重庆大学

硕士学位论文

嵌入式实时操作系统的自适应调度算法研究

姓名：孙萍

申请学位级别：硕士

专业：计算机软件与理论

指导教师：王铮

20070415

摘 要

当前数字信息技术和网络高速发展，嵌入式系统已经广泛地渗透到科学研究、工程设计、军事技术、各类产业和商业、文化、艺术以及人们日常生活等方方面面。随着嵌入式系统应用范围的不断扩大，功能不断增强，各个应用领域对嵌入式系统的要求越来越高，嵌入式实时操作系统逐渐被引入到嵌入式系统中。

嵌入式实时系统要求任务能按时、正确地完成功能操作，这些任务还可能会互斥地访问共享资源。另外，系统的处理能力和内存容量往往是有限的，这就对调度算法提出了严格的要求。实际的嵌入式实时系统在开发过程中往往忽视从运行的角度利用成熟的实时调度理论，而已有的实时调度理论过于复杂，不适于直接应用在嵌入式系统之中。

针对上述需求，本文研究了现有的经典实时调度算法及传统的自适应调度算法，分析了其应用于嵌入式系统的不足，从工控系统中任务的特点出发，将自动控制原理的思想引入到调度算法中。

本文提出了周期与非周期任务、I/O 消耗型与处理器消耗型任务的自动划分策略。对周期任务采用 RM 调度算法；而对于非周期任务，则采用 FIFO 调度算法。I/O 消耗型任务经常处于可运行状态，但一般都只运行短短的一会，因为它在等待更多的 I/O 请求时总会阻塞。对这类任务来说，调度策略适当提高它们的优先级，延长其运行时间。而处理器消耗型任务将时间大多用在执行代码上，对于这类任务，调度策略尽量降低它们的运行优先级，减少其运行时间。从这两个方面对任务进行相应的反馈控制，使调度器具有一定程度上的自适应功能。

最后，分析了 μC/OS-II 内核中与调度相关的数据结构，提出在其上的自适应调度算法的实现，主要包括控制器与基础调度器的实现。并通过实验比较了改进的算法与原算法的性能差别，验证了该调度模型用于嵌入式实时系统的可行性。系统所具备的主要功能特点使其在嵌入式实时系统领域具有良好的应用前景。

关键词： 内核，实时系统，自适应，调度算法

ABSTRACT

With the high-speed development of current numerical information technology and network, embedded systems have extensive penetration of scientific research, engineering design, military technology, various industries and commerce, culture, art and people's daily life, and so on. With the unceasing expansion of embedded system application, the function strengthening, various application areas of embedded systems increasingly high demands, so real-time operating system is gradually applied to the embedded system.

Real-time embedded system requires to complete functional operation timely and correctly, and these tasks may also contradict to access shared resources. In addition, the processing capacity and memory capacity of the system is often limited, which results in the stringent requirements to this scheduling algorithm. The development process of actual real-time embedded systems is often ignored from the running perspective using mature scheduling theory, but, the existing real-time scheduling theory is too complex and not suitable for direct application in embedded systems.

In response to these needs, this dissertation has studied existing classic real-time scheduling algorithms and traditional adaptive scheduling algorithms, and analyzed them applied to embedded systems inadequately, and we take the idea of automatic control theory into the scheduling algorithm from the task of the characteristics of the control system.

This dissertation presents the automatic strategy of periodic and non-periodic, I/O consuming and processor consuming tasks. We take RM scheduling algorithm for periodic tasks, and take FIFO scheduling algorithm for non-periodic tasks. I/O consuming task can often run , but, generally it run only in a short , because of it being obstructed when awaiting more I/O request. For such tasks, we use scheduling strategy to raise appropriately their level of priority, and the extension of its running time. But, processor consuming tasks spend most of their time on the implementation of code, and for such tasks, we take scheduling strategy to minimize their operational priorities and to reduce its running time. From these two aspects of the mission of the corresponding feedback control, we make the scheduler with a certain degree of adaptive function.

Finally, with analysing data structure of the scheduling of μC/OS-II kernel, we

have realized the adaptive scheduling algorithm in it, and mainly realized the controller and basic scheduling. The police compared the forged signature with the original. We have compared improved algorithm with original algorithm by simulation results, and validated feasibility in the scheduling model for real-time embedded systems. The system has a good prospect in the field of real-time embedded system because of its main function features.

Keywords: kernel , real-time system , adaptive, scheduling algorithm

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 重庆大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：孙舞　　签字日期：2007 年 6月 5 日

学位论文版权使用授权书

本学位论文作者完全了解 重庆大学 有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 重庆大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

保密（　），在____年解密后适用本授权书。

本学位论文属于

不保密（✓）。

（请只在上述一个括号内打“√”）

学位论文作者签名：孙舞　　导师签名：

签字日期：2007年 6月 5日　　签字日期：2007年 6月6 日

1 绪论

1.1 嵌入式实时操作系统

1.1.1 嵌入式系统

嵌入式系统(Embedded System)多指深藏于通信设备、工业系统、机电仪表、消费电子类产品内部，完成某种特定功能的计算机系统[1]。嵌入式系统可定义为：以应用为中心，以计算机技术为基础，软硬件裁剪，对功能、可靠性、成本、体积和功耗严格要求的专用计算机系统。

在当前数字信息技术和网络高速发展的后 PC（Post-PC）时代、嵌入式系统已经广泛地渗透到科学研究、工程设计、军事技术、各类产业和商业、文化、艺术以及人们日常生活的方方面面中。

