);   /*继承优先级*/
                enable_int( );
                tsk_schedule( );   }
         else { enable_int( );
                tsk_schedule( ); }
        }
}
```

在完成对共享临界资源的使用后，用 sem_signal( )来释放信号量，如果任务被提升过优先级，就恢复到原来的优先级。在信号量的等待队列上如果没有任务在等待，就简单的把信号量的计数值 count 加 1，否则，把信号量给等待队列中优先级最高的那个任务，当多个任务的优先级相同时，则给先等待的那个任务。

3.6 时钟管理模块的设计

系统中所有与时间相关的操作都通过时钟管理模块来进行管理，对时钟的管理关系到系统的实时性、任务调度和任务运行的精度等问题。系统时钟按其应用不同分为绝对时钟和相对时钟两种，绝对时钟用于提供时间信息和各种事件发生的时刻记录[34]，相对时钟主要用于任务的启动和停止等操作。

系统里的任务启动后就会处于运行状态、就绪状态和等待状态这些状态中的一个，如果处在运行状态的任务想延时一段时间，就需要把其从运行状态转入到等待状态，并放到等待队列上，当延时时间到后再把其转为运行态或就绪态，这都需要用到系统的时钟管理功能。

为实现延时和定时功能，要求系统的硬件能够产生定时中断，通过定时中断来产生系统的时钟节拍，时钟节拍的频率越高系统的精度越高，但负荷也会越重。当时钟中断节拍到来时，系统里的时钟中断服务程序就会响应中断，并做一些必要的处理，在中断服务其间，系统把中断关闭掉，为了保证系统的实时性，必须把中断服务程序设计得尽可能简短，把复杂的功能实现放到相应的任务里去处理，而不直接在时钟中断服务程序里作处理。通过时钟延时程序能够把当前的任务延时一定量的时钟节拍（Ticks），所有被

延时的任务都挂在延时队列上，每个时钟节拍到来时都要对些队列上的任务的延时值减1，当任务的延时值被减为0时就把任务重新置为就绪状态，并把其从延时队列中删除，挂相应的就绪队列上。

3.7 中断管理模块的设计

中断是一种硬件机制，是对内部、外部事件做出的一种响应，用于通知CPU有异步事件发生，一般由外设或机器指令触发，不受调度程序支配。中断一旦被系统识别，系统就跳到相应的中断入口处，保存部分或全部现场(context)，然后执行中断服务程序(ISR)。中断服务程序对事件作必要的处理，处理完成后再执行任务调度，如果中断服务使比被中断的任务优先级更高的任务就绪了，则系统就进行任务切换，否则返回到被中断了的任务[48]。

采用中断的方式使得CPU可以在事件发生时才予以处理，而不必让微处理器连续不断地查询是否有事件发生，从而提高处理效率。系统也可以禁止中断，这样系统就不响应中断了，对操作系统中一些需要具有原子性的操作，就必须用关闭中断的方式来实现，但是，关闭中断的时间过长会影响系统的实时性，所以在实时系统中，关闭中断的时间要尽可能的短。中断关闭的时间往往是一个实时操作系统的重要性能指标，因此，内核的中断处理程序的设计颇为重要，中断处理程序应该写得短小，仅对中断做做一些简单和必要的处理，更复杂的功能交给相应的任务去完成，中断处理的过程如图3.13所示：

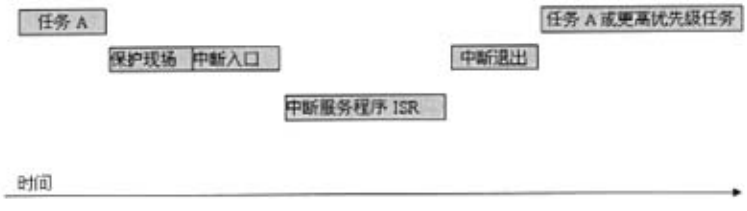
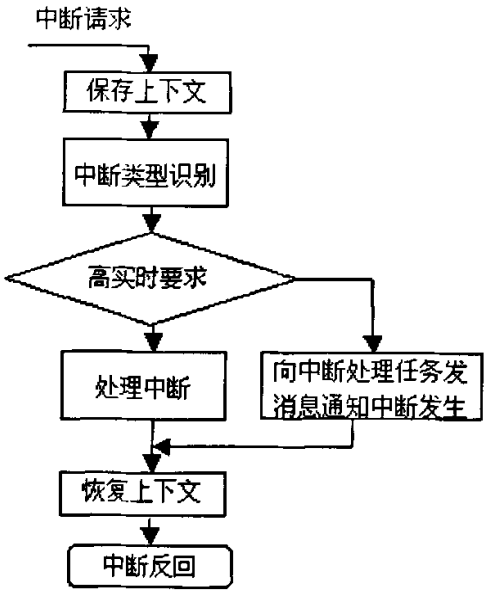


图3.13 中断处理过程

本系统在设计时为了提高对有高实时性要求的中断处理的响应，中断服务程序（ISR）只是负责在最小的时间内通知系统中断的产生，除非是异常紧急的一些中断请求才放在中断服务程序中处理，否则，将中断引起的不要求实时处理功能放在相应的任务中实现，这使得系统对中断的反应速度能得到及大的提高。系统通知中断处理的任务是通过在中断服务程序里向中断处理的任务发送消息来实现的，中断处理任务在收到消息后，将在适当的时候完成对中断的处理，中断响应的流程如图3.14所示。



3.14 中断响应的流程图

3.8 内存管理的设计

与桌面操作系统相比，嵌入式操作系统的内存管理有着自身的特点。首先，嵌入式系统受成本、体积等因素的限制，内存容量非常小，需要高效的使用。其次，嵌入式系统一般对实时性有所要求，因此，内存的分配和回收必须迅速，并且内存分配的时间应该具有确定性。内存的管理策略可以分为静态分配和动态分配两大类[47]，对于静态分配策略，内存空间的大小在编译时就已经确定，在系统初始化的时候分配好固定数量的内存来存放需要使用的对象和数据，任务在运行过程中使用这些内存，即使任务运行结束，这部分内存也不会被回收。对于动态分配策略，内存是在任务运行的时候根据需要向系统申请分配的，任务运行结束或使用完内存后把其归还给系统[41]。

在本文所设计的系统中，采用了二元伙伴算法来对内存进行管理，把除系统内存和堆栈之外的内存空间划分为 N 个区，每个区中内存块的大小为 2^nKB，并且把分区内的内存块连成一个双向链表[42][43][44][45]。内存的分区控制块 MEMEY_PARTITION_CB 和内存块控制块 MEMERY_BLOCK_CB 的数据结构分别如下：