随着嵌入式系统应用范围的不断扩大，功能不断增强，各个应用领域对嵌入式系统的要求越来越高。随着微处理器性能的不断提高，硬件结构的不断优化，那么限制嵌入式系统发展的瓶颈就突出表现在系统软件方面。1970 年左右出现了嵌入式系统的概念，此时的嵌入式系统很多都不采用操作系统，它们只是为了实现某个控制功能，使用一个简单的循环控制对外界的控制请求进行处理。当应用系统越来越复杂、应用的范围越来越广泛的时候，每添加一项新的功能，都可能要从头开始设计，没有操作系统已经成为一个最大的缺点。同时随着嵌入式系统规模的不断扩大，这种方法已经不能得到满足。在大型的嵌入式应用系统中，为了使嵌入式开发更加方便、快捷，需要一种稳定、安全的软件模块集合，用以管理存储器分配、中断处理、任务间通信、定时器响应以及多任务处理等，即嵌入式操作系统。它提高了嵌入式系统功能，方便了应用软件的设计，但同时也占用了宝贵的嵌入式系统资源。一般在比较大的多任务系统的应用中才使用。

① 嵌入式系统的特征[2]

1) 采用专用的嵌入式 CPU，其通常具有低功耗、体积小、集成度高等特点。

2) 具有实时性要求，即大多数嵌入式系统是实时系统。

3) 软硬件互相依赖。

4) 系统对用户透明。

5) 系统配置专一，结构紧凑，计算资源（速度和存储容量）有限。

6) 系统可靠性和安全性要求高。

7) 是多种技术的结合体。嵌入式系统是将先进的计算机技术、半导体技术和电子技术等各个专业的具体应用结合的产物，是一个技术密集、资金密集、不断创新的集成系统。

② 嵌入式软件的特征

嵌入式系统的软件与一般的软件相比，具有以下特点：

1) 响应时间快，有确定的实时性要求

嵌入式系统软件对外部事件的反应是快速、确定、可重复实现的，不管系统当时的状态如何都是可以预测的。

2) 嵌入式系统软件中应用软件与操作系统之间的界线模糊，往往是一体化设计

在通用系统中操作系统与应用软件之间的界线分明，应用程序独立设计、运行。而在嵌入式系统中，由于应用需求差别很大，其所需操作系统也繁简不一。这就要求根据不同的应用需求来配置操作系统和应用程序，装配成统一运行的软件系统。

3) 能够处理异步并发事件

嵌入式系统大多是事件驱动，外部事件是异步的并发事件。嵌入式软件应能够有效及时的完成这类外部事件。

4) 软件开发困难，要使用交叉开发环境

嵌入式系统的软件开发受到时间和空间的限制，一方面是由于嵌入式系统的资源（速度、内存容量、外设）有限，另一方面其软件要满足实时性和高可靠性的要求。再者，嵌入式系统的运行环境复杂，使得其软件的开发困难。

一般嵌入式软件开发要使用交叉开发环境，开发平台一般是具有在丰富计算机资源的通用计算机，称为宿主机（host），而实际的嵌入式系统称为目标机(target)。在交叉开发环境中，应用软件一般是先在宿主机上建立开发环境，进行应用程序编码和交叉编译，然后宿主机通过网口或是串口将交叉编译生成的可执行目标代码下载到目标机上进行调试。应用程序经过测试和优化，最后将应用程序固化到目标机中实际运行。

1.1.2 实时系统

实时系统的产生和发展动力来源于现实世界的实际需求。实时系统起源于二十世纪中叶，最初与军事上的需求紧密相关。随着计算机技术的飞速发展，实时系统逐渐在民用领域也得到广泛应用，如：铁路及机场调度、航空航天、核电站及化工过程监控以有具有 QoS（Quality of Service）需求的网络应用等，它已成为人类社会生活中所不可缺少的部分。

按照通常的定义，实时系统（Real-Time System）就是有一定时间约束的计算机系统，系统必须在明确的有界时间要求范围内，对信号做出响应[3]。当对处理器操作或数据流动有严格时间要求时，就需要使用实时系统；因此，它常用于控制特定应用的设备。实时系统不但要具有功能上的正确性，还要具有时间上的正确性，即，只有限定的时间内完成的计算及操作才是正确的。这一限定的时间被

称作最终截止期（deadline）。

1.1.3 嵌入式实时操作系统

嵌入式实时操作系统（Real-Time Embedded Operation Systems）是随着嵌入式系统的复杂度增加而引入到嵌入式软件系统中的。简单的嵌入式系统中不需要操作系统，但对于如今越来越复杂化的嵌入式系统，就需要相应的实时操作系统来管理嵌入式计算机的资源[4]。

许多实时系统运行在一个嵌入式系统中。如：电信系统的交换设备、飞机的飞行控制部分、汽车的驱动等等。虽然实时系统不一定是嵌入式系统，但嵌入式系统往往是实时的。嵌入式环境下的实时系统除了要考虑满足任务的最终截止期外，还要考虑其它的限制条件，如：内存资源有限，电能消耗受限等。

① 嵌入式实时操作系统的定义

操作系统是一组计算机程序的集合，用来有效的控制和管理计算机的硬件和软件资源，即合理的对资源进行调度，并为用户提供方便的接口。

嵌入式实时操作系统是指系统能及时响应外部事件的请求，在规定时间内对事件进行处理，并控制所有任务协调一致运行的操作系统。其中内核的大小，任务执行和切换效率，对中断响应的速度是嵌入式实时操作系统的重要指标。

② 嵌入式实时操作系统的特征

嵌入式实时操作系统除了具有通用操作系统的特征外，还具有以下特点[5]：

1) 实时性

实时性是嵌入式实时操作系统的首要特点，即在规定的时间内对外部事件作出响应。

2) 基于优先级的抢占调度

实时操作系统的实时性和多任务能力在很大程度上取决于它的任务调度机制。为保证响应时间，实时操作系统必须允许高优先级任务一旦准备好，立即抢占低优先级任务的执行。

3) 快速的任务切换

当由于某种原因使一个正在运行的任务退出运行时，RTOS 保存它的运行现场信息，并依据一定调度算法重新选择一个任务运行，这一过程所需的时间称为任务切换时间。任务切换时间一般在毫秒或微秒级上。

4) 禁止中断的时间间隔短

为了保证某些操作的原子性，通常在执行这些操作前关中断，执行完毕后再打开，从关中断到开中断之间的最大时间就是最大中断禁止时间间隔。RTOS 设计中要使中断禁止时间间隔最小。

5) 体积小

在 RTOS 的设计中，其占用内存大小是一个很重要的指标，一般实时内核只有几十 K 字节大小，并可固化使用。

1.2 国内外嵌入式操作系统简介

很多商业化的嵌入式操作系统是在八十年代出现的，这一热潮持续到现在。国际上有一些 IT 公司，在八十年代就开始进行商用嵌入式系统的研发，其中涌现了一些著名的商业化嵌入式操作系统，如：VxWorks、pSOS、WindowsCE 等，这些都是当今非常优秀的嵌入式操作系统[6]，下面简单的对这些产品进行介绍。

1.2.1 商业化产品

① WindowsCE：WindowsCE 是从整体上为有限资源的平台设计的多线程、多任务的操作系统。它的模块化设计允许它可以为具体的应用量体裁衣，操作系统的基本内核要至少 200K 的 ROM。

② VxWorks：VxWorks 是目前嵌入式系统领域中使用最广泛的系统。它支持多种处理器，如 x86、i960、Motorola MC68xxx、MIPS RX000、POWER PC 等。

③ pSOS：ISI 公司已被 WindRiver 公司兼并，现在 pSOS 属于 WindRiver 公司的产品。这个系统是一个模块化、高性能的实时操作系统，专为嵌入式微处理器设计，提供一个完全多任务环境，在定制的或是商业化的硬件上提供高性能和高可靠性。开发者可根据应用程序需要完成的功能和内存需求定制所需要的操作系统。

④ QNX：QNX 是一个实时的、可扩充的操作系统，它部分遵循 POSIX 相关标准。其内核仅提供 4 种服务：进程调度、进程间通信、底层网络通信和中断处理，其进程在独立的地址空间运行。所有其它的 OS 服务，都实现为协作的用户进程，因此 QNX 内核非常小巧（大约 12KB）而且运行速度极快。

⑤ Palm OS：3COM 公司的 Palm OS 在 PDA 市场上占有很大的市场份额，它有开放的操作系统应用程序接口，开发商可以根据需要自行开发所需要的应用程序。

⑥ OS-9：Microwave 的 OS-9 是为微处理器的关键任务而设计的操作系统，广泛应用于高科技产品中，包括电子产品、工业自动化产品、无线通讯产品、数字电视和多媒体设备中。它提供了很好的安全性和容错性，与其它的嵌入式系统相比，它的灵活性和升级性非常突出。

⑦ Lynxos：Lynx Real-Time System 是一个分布式、嵌入式、可扩展的实时操作系统，支持线程概念，提供 256 个全局用户线程优先级；提供一些传统的、非实时系统的服务特征。

这些专用操作系统均属源代码不公开的商业化产品，价格昂贵。由于他们各自的源代码不公开，使得每个系统上的应用软件与其它系统都无法兼容。并且，由于这种封闭还导致了商业嵌入式系统对各种设备的支持方面存在很大问题，使

得对其软件移植变得很困难。

1.2.2 开放源代码的嵌入式操作系统

① 嵌入式 Linux：Linux 本身不具有实时性，必须通过改造来获得对实时性的支持。改造的方式有两种，第一种方式是 RT-Linux 的思想，即加一个实时内核充分利用原有资源。第二种方式是利用 Linux 的开放源代码，从整体上改造 Linux。在嵌入式应用中还要把 Linux 裁减成最小的应用系统。现在市场上有 μCLinux、红旗 Linux、中软实时嵌入式 Linux 等很多 Linux 方面的产品，它们的性能比较好。但是需要指出的是 Linux 虽然没有版权费，但是商品化的 Linux 产品价格昂贵[7]。

② μC/OS-II：μC/OS-II 基本上免费，只是象征性的收取版权费用。μC/OS-II 支持多种单片机、甚至支持 DSP。μC/OS-II 专门为嵌入式应用而设计，具有很小的内核、很短的任务切换时间和实时响应速度。应用 μC/OS-II 进行开发，具有提高系统的实时响应速度、有利于软件设计的模块化、提高系统整体性能、以及提高抗干扰能力等优点。目前，μC/OS-II 已经在世界控制领域取得了一席之地，被广泛应用于各种嵌入式实时控制系统中。

1.3 本文的主要研究工作

本文提出了周期与非周期任务、I/O 消耗型与处理器消耗型任务的自动划分策略。对周期任务采用 RM 调度算法，而对于非周期任务，则采用 FIFO 调度算法。I/O 消耗型任务经常处于可运行状态，但一般都只运行短短的一会，因为它在等待更多的 I/O 请求时总会阻塞。对这类任务来说，调度策略适当提高它们的优先级，减少其运行时间。而处理器消耗型任务将时间大多用在执行代码上，对于这类任务，调度策略尽量降低它们的运行优先级，延长其运行时间。从这两个方面对任务进行相应的反馈控制，使调度器具有一定程度上的自适应功能。