```
Typedef struct memery_partition_cb{
        MEMERY_BLOCK_CB *first_block_pt ; /*分区中的第一个内存块*/
        UINT32 *start_addr_pt;        /*分区起始地址*/
```

```
    UINT32 *end_addr_pt;        /*分区结束地址*/
    UINT16 total_num;       /*总内存块数*/
    UINT16 free_num;        /*空闲内存块数*/
    UINT16 block_size;      /*分区中内存块的大小*/
    }MEMEY_PARTITION_CB
Typedef struct memery_block_cb{
    MEMEY_BLOCK_CB *block_pre_pt;     /*指向前一内存块*/
    MEMEY_BLOCK_CB *block_next_pt;        /*指向后一内存块*/
    UINT16 size;        /*分区中内存块的大小，与 block_size 同*/
    UINT16 current_size;        /*内存块当前大小*/
    UINT32 *start_addr_pt;  /*内存块起始地址*/
    UINT32 *end_addr_pt;        /*内存块结束地址*/
    UINT8 state;    /*内存块的占用情况*/
    UINT16 task_id;     /*使用该内存块的任务号*/
    }MEMEY_BLOCK_CB
```

伙伴算法的内存管理策略实现简单，能高效的分配和回收内存块，从而很好的满足嵌入式系统中对实时性有要求的应用。分区和内存块的关系结构如图 3.15 所示。

伙伴算法的思想很简单，就是在分配内存时，系统根据申请内存块的大小，查找最小且有能满足要求的空闲内存块的分区链表。假设任务需要的内存空间为 1KB，系统首先从 0 号分区中查找，如果有空闲内存块，就从链表头取一内存块分配出去，否则在 1 号从区中查找，如果找到空闲块，则把空闲块分成相等的两部分，前面的 1K 挂到 0 号分区的链表头部，后面 1K 分配给任务。如果 1 号分区中仍然没有空闲块，则从 2 号分区中进行查找，如果找到，则把内存块进行划分，先把 4K 的空间分成两个 2K，前面的 2K 挂到 1 号分区上，后面的 2K 再进行划分，前 1K 挂到 0 号分区上，后 1K 分配给任务，如果 2 号分区也没有空闲块，则重复前面步骤，直到所有分区都没有符合要求的空闲块，则把任务挂起。

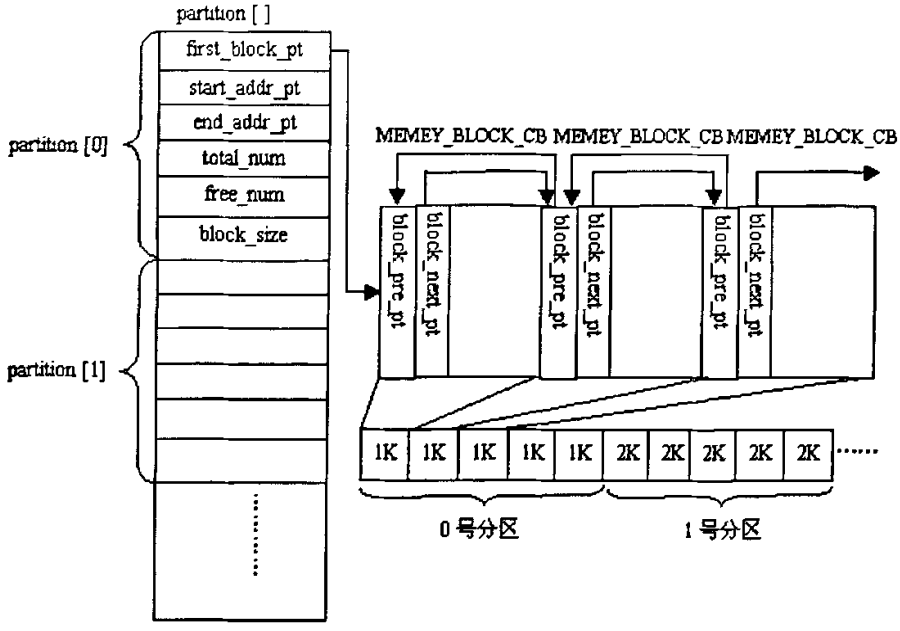


图 3.15 分区和内存块的关系图

释放内存块时，先查看内存块当前所属链表上是否有伙伴存在，如果没有伙伴存在就直接把要释放的内存块挂在链表头；如果有伙伴内存块存在，则把伙伴块从链表中取下来，合并成一个更大的内存块，然后用合并后的内存块在下一个分区中找伙伴块，如果找到就继续合并，直到没有伙伴块可合并为止。两个内存块要成为伙伴块，必须满足三个条件：（1）两个内存块的大小相同。（2）两个内存块的地址连续。（3）两个内存块是从同一分区中分离出来的。

第 4 章 嵌入式操作系统系统在 ARM 平台上的实现

随着嵌入式应用系统对实时多任务、高处理能力、网络通信等应用要求的提高，传统的 8 位处理器已逐渐不能满足要求，高性能低功耗的 16/32 位 ARM 处理器随着价格的下降，得到了广泛的应用，目前，基于 ARM 技术的微处理器应用约占据了 32 位 RISC 微处理器 75%以上的市场份额。

4.1 硬件平台介绍

4.1.1 EL-ARM-830 硬件平台