最后，分析 μC/OS-II 内核中与调度相关的数据结构，提出在其上的自适应调度算法的实现，主要是包括控制器与基础调度器的实现。并通过仿真实验比较了改进的算法与原算法的性能差别，验证了该调度模型用于嵌入式实时系统的可行性。系统所具备的主要功能特点使其在嵌入式实时系统领域有良好的应用前景。

1.4 论文的组织结构

本文共分六章，内容的具体安排如下：

第一章，研究了嵌入式实时系统及其上所用的操作系统，并对国内外嵌入式的操作系统进行对比，引出本文所要研究的主要内容，并对论文的组织结构进行概要地介绍。

第二章，研究现有的实时调度的相关概念、基于单处理器的经典实时调度

算法、自适应在调度算法上的研究现状、及其应用于嵌入式系统的意义；在些基础上提出本文主要所研究的自适应调度问题。

第三章，根据嵌入式实时系统的需求及应用领域中任务的特点，提出一种针对两种常见的不同类型任务，进行自适应调度算法 ERTAS（Real-Time Embedded Adaptive Scheduling Algorithm），并提出了其系统结构。

第四章及第五章，分析 μC/OS-II 内核中与调度相关的数据结构，提出在其内核中自适应调度算法的实现，主要是包括控制器与基础调度器的实现。通过仿真实验比较了改进的设计算法与原调度算法的性能，通过分析说明了该调度算法的可行性。

最后，在第六章中，总结了论文的工作，并指出了进一步研究工作的方向。

2 实时系统调度

本章首先介绍实时系统的基本概念及相关术语，包括实时系统、实时调度、以及一些经典的实时调度算法；然后在此基础上，分析调度发展趋势，从而引出本文的研究内容。

2.1 实时调度基本理论

2.1.1 基本概念和相关术语

任务是指完成某一特定功能的软件实体，它是实时调度中的一个基本单位；任务在其生命周期中的某一次执行称为该任务的一个作业；具有实时性能需求的任务称为实时任务，反之则称为非实时任务[8]。

当作业已分配到除处理机以外的所有必要资源的时候，它能够被系统调度执行，此时的状态为就绪态。一个作业由其它状态转变为就绪态的那个时刻称为它的到达时间（Arrive Time）。

响应时间(Response Time)：作业自它的到达时刻起，到它被完成时刻的时间长度称为它的响应时间。若作业的响应时间有一个最大值限制，那么这个最大值被称为它的相对截止期（Relative Deadline）。若一个作业被要求在某个时刻之前完成，那么这个时刻称为它的绝对截止期（Absolute Deadline）。

最坏执行时间（Worst Case Execution Time，WCET），任务在其生命期中的每个作业的执行时间的最大可能值。WCET 常用来作为实时调度中任务对处理机的可能占用情况的一个衡量值。

截止期错过率（Deadline Missing Ratio，DMR），假设在某一段确定的时间内，错过其截止期的作业数为 n，正常完成的作业数为 m，则 $DMR = n/(m+n)$。DMR 是描述实时任务集在实际运行中，其实时性能需求情况的一个重要衡量值。

按对实时性能要求的程度，实时任务可分为硬实时(Hard Real-Time)和软实时(Soft Real-Time)两类。对于硬实时和软实时的定义并不统一，目前比较常用的有两种：

① 从任务的结果上考虑：硬实时有很强的可确定性要求，具有明确的时间约束，任务必须在其截止期限内执行完毕，在某个限定时刻之前不能完成任务将导致整个应用失败；软实时也对时间敏感，但偶尔发生的不能满足严格实时要求的情况是允许的，不过是不受欢迎的。

② 从任务完成的价值函数上来考虑：硬实时任务在超过截止期后，任务的价值函数将大幅度下降，甚至跌到负值；软实时任务在超过截止期后，任务的价值函数，

随任务的延迟逐渐减小。

2.1.2 实时系统

对于大部分计算机系统而言，只要计算结果正确，就可以满足正确性要求。而对于实时系统来说，正确执行指令和结果仅仅是问题的一个方面。关于实时系统的概念，人们给出了多种描述，以下是两种有代表性的：对于任何一个系统，若它对外部事件的及时响应是至关重要的，那么此系统是一个实时系统；在实时系统中，计算任务的正确性不仅仅依赖于计算结果逻辑上的正确性，而且还依赖于计算结果产生的时间。

所以，实时系统是与时间约束紧密相关的。这里的时间约束指：计算任务的开始时间、完成时间、截止期等。为了保证实时系统的时间约束得到满足，从系统结构上来看，可分为两个层面上的问题：

① 具体操作层面

主要考虑的是实时任务执行中每个具体操作所用时间及可预测性。如：CPU对每条指令的执行时间、内存访问时间、上下文切换时间等。

② 系统调度层面

通常体现在实时操作系统的调度器，依据各个实时任务对时间要求的紧迫程度等因素，对多个实时任务的执行次序的安排上。在多任务并发运行的系统中，调度器在何时选择某个任务运行，直接关系到这个任务是否在截止期之前完成。如果是给定的一组实时任务集，那么采用什么样的调度算法，直接关系到这组任务集的时间约束是否能够都被满足。

其中，第一个层面上的问题主要与一些系统具体实现的细节和硬件设计相关。第二个层面上的问题涉及实时系统调度理论、操作系统及计算机领域其它方面的内容，是影响面较广的根本性问题，需要做深入的理论研究。本文的工作主要在这个层面上展开。

2.1.3 实时调度

实时任务与一般只要求计算正确性任务之间最大不同就在于要满足处理与时间的关系。特别是在多任务并发的系统中，若要确保各个实时任务的时间约束，就需要对任务集合在处理机上的执行次序进行合理的安排。这样才能达到充分利用计算资源完成实时计算的目的。于是便涉及到实时系统研究的关键问题：实时调度。调度的实质是一种资源分配，即如何在系统的各个任务间合理的分配运行所需的资源，与分时调度相比，实时调度强调的是时间约束。

调度（Scheduling）和调度器（Scheduler）：调度就是根据调度算法的规则与资源协议，为一组任务选择符合要求的执行顺序，从而满足所有任务的时限要求。用来实现这一功能的软件模块就叫做调度器[9]。

调度表（Schedule）：由调度器生成的关于系统中所有任务的执行顺序的方案。

可行调度（Feasible Schedule）：一个正确的调度如果能保证系统中的所有任务的时限都得以满足，那么这样的调度叫做可行调度。

最优调度（Optimal Schedule）：对于任何一个客观存在可行调度性的任务集，如果这种算法总能找到这个任务集的可行调度，那么这个算法就叫最优算法，这种调度就叫最优调度。

2.1.4 实时调度方法的分类

根据不同的分类依据，实时调度方法的分类可有多种方法。

以下讨论常见的几种分类方式：

① 根据正在运行的任务是否可以被别的更紧迫和更重要的任务抢占，可分为抢占式调度和非抢占式调度。

在非抢占式调度中，不允许正在运行的任务被别的任务中断，任务一旦占有处理器便会一直运行直至完成。它比较适合于所有任务可依次执行的系统。由于没有额外的上下文切换，系统开销小，实现简单，具有较好的可预测性，易于测试。但此类方法不够灵活，对资源的利用率也较低。

在抢占式调度中，可允许更紧迫的任务中断目前正在运行的任务。同时，被抢占的任务在未来可以恢复运行，且不会影响到任务的整体时间约束。现代的操作系统一般采用此类调度。它的优点是比较灵活，对资源的利用率高；但由于经常出现的上下文切换使得系统开销相对较大。

② 根据调度顺序产生时机的方式可分为静态调度和动态调度。

静态调度是在任务集运行之前就产生一个静态的调度表，在运行时总是按照调度表决定从就绪任务队列中选择哪个任务来运行。这类调度算法假设系统中实时任务的特性是已知的，适合于问题需求确定，并且运行中不会有较大变化的情况。静态调度的优点是运行开销小，可预测性强。但是它的灵活性较差，不适合于动态变化或不可预测环境下的调度。

动态调度算法是在运行期间才决定选择哪个就绪任务来运行，它所根据的是目前已处于就绪态的各任务的相关属性，以此来决定调度序列。这类算法能够对环境变化做出反应，适合于任务不断生成，并且在任务生成之前，其特性并不清楚的动态实时系统。但是，动态调度算法的运行开销一般较前者大。

③ 根据调度方法是否具有自适应功能，可分为自适应调度和非自适应调度。

自适应调度通常采用闭环的结构，即在运行中不断采样系统输出，利用反馈信息自动适应环境变化。而非自适应调度则往往是开环结构，即在预知实时任务与调度相关的确切信息的前提下，系统运行开始前就确定好调度的策略和参数，而在系统运行中就不再根据反馈信息对它们进行调整。

2.1.5 资源访问控制协议

1997 年 7 月 4 日，美国国家航空航天局（NASA）的火星探测器探路者成功登陆火星。起初探路者在火星上的工作相当成功，但是几天以后它就开始出现错误，不断无规则地重启，每次重启都造成数据丢失。当时的解释是计算机试图同时完成太多的事情而导致问题的出现。但是后来喷气推进实验室的研究表明问题来自于优先级反转[10]。

在多任务共享资源时，当高优先级任务 T_1 访问被低优先级任务 T_2 所占用的临界资源 S 时，这时候就会出现优先级反转现象，尽管 T_1 具有高优先级，但只有 T_2 完成临界资源的访问后，T_1 才能接着访问，换言之，T_1 被 T_2 阻塞。

① 优先级反转（Priority Inversion）

图 2.1 解释了优先级反转是如何出现的，其中任务 1 优先级高于任务 2，任务 2 优先级高于任务 3。在这种情况下，任务 1 的优先级实际上降到了任务 3 的优先级水平，任务 1 和任务 2 的优先级发生反转。状态解释如下：

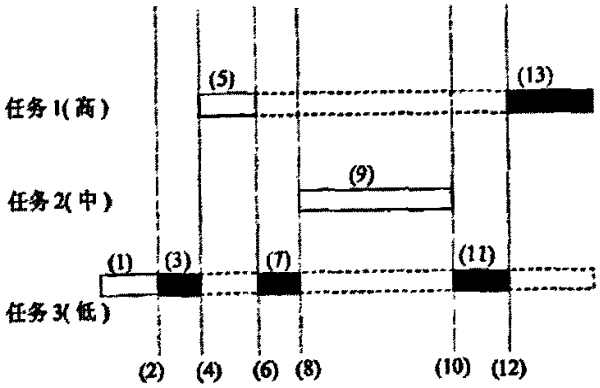

图 2.1 优先级反转

Fig 2.1 Priority inversion

(1) 任务 1 和任务 2 处于挂起状态，等待某一事件的发生，任务 3 正在运行。

(2) 此时，任务 3 要使用共享资源。使用共享资源之前，必须首先得到该资源的信号量。

(3) 任务 3 得到了该信号量，并开始使用该共享资源。

(4) 由于任务 1 优先级高，它等待的事件到达之后，剥夺了任务 3 的 CPU 使用权。

(5.6) 任务 1 开始运行。运行过程中，任务 1 也要使用任务 3 正在使用的资源。由于该资源的信号量还被任务 3 占用着，任务 1 只能进入挂起状态，等待任务 3 释放该信号量。