EL-ARM-83 硬件平台是一个功能强大、使用方便的开发设计平台，可以方便的在平台上进行软件开发和硬件设计。平台上的 S3C2410A 处理器采用的是 ARM920T 处理器核，ARM920T 在 ARM9TDMI 的基础上增加了指令和数据 Cache，指令和数据端口通过 AMBA 总线主控单元合并在一起。EL-ARM-830 平台有着丰富的外围扩展资源，如 320×240 的 256 色 LCD 液晶显示屏、4×4 的 16 键小键盘、UART 串口、数字量输入输出接口、数模信号发生器、语音编码器和解码器、以太网接口、USB 接口等，平台的功能框图如图 4.1 所示。

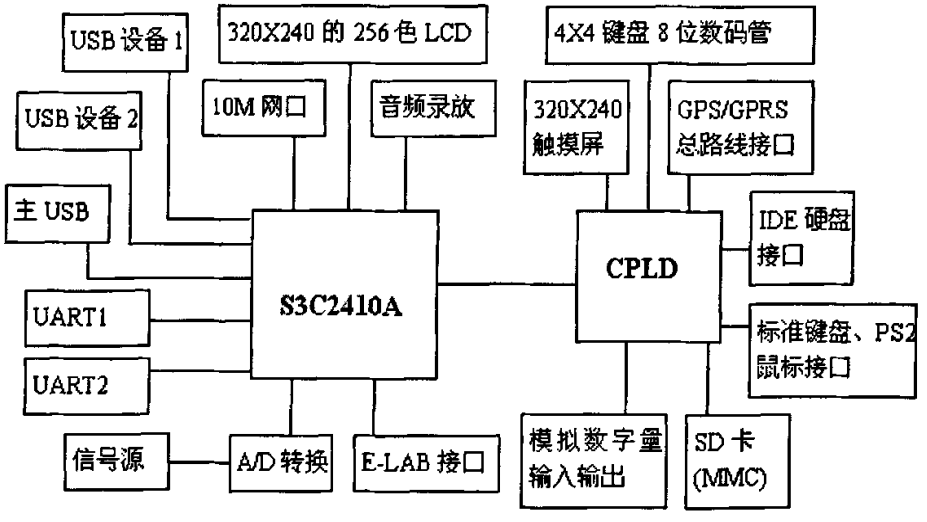


图 4.1 EL-ARM-830 功能框图

4.1.2 S3C2410A 处理器

S3C2410A 处理器采用的是 ARM 公司设计的 ARM920T 的处理器核[35]。ARM（Advanced RISC Machines）既可以认为是一个公司的名字，也可以认为是对一类微处理器的通称，还可以认为是一种技术的名字。ARM 公司于 1991 年在英国剑桥成立，主要靠出售芯片设计技术的授权盈利，不直接生产处理器芯片。各大半导体生产商从 ARM 公司购买其设计的 ARM 微处理器核，根据不同的应用领域，加入适当的外围电路，从而形成各具特色的 ARM 微处理器芯片进入市场。

处理器共有 7 种运行模式[36]，除用户模式外的其它 6 种模式都是特权模式，这 7 种模式为：

处理器模式	用　途	CPSR[4：0] 内容
用户模式(User)	正常程序执行的模式	10000
快速中断模式（FIQ）	用于高速数据传输和通道处理	10001
IRQ	处理标准中断	10010
SVC	处理软件中断	10011
中止	处理存储器故障	10111
未定义指令	未定义指令陷井	11011
系统	运行特权操作系统任务	11111

除系统模式外的每一种特权模式都有一个与之相关的程序状态保存寄存器 SPSR 用于保存进入特权模式时 CPSR 的状态，以便当重新开始用户进程时能全部恢复用户状态。处理器总共有 37 个 32 位的寄存器，其中 31 个为通用寄存器，6 个为专用寄存器。R0~R7 是未分组寄存器，同一个寄存器名在处理器的内部只有一个独立的物理寄存器与它对应；R8~R14 是分组寄存器，同一寄存器名，由于处理器的运行模式不同会对应着不同的物理存储器，其中的 R13 又叫作 SP，用于存放堆栈指针，R14 又叫 LR，用于保存返回地址，这两个寄存器在每种异常模式下都对应到不同的物理寄存器上。R15 也就是程序计数器（PC）和 CPSR 为所有模式下共用。

4.1.3 处理器的中断系统

当异常中断发生时，系统在执行完当前的指令后就会跳到中断入口处执行中断处理程序，执行完中断处理后再返回到被中断的指令。在处理器的低地址处有一个 32 字节的中断向量表，共有 8 个中断向量，每个中断向量占 4 个字节，由于中断向量空间比较小，不能直接存放中断处理程序，只能用来存放中断处理程序的入口地址，而把中断处理程序放在中断向量表之后的地址里。不同类型的中断其中断向量地址不同，如复位中断的

中断向量地址为 0x0、预取指中断的向量地址为 0x0C。当一个异常中断出现后，处理器就会执行以下一些操作[37]：（1）保存处理器当前状态字、中断屏蔽位以及各条件标志位。（2）设置当前程序状态寄存器中的相应位，使其为相应的中断模式。（3）将相应模式下的 LR 寄存器设置成中断的返回地址。(4)将程序计数器 PC 设置成中断服务程序的地址，从而使程序跳到中断服务程序处进行中断处理。

中断在对外部事件的处理中起着重要的作用，每一个中断发生时系统总是从中断向量处开始起跳，如系统的启动就是通过一个复位中断来实现的，当按系统的电源键或复位键时，系统就跳到复位中断向量（0X0）处开始执行程序。在系统的启动文件里，通过以下的代码来完成对中断向量表的设置：

```
AREA Init, CODE, READONLY
    ENTRY
      B Reset_Handler
      B Undef_Handler
      B SWI_Handler
      B PrefAbt_Handler
      B DataAbt_Handler
      B.
      B IRQ_Handler
      B FIQ_Handler
    END
```

为了提高编程效率，只用汇编写简单的中断服务程序（ISR），用于保存中断时的上下文和中断处理完成后恢复上下文，真正的中断处理程序用 C 语言来编写，下面是 IRQ 的中断服务程序：

```
IRQ_Handler
        STMFD sp!, {r0-r12,LR}    //保护现场.
        BL HandleIRQ              //进入 C 语言编写的中断处理程序
        LDMFD sp!, {r0, pc}       //恢复现场
        SUBS   PC，LR，#4         //中断返回，需要调整连接寄存器 LR 的//内容作
                                  为返回地址。
```

4.2 开发环境的建立

嵌入式系统及其软件的开发与 windows 等桌面系统下的应用程序开发不同，它首先需要建立一个交叉编译环境，包括编译器、连接器和 libc 等库文件，然后在宿主机上生成能在目标机上运行的代码。

ADS 是由 ARM 公司自己设计的集成开发环境，其编译效率很高，支持各种 ARM 内核，包括有编辑器、编译器、链接器、调试器等工具。使用 ADS 里的 AxD 调试器，通过 JTAG 接口就可以在线调试程序，极大的方便了程序的开发。

在启动 ADS 后新建一个工程，把操作系统内核文件加入进去，如图 4.2 所示。然后对各选项进行设置，由于平台使用的 S3C2410A 处理器采用的是 ARM920T 核，因此在处理器内核选择一栏里选择 ARM920T，程序的入口地址 RO Base 设置为 0x30000000。

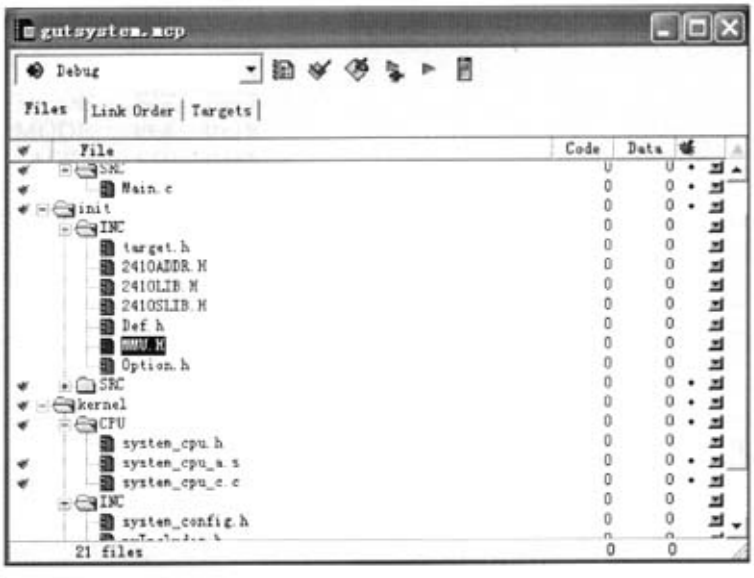


图 4.2 开发环境的建立

4.3 系统初始化代码的设计

系统的初始化分为两部分，一是对硬件平台的初始化，二是对操作系统的初始化。在操作系统内核启动前，需要对部分硬件进行初始化，如对存储器地址的映射等，为系统的启动带入一个合适的环境。在系统启动文件里通过以下几步来对硬件进行初始化：

（1）设置中断向量表。系统在启动或复位后都是从 0X0 地址开始取址执行程序代码的，并且从 0X0 开始到 0X1C 的 32 个字节是系统的中断向量表空间，要在这里存放

中断服务程序的入口地址。

（2）关闭看门狗，禁止所有的中断，设置时钟模块。

（3）初始化存储器。S3C2410A 处理器支持多种类型的存储器，如 ROM、Flash、SDRAM、SRAM 等，在启动时配置各类存储器的容量、总线宽度、时序和存储空间地址映射等。

（4）初始化各工作模式的堆栈。处理器总共有 7 种工作模式，每种工作模式下都有一部分分组寄存器，各模式下的堆栈指针寄存器 SP 也是不同的，因此需要为各模式下的堆栈指针寄存器指定一个地址值，并为该模式下的堆栈分配一定的堆栈空间。各堆栈设置的代码如下：