(7.8) 任务 3 得以继续运行。由于任务 2 的优先级高于任务 3，当任务 2 等待的事件发生后，任务 2 剥夺了任务 3 的 CPU 使用权并开始运行。

(9.10) 任务 2 运行，直至处理完成，将 CPU 使用权还给任务 3。

(11.12) 任务 3 继续运行，直至释放共享资源的信号。此时，由于实时内核知道有高优先级任务在等待这个信号量，内核做任务切换，使任务 1 得以运行。

(13) 至此，任务 1 得到该信号量，并开始处理共享资源，并得以运行。

实际的实时系统中，任务很少是独立执行的，经常需要互斥地访问共享资源。当任务试图访问资源时可能被正使用该资源的其它任务阻塞，调度分析时要考虑到这一因素。目前解决优先级反转有多种方法，其中使用比较普遍的有以下三种：L.Sha、R.Rajkumar 和 J.P.Lehoczky 提出的优先级继承协议（Priority Inheritance Protocol，简称 PIP）、J.B.Goodenough 和 L.Sha 提出的优先级天花板协议（Priority Ceiling Protocol，简称 PCP）、以及 T.P.Baker 提出的栈资源协议（Stack Resource Protocol，简称 SRP）。

② 优先级继承协议（PIP）

PIP 协议能与任何优先级驱动的抢占式调度算法配合使用，而且不需要有关任务访问资源情况的等级知识。该协议的执行方式是：当低优先级任务正在使用资源，而高优先级任务抢占执行后也要访问该资源时，低优先级任务将提升自身优先级到高优先级任务的级别。保证低优先级任务继续使用当前资源，以尽快完成访问并释放占用资源，使高优先级任务得以执行，从而减少高优先级任务被多个低先级任务阻塞的时间。

当高优先级任务被低优先级任务由于临界区资源而阻塞运行时，低优先级任务可以暂时继承高优先级任务的优先级，当阻塞结束后，此任务再恢复它以前的优先级，这种策略被称为优先级继承，如图 2.2 所示。状态解释如下：

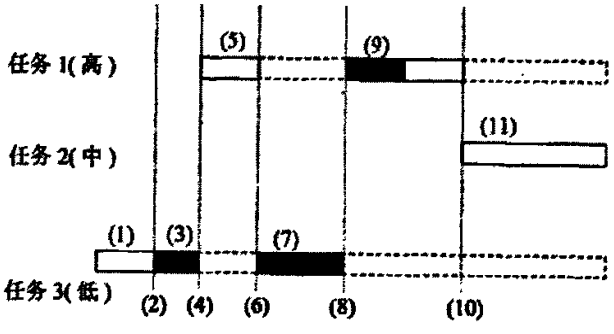

图 2.2 优先级继承

Fig 2.2 Priority Inheritance

(1.2) 任务 3 正在运行，但此时，任务 3 申请互斥型信号量(mutex)，以获得共享资源的使用权。

(3) 任务 3 得到并开始使用共享资源。

(4) CPU 的使用权被任务 1 剥夺。

(5.6) 任务 1 开始运行，任务 1 申请共享资源信号量 mutex。此时，内核知道该共享资源信号量 mutex 被任务 3 占用，而任务 3 的优先级比任务 1 低，于是内核将任务 3 的优先级升至与任务 1 相同。

(7) 内核将任务 1 列入等待共享资源信号量 mutex 的任务列表中，然后回到任务 3 继续运行，任务 3 继续使用该共享资源。

(8) 任务 3 完成后，释放共享资源信号量 mutex。这时，内核恢复任务 3 本来的优先级，并查看等待共享资源信号量 mutex 的任务列表，看是否有任务在等这个 mutex。内核看到任务 1 在等这个 mutex，将信号量 mutex 交给任务 1。

(9) 任务 1 得以运行。

(10.11) 任务 1 运行完成后，任务 2 开始运行。

因为只有高优先级任务访问正被低优先级任务使用的资源时，优先级继承才会发生，在此之前，高优先级任务能够抢占低优先级任务并执行，所以优先级继承协议不能防止死锁，而且阻塞是可以传递的，会形成链式阻塞。另外，优先级继承协议不能将任务所经历的阻塞时间减少到尽可能小的某个范围内。最坏情况下，一个需要 u 个资源，并且与 v 个低优先级任务冲突的任务可能被阻塞 $\min(u,v)$ 次。

③ 优先级天花板协议（PCP）

PCP 协议扩展了 PIP 协议，能防止死锁和减少高优先级任务经历的阻塞时间。该协议假设所有任务分配的优先级都是固定的，每个任务需要的资源在执行前就已确定。在这种方案中，系统把每一个临界资源与 1 个极限优先级相联系。这个极限优先级等于系统此时最高优先级加 1。当一个任务进入临界区时，系统便把这个资源优先权传递给这个任务，使得这个任务的优先级最高；当这个任务退出临界区后，系统立即将它的优先级恢复正常，从而保证系统不会出现优先级反转的情况。

已经证明，PCP 协议的执行规则能防止死锁，但其代价是高优先级任务可能会经历优先级冲顶阻塞（priority ceiling blocking）。即：高优先级任务可能被一个正使用某资源的低优先级任务阻塞，而该资源并不是高优先级任务请求。这种阻塞又被称为回避阻塞（avoidance blocking），意思是因为回避死锁而引起的阻塞。即使如此，在 PCP 协议下，每个高优先级任务至多被阻塞时会放弃处理器，因此访问共享资源的任务可能会产生 4 次现场切换。

④ 栈资源协议（SRP）

SRP 协议用于单处理器实时调度系统，能与 RM 或者 EDF 调度算法配合使用。应用 SRP 协议时，任务τ_i除了被分配优先级之外，还具有静态分配的抢占级λ_i。系统运行时，只有当$\lambda_i \succ \lambda_j$时，才允许任务$\tau_i$抢占任务$\tau_j$。

在 RM 算法下，定义任务的抢占级与周期成反比，即：

$$\forall \tau_i,\ \lambda_i \propto \frac{1}{T_i} \tag{2.1}$$

而 RM 算法给任务分配的优先级也与周期成反比，所以，此算法下任务优先级等于抢占级，都用π表示。

在 SRP 协议下，每个资源ρ^k会分配一个静态的冲顶值：

$$ceil(\rho^k) = \max\{\lambda_i | \tau_i uses \rho^k\} \tag{2.2}$$

协议还定义一个动态的系统冲顶值：

$$\Pi_s(t) = \max[\{ceil(\rho^k \mid \rho^k is currently locked)\} \cup \{0\}] \tag{2.3}$$

SRP 协议的调度规则是：只有优先级是当前活动任务实例中最高的，且抢占比系统冲顶值大时，任务实例才被允许执行。该协议保证任务实例一旦执行，直到结束也不会被阻塞，只能被其它高优先级任务实例抢占。这里的高优先级任务实例，指的是抢占级高于该实例执行期间系统冲顶值的任务实例。

SRP 协议又对 PCP 协议进行了简化和精炼，可用于多重资源和动态优先级调度算法。使用 SRP 协议时，每个资源分配一个静态的冲顶值，等于所有访问该资源任务具有的最高抢占级。协议还定义了一个动态变化的动态的系统冲顶值，等于所有当前正被使用资源具有的最高冲顶值。对于就绪任务，只有当其优先级是所有活动任务中最高的，而且抢占级大于系统冲顶值时才被允许执行。在静态优先级调度算法中，任务的抢占级等于其优先级。

与 PCP 协议相同，SRP 协议能防止死锁，高优先级任务经历的最大阻塞时间是有限的。在 PCP 协议下，只有高优先级任务试图访问共享资源时才可能被阻塞。而在 SRP 协议下，阻塞发生在任务执行前，一旦开始执行，直到结束也不会被低优先级任务阻塞，只可能被高优先级任务抢占，这被称作为提前阻塞（ealier blocking）。所以，与 PCP 协议相比，访问共享资源的任务在执行中的现场切换次数减少到 2，与不访问共享资源的任务相同。从实现的角度看，SRP 协议允许任务共享一个栈，从而减少了内存消耗。因此，任务开始执行后不会被阻塞，所以系统不需要实现等待队列，也减少了内存消耗，简化了调度内核的实现结构。

2.1.6 实时调度算法的性能评估标准

实时调度算法有多种性能评估标准，其中最常见的有：调度成功率、处理器利用率、处理器空闲率、吞吐率等[11]。

值得注意的是，以上这些性能评估标准不能完全兼容，没有一个调度算法可以同时在这几方面均达到最佳性能。对于实时调度算法来说，调度成功率是最重要的性能评估标准，算法在其它几个标准上的提高均不应导致调度成功率的明显下降。

2.2 单处理器实时调度算法

2.2.1 调度模型

周期任务在实时系统中较为常见，这类任务常用以下任务模型描述[12][13]：

问题描述： 假设一任务集: $S=\{r_1,r_2,....,r_n\}$

周期分别是: T_1, T_2,....T_n

每个周期内的最坏执行时间为:$C_1,C_2,...,C_n$

相对截止期为:$D_1,D_2,...,D_n$

对于第i个周期任务 $r_i(1\le i\le n)$，一定有 $C_i\le D_i\le T_i$

CPU 利用率用式(2.4)计算：

$$U=\sum_{i=1}^{n}\frac{C_i}{T_i} \tag{2.4}$$

对于周期任务的调度研究，由 Liu 和 Layland 在 1973 年提出了两种经典的调度算法：速率单调（Rate Monotonic，RM）算法和最早截止期优先（Earliest Deadline First，EDF）算法。这两种算法及其扩展算法已被人们深入研究，并且至今仍被广泛应用。

2.2.2 RM（Rate Monotonic）调度算法

在讨论 RM 之前，首先需要以下假设[14]：

A1. 所有任务可抢占，而且优先级抢占的耗费可以忽略；

A2. 只有数据处理的需求是重要的，内存、I/O 和其它资源需求可以忽略；

A3. 所有任务都是独立的，同时不存在优先级约束；

A4. 任务集合中的所有任务都是周期性的；

A5. 一个任务的相对结束期限等于它的生命期；

在 RM 算法中，一个任务的优先级和它的周期成反比，即一个具有较小周期的任务可以分配到一个较高的优先级。同时具有高优先级的任务可以抢占低优先级的任务。

静态优先级调度算法的一个最大的局限就是它不可能总是能够使得 CPU 完

全被利用，它在最差情况下的调度范围公式如下：

$$W_n = n(2^{\frac{1}{n}} - 1) \qquad (2.5)$$

其中，n 是系统中任务的个数。从以上公式可以看出，当系统中只有一个任务时，最差情况可调度范围是 100%，但是随着任务数目的增多，可调度范围随之降低，最终达到它的极限值 69.3%（ln2）左右。