```
;各种模式的常量定义
USERMODE    EQU  0x10
FIQMODE     EQU  0x11
IRQMODE     EQU  0x12
SVCMODE     EQU  0x13
ABORTMODE   EQU  0x17
UNDEFMODE   EQU  0x1b
MODEMASK    EQU  0x1f
NOINT       EQU  0xc0
STACK_BASEADDRESS   EQU   0x33ff8000   ;所有堆栈的起始地址

UserStack     EQU (STACK_BASEADDRESS-0x3800)  ;//0x33ff4800～
SVCStack      EQU (STACK_BASEADDRESS-0x2800)  ;//0x33ff5800～
UndefStack    EQU (STACK_BASEADDRESS-0x2400)  ;//0x33ff5c00～
AbortStack    EQU (STACK_BASEADDRESS-0x2000)  ;//0x33ff6000～
IRQStack      EQU (STACK_BASEADDRESS-0x1000)  ;//0x33ff7000～
FIQStack      EQU (STACK_BASEADDRESS-0x0)     ;//0x33ff8000～

initstacks
        mrs r0,cpsr          ;关中断
        orr r0,0xc0
        msr cpsr,r0

    bic  r0,r0,#MODEMASK
    orr  r1,r0,#UNDEFMODE
    msr cpsr_cxsf,r1
    ldr  sp,=UndefStack

    orr  r1,r0,#ABORTMODE
    msr cpsr_cxsf,r1
    ldr  sp,=AbortStack

    orr  r1,r0,#IRQMODE
    msr cpsr_cxsf,r1
    ldr  sp,=IRQStack
```

```
orr  r1,r0,#FIQMODE
msr cpsr_cxsf,r1
ldr  sp,=FIQStack

bic  r0,r0,#MODEMASK
orr  r1,r0,#SVCMODE
msr cpsr_cxsf,r1
mov     pc,lr
```

（5）初始化数据存储区，并切换到用户模式。

在完成以上部分的硬件初始化后，进入操作系统前还需要对各外设部件进行初始化，如 LCD 显示屏、键盘、串口等。各外设的初始化程序都在对应外设的驱动程序里，通过在 target_init( )函数里调用各外设初始化函数来完成对外部设备的初始化。

当硬件和各外部设备的初始化工作都完成后，就进入到操作系统的初始化，操作系统的初始化主要完成对一些系统公共变量赋初值，如任务数、优先级表、各任务队列等；把一些控制块连成链表，如空闲任务控制块 TASK_TCB、信号量控制块 SCB 等；在操作系统初始化的最后创建一个空闲任务，其优先级设置为 MAX_TASK_LEVEL-1，ID 号设置为 MAX_TASK_NUM-1，这样能保证系统在进行多任务调度时至少有一个任务可调度，这是一个优先级最低，ID 号最大的任务，在系统运行的整个过程中不能被删除。操作系统初始化的所有工作通过调用 tsk_system_init( )函数来完成。

4.4 处理器相关的部分代码

系统在设计的时候就考虑了移植性的问题，因此绝大部分的代码都是采用标准 C 来进行设计和实现的，只有极少一部分代码是采用汇编语言来编写，这样在针对具体的硬件平台时只需要重新改写这部分汇编代码即可。根据编译器文档和处理器手册，对系统用到的数据类型定义如下：

```
Typedef unsigned char UINT8;

Typedef signed char SINT8;

Typedef unsigned short UINT16;

Typedef signed short SINT16;

Typedef unsigned int UINT32;

Typedef signed int SINT 32;

typedef unsigned int TASK_STACK;

typedef unsigned char BOOLEAN;
```

系统在进行任务切换的时候需要对任务的上下文保存和恢复，这部分的内容需要直接访问寄存器，因此要用汇编语言来编写。 程序的关键代码如下：

```
context_swin
        ldr r4,= task_current_pt      ；将要切换进的任务的指针，经过调整
        ldr r4,[r4]                   ；任务堆栈的指针
        ldr sp,[r4]
        ldmfd sp!,{r4}                ；恢复任务的 SPSR
        msr SPSR_cxsf,r4
        ldmfd sp!,{r4}                ；恢复任务的 CPSR
        msr CPSR_cxsf,r4
        ldmfd sp!,{r0-r12,lr,pc}      ；恢复其它寄存器的内容，任务继续
context_sout
        ldr r4,= task_current_pt      ；将要切换出的任务的指针
        ldr r4,[r4]                   ；任务堆栈的指针
        ldr sp,[r4]
        stmfd sp!,{lr}                ；保存任务被中断处的地址
        stmfd sp!,{r0~r12,lr}         ；保存其它寄存器内容
        mrs r4,cpsr
        stmfd sp!,{r4}                ；保存 CPSR 内容
        mrs r4,spsr
        stmfd sp!,{r4}                ；保存 SPSR 的内容
```

在处理临界资源时，有时需要关中断，在完成对临界资源的处理后又需要开中断，在 ARM 的 CPSR 寄存器里有两位负责 IRQ 和 FIQ 中断的开/关，可以通过汇编指令操作寄存器里相应的位来开/关中断，程序的关键代码如下：

```
disable_int                           ；关中断
        Mrs r0,cpsr
        Orr r0,r0,#0xc0
        Msr cpsr,r0
        Mov pc,lr
```

```
enable_int                              ；开中断
        Mrs r0,cpsr
        Bic r0,#0xc0
        Msr cpsr,r0
        Mov pc,lr
```

4.5 系统的多任务调度

4.5.1 同优先级任务的调度

在系统里设置了 3 个任务 TASK_A、TASK_B 和 TASK_C，给这三个任务的优先级都为 10，任务 id 号分别为 10、11 和 12，TASK_A 的部分代码如下：

```
void    TASK_A( )
{
        ……
    while(1)
      {
       dispstring("TASK_A is running    AAAAA \n");
       ……
       delay(1000); }}
```

然后通过 tsk_create(TASK_A,task_a_stack[STACKSIZE-1],10,10,TASK_A) 创建任务 TASK_A，任务 TASK_B 和 TASK_C 的结构和创建方式与 TASK_A 相同，系统运行的结果如图 4.3 所示。

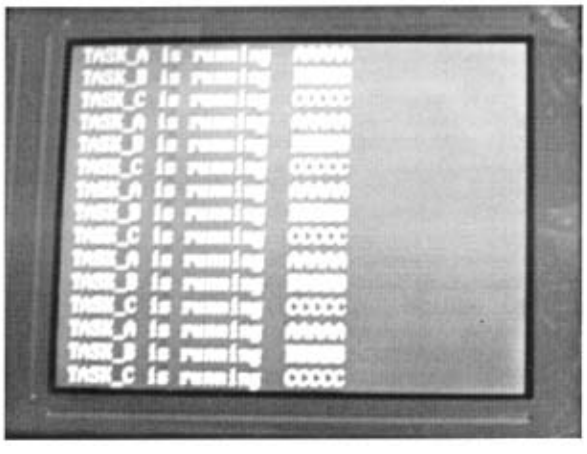


图 4.3 时间片分时调度

4.5.2 抢占式调度

在系统里增加了键盘处理任务 TASK_KEY，把键盘处理任务的优先级设置为 9，任务 id 号为 9，由于其优先级比 TASK_A、TASK_B 和 TASK_C 都要高，所以只要 TASK_KEY 就绪，就会马上得到执行。平台上键盘采用的是中断方式，当有键按下时，系统会产生一个中断，在这里，中断服务程序并不对按键做太多的处理，只是向保存按键的邮箱发送消息，等待消息的键盘处理任务 TASK_KEY 在收到消息后就会被置为就绪，然后对按键进行处理。键盘处理任务的部分代码如下：

```
void TASK_KEY( )
    {  char  key;
      char  *key_pt;
    while(1){
          key_pt=mbox_wait( keymbox);
          key=*key_pt;
          switch(key)
          {case 0x01:
                    dispstring("the key 1 was pressed \n");
                    ……
                    break;
              case 0x02:
                     dispstring("the key 2 was pressed \n");
                     ……
                     break;
               case 0x03:
                      ……
            }
      }
    }
```

系统运行的结果如图 4.4 所示。

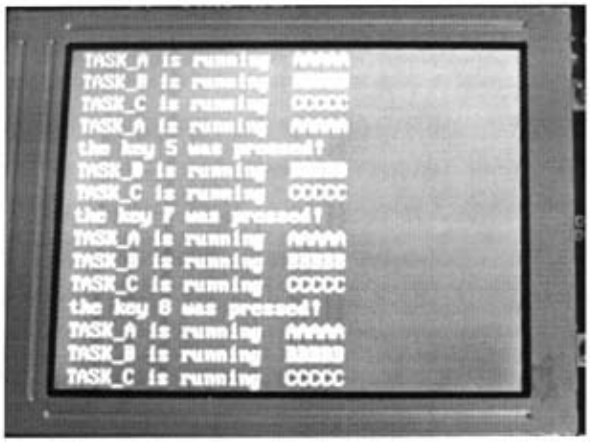


图 4.4 高优先级任务抢占低优先级任务

4.5.3 图形演示键盘输入

在基于优先级的系统中，处理显示工作的任务常常设置为很低的优先级，这样就不会影响对数据的处理，在这里设计了一个用于显示按盘按下情况的任务，通过在键盘处理任务里向其发送按键消息，按键显示任务在没有其它高优先级任务运行的时候就会运行，显示出键盘的各按键状态，如图 4.5 所示。

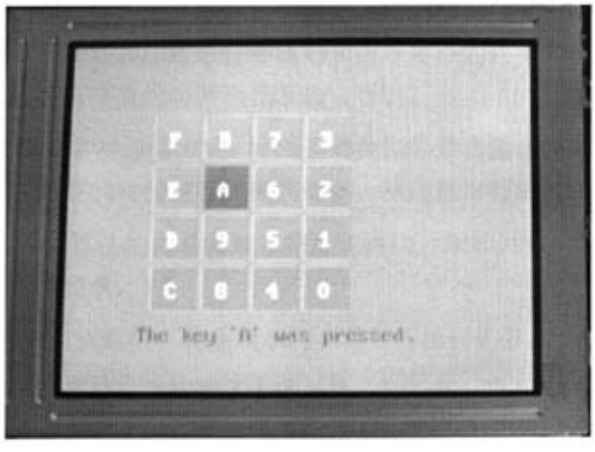


图 4.5 图形演示键盘输入

第5章 总结与展望

本文详细的论述了作者在嵌入式操作系统微内核的研究与设计方面所做的工作，通过对嵌入式操作系统特别是微内核结构的操作系统的深入探讨和研究，分析了一些系统内核的源代码，如 uClinux、RTOS51、uC/OSII 等，特别对 uC/OSII 进行了深入的研究，吸取了其设计思想和微内核理念，设计了一个嵌入式操作系统内核，实现了多任务的调度、任务间的通信、时钟管理、内存管理等基本功能。系统内核在任务调度方面采用基于优先级和时间片相结合的调度算法，能根据任务对时间要求的严格程度不同赋予不同的优先级，可很好的满足嵌入式应用中的实时性要求。在进行任务调度时，通过预先设计好的查找表，能快速的找到所要调度的任务，速度快、效率高。对控制临界资源访问的信号量进行了特别的设计，能够克服任务调度过程中出现的优先级反转问题，使系统更加稳定。本系统是作者在参与广西区科技攻关项目“基于信息家电的嵌入式操作系统内核研制”的研发期间所设计的。

概括的讲，本文所设计的系统具有如下的一些特点：

（1）内核在调度策略方面吸取了抢占式调度和分时调度的优点，从而弥补了单独使用这两种调度策略的不足，能很好的满足那些对实时性有要求的嵌入式系统的应用，同时也方便了应用程序的开发。

（2）基于查表法的优先级队列管理，使得系统在进行任务调度时能快速的找到所要调度的任务，调度过程所需要的时间与系统当前的任务数无关，时间复杂度为 O(1)，而不像 Linux 等系统需要遍历就绪任务队列和进行复杂的任务权值计算，当就绪任务数不确定时，查找任务所需要的时间也是不确定[42]。

（3）内核提供多种通信方式，特别是对控制临界资源访问的信号量进行了特别的设计，便于任务在必要的时候进行优先级继承，从而能很好的处理优先级反转问题。

（4）二元伙伴算法的内存管理策略能快速的分配和回收内存，减少内存碎片，很好的满足嵌入式应用要求。

（5）采用微内核结构，使系统小巧、易于移植和扩展。

虽然所设计的系统内核实现了上述功能，但仍有一些不足，需要在今后逐步的完善。本文只是设计和实现了嵌入式操作系统的内核，尚有很多的系统服务功能需要添加，如，文件系统、网络协议栈、图形操作界面等，这些也都还需要大量的研究工作。嵌入式系统的应用是广泛而多样的，而嵌入式操作系统又是嵌入式系统中的灵魂，其性能决定着整个系统的性能，微内核结构的操作系统与传统的单内核操作系统比起来有很多优良的特点，如，系统小巧、易于移植和功能扩展、操作灵活等，能很好的满足嵌入式系统应用复杂多样性的需求，将成为嵌入式领域的重要研究方向。

致　谢

这篇论文能够顺利完成，首先要感谢我的导师程小辉教授。

非常荣幸能够在程老师的指导下完成硕士学业。三年的硕士学习期间，程老师给予了我悉心的指导和无微不至的关怀，程老师以他渊博的知识、积极的生活态度、严谨的工作作风熏陶着我。在论文的选题、材料的组织以及课题研究等方面给了我精心的指导，在程老师的身上，我学到了很多的东西，再次对程老师表示深深的感谢！

感谢同一项目组的牛秦洲教授、刘电霆副教授、蒋存波副教授，正是有了他们的关怀和指导，为我创造了一个良好的学习和科研究环境，才使得我的毕业论文能够顺利的完成。感谢陈冬旭、周华茂、郑安兵、王丽霞、邓建志、刘政等同学，他们在我论文完成期间提出了很多宝贵的建议，感谢与我一个实验室的邹明亮老师、陆秋老师、刘亚荣老师以及陈德美、陈宫、曾超等众多的兄弟姐妹们。感谢他们在我研究生学习生活中给予我的帮助和支持，我们是一个团结的集体，与他们一起生活、奋斗的日子是令人难以忘怀的。

最后我要感谢我的父母及其这么多年来对我风雨无阻支持的亲人和朋友，对他们的感激之情无法用言语来表达。

参考文献

[1] 蒋静,徐志伟.操作系统原理技术与编程[M].北京：机械工业出版社，2004

[2] http://www.e-works.net.cn/Articles/452/Article36683_1.htm

[3] William Stallings 著,魏迎,王涌等译. 操作系统内核与设计原理[M].电子工业出版社,2003

[4] 毛卫良,成盛烨,郝琴等.基于微内核的嵌入式实时 OS 设计[J].微型电脑应用,2000.1

[5] http://www2.ccw.com.cn/1996/23/142274.shtml

[6] 徐昀.基于嵌入式实时操作系统的通讯管理机的研制[D].河海大学,2004

[7] 陈明计,周立功等著.嵌入式实时操作系统 Small RTOS51 原理及应用[M].北京航空航天大学出版社,2004

[8] 石高涛.嵌入式操作系统微内核中进程管理的研究[D].哈尔滨工业大学,2002

[9] 郑竑宇,宋寅卯.单片机占先式实时微内核的设计[J].航空兵器,2003.2

[10] Jeffay K,Martel CU.On non-preemptive scheduling of periodic and sporadic tasks.Proceedings of the 12th IEEE real-time systems symposium.Sna Antonio, Texas:IEEE Computer Society Press,1991

[11] Georges L,Muehlethaler P, Rivierre N.A few results on non-preemptive real-time scheduling.INRIA Research Report nRR3926,2000

[12] Reddy ALN, Wyllie J.Disk Scheduling in Multimedia I/O system.In Proceedings of ACM multimedia' 93,Anaheim,CA,August 1993

[13] Wenming Li,Krishna Kavi,Robert Akl.A non-preemptive scheduling algorithm for soft real-time systems.Computers and Electrical Enginerring,2006

[14] 褚亚铭.一个教学用微内核操作系统的设计与实现[D].苏州大学,2005

[15] 杨静,戴华平.在 uc/os II 中消除优先级反转[J].计算机工程与应用,2005.7

[16] L Sha, R Rajkumar, J P Lehoczky.Priority inheritance protocols: An approach to real-time synchronization[J].IEEE Transactions on Computers,1990:39(9)

[17] Jim Abu-Ras. Priority inheritance protocol in Ada 95.Information and Software Technology 38(1996)

[18] Kam-yiu Lam,Joseph Kee-Yin Ng.A conditional abortable priority ceiling protocol for scheduling mixed real-time tasks[J].Journal of Systems Architecture,3 April 2000

[19] 范律,丁伟,朱建林.在 RT-Linux 实现优先级继承机制[J].计算机工程与应用,2003.5

[20] Ali Derbala.Priority queuing in an operating system. Computers & Operating Research,2005.

[21] Cheng,Xiao-Hui,Li Ming-Qiang,Wang Xin-Zheng.Embedded real-time operating system micro kernel design.ICMIT 2005:Information Systems and Signal Processing,2005

[22] 王新政，程小辉.嵌入式操作系统 uclinux 在 ARM 上的移植[J].桂林工学院学报，2005.12

[23] 王新政，程小辉，周华茂.实时操作系统任务调度策略的研究与设计[J].微计算机信息，2007.5

[24] 王俊祥,常用进程调度算法的分析与评价[J].计算机与信息技术,2006.8

[25] 程红蓉,一种实时嵌入式操作系统内核 DeltaCORE 的设计与实现[D].成都：电子科技大学,2001

[26] Jean J.Labrosse 著，邵贝贝等译. 嵌入式实时操作系统 uC/OS II (第二版)[M],北京航空航天大学出版社,2003

[27] 谭琦,嵌入式操作系统通信和同步机制的研究[D].长沙理工大学,2005

[28] 李明强.基于信息家电的嵌入式实时操作系统微内核研究与设计.桂林工学院,2005

[29] 李善平,刘文峰,李程远等. Linux 内核 2.4 版源代码分析大全[M].北京：机械工业出版社,2002.1

[30] 郑宗汉.实时系统软件基础[M].清华大学出版社,2003.1

[31] 汤子瀛,哲凤屏,汤小丹.计算机操作系统[M].西安电子科技大学出版社,1996.12

[32] 鲍远慧.形参个数不定的函数的编制和使用[J].微机发展,2001.3

[33] 丁国超.李全利.UC/OS-II 邮箱机制的分析与改进[J].信息技术,2004.12

[34] 米根锁.单片机应用系统中时钟管理方法的研究[J].电气传动自动化,2000.6

[35] S3C2410A 200MHZ&266MHZ 32-BIT RISC MICROPROCESSOR USER' S MANUAL

[36] Steve Furber 著,田泽,于敦山等著.ARM SoC 体系结构[M].北京航空航天大学出版 2002.10

[37] 王皓，平一凡.S3C2410 的中断异常处理机制[J].电子原器件应用,2006.8

[38] 张嘉庆.一种实时任务可调度性问题的研究[D].东北大学信息科学与工程学院,2004

[39] http://www.xiaocao.com/thesis/class2/class2/200611/30190.html

[40] 张尧学,史美林.计算机操作系统教程（第 2 版）[M].清华大学出版社,2001

[41] Tanenbaum A.Modern Operating System[M].Beijing China Machine Press,2002

[42] 方林波,黄樟钦,侯义斌.Linux 进程调度机制分析[J].北京工业大学学报,2005.7

[43] 曾非一,桑楠,熊光泽.嵌入式系统内存管理方案研究[J].单片机与嵌入式系统应用,2005.1

[44] 黄贤英,王越,陈媛.嵌入式实时系统内存管理策略[J].计算机工程与设计,2004.10

[45] 赵跃华,蔡贵贤,黄卫菊.一种嵌入式安全内存管理的设计与实现[J].计算机工程与设计,2006.8

[46] 沈勇,王志平,庞丽萍.对伙伴算法内存管理的讨论[J].计算机与数字工程,2004.3

[47] 刘忠仕,戴金海,桂先洲.实时操作系统 SACOS 的内存管理[J].计算机工程与应用,2006.5

[48] 吴荣华,邵时,杨早.基于中断的实时任务调度策略[J].计算机应用与软件,2007.1

个人简介

人个信息：

王新政（1981.2-）男 广西桂林人，2000.9-2004.7就读于桂林工学院计算机科学与技本科专业，2004.9-2007.6攻读桂林工学院硕士学位，专业为计算机应用技术，研究方向是嵌入式系统和嵌入式操作系统。

参与项目：

广西区科技攻关项目，项目的名称为：“基于信息家电的嵌入式实时操作系统研究”，项目编号：（桂科攻02350142）。

发表论文：

1、王新政，程小辉，周华茂. 实时操作系统任务调度策略的研究与设计. 微计算机信息, 2007.4(2)

2、王新政，程小辉. 嵌入式操作系统uClinux在ARM上的移植. 桂林工学院学报, 2005.12

3、Chen Xiao-Hui,Li Ming-Qiang,Wang Xin-Zheng. Embedded real-time operating system micro kernel design. ICMIT 2005: Information Systems and Signal Processing

4、周华茂，程小辉，王新政. 基于CAN总线的嵌入式测温系统设计. 微计算机信息（已录用）