RM 调度算法在实际中被广泛研究和使用。它是最理想的单处理器基于静态优先级的可强占式调度算法，也就是说，当任意一种静态优先级算法可调度时，RM 算法也可调度。

RM 调度算法的优点是开销小，灵活性好，可调度性测试简单。在任务的截止时间等于其周期的条件下，RM 调度算法已被证明是静态最优的调度算法。

2.2.3 DM（Deadline Monotonic）调度算法

DM 调度算法是在 RM 调度算法的基础上发展起来的，不同的是任务的优先级按截止时间来分配。截止时间短的任务优先级高，截止时间长的任务优先级低[15]。

DM 调度算法具有与 RM 调度算法相同的优点，但 DM 调度算法放松了对任务的周期必须等于其截止时间的限制。在任务的截止时间小于或等于其周期的情况下，DM 调度算法已被证明是静态最优的调度算法。

2.2.4 EDF（Earliest Deadline First）调度算法

EDF 算法是一种动态优先级调度算法，任务优先级在其初始时并不固定，而根据其绝对最后截止期限改变，所以在此算法下的处理器总是执行所有任务中最早达到绝对最后期限的任务。

在讨论 EDF 算法中，我们将保留讨论 RM 算法时的所有假设，除了这一假设：所有任务都是周期性的。EDF 调度算法的实现步骤：检查任务队列中所有就绪任务；比较所有任务最后时间期限；将具有最先时间期限的任务给予最高优先级，即最先执行该任务。

因此，抢占式 EDF 调度算法对于给定周期性任务集可调度性的充分必要条件为式(2.4)中 U≤1。由此可知，抢占 EDF 调度算法最大的优势在于，对于任何给定的任务集，只要处理器的利用率不超过 100%，就能够保证它的可调度性。

EDF 算法是一种最优的单处理器动态调度算法，也就是说，如果 EDF 算法不能合理调度一个任务集，那么其它算法也不能够调度该任务集[16]。

EDF 算法的优点在于效率高、容易计算，这是动态优先级调度研究的一个主要组成部分。EDF 算法的缺点在于，理论表明这种算法能对可调度负载进行优化，但是它不能解决过载问题。EDF 算法在系统的负载相对较低时非常有效，但在系统负载较重时，系统性能急剧下降，引起大量任务错过截止期，甚至可能导致 CPU

花费大量时间在调度上，在这里系统的性能还不如 FIFO（First In First Out）算法。

EDF 算法在实际应用中也存在一些问题，如：系统开销太大、在过载情况下会出现不稳定现象、优先级反转等。

2.2.5 LSF（Least Slack Time First）调度算法

LSF 调度算法计算任务的松弛时间，其中松弛时间为任务截止时间与剩余执行时间之差。该算法在任意时刻把最高优先级分配给具有最小松弛时间的任务，以此来保证紧急任务的优先执行。然而，由于等待任务的松弛时间是严格递减的，其等待执行的缓急程度也随着时间越来越紧迫，因此，在系统执行过程中，等待任务随时可能会抢占当前执行的任务。LSF 调度算法造成任务之间的频繁切换（颠簸现象）较为严重，颠簸现象增大了系统开销，并限制了 LSF 调度算法的应用。

2.3 各类型实时任务的混合调度

在实际的实时应用中，常常会遇到周期任务与非周期任务同时存在，硬实时与软实时需求同时存在的情况。例如，在雷达实时成像系统中，主计算机接收由数据采集设备按一定周期产生的外部图像数据，然后做存储和显示工作。为保证数据的完整性，存储操作要求为硬实时，而显示部分为软实时即可。当监控员在此过程中进行控制操作时，还产生了非周期的任务请求。

为了解决不同类型的实时任务同时存在情况下的调度，人们已经做了一些研究。下面对几种典型方法进行讨论，并基于这样的前提：系统中存在硬实时的周期任务，它们的 CPU 利用率小于 1，同时存在软实时的非周期任务[17]。调度目标是在保证硬实时周期任务的截止期的前提下，尽可能的提高软实时的非周期任务的响应速度。

2.3.1 后台执行法

首先满足周期任务的运行需要，所有的软非周期任务利用周期任务的剩余处理机时间运行。这种方法思路简单，但没有考虑到软非周期任务的响应时间，当处理机的硬周期任务负载达到一定程度时，响应时间就会比较长[18]。

2.3.2 轮询服务器法

创建一个周期性的任务为非周期性任务服务，称之为轮询服务器法（Polling Server）。每过一段时间该服务器就启动并执行待处理的非周期性请求。这种方法可以保证一定的非周期任务响应时间，它的响应性能优于后台执行法[19]。

2.3.3 能量保留算法

为了改善轮询服务器法的性能，在此基础上又发展了一组算法，称为能量保留算法。例如，优先级交换算法、可延期服务器算法和间发服务器法等。这里的服务器能量是指服务器在它的一个周期内可使用的处理机时间。这一类方法相对

于轮询服务器法所做的主要改进是，当服务器的服务周期内没有非周期任务执行请求出现时，它们不像轮询服务器法那样简单的丢弃服务器能量，而是将其保留下来待需要时使用。它们三者之间的差别是采用的保留能量的方法不同，所以可通称为能量保留算法[20]。

2.3.4 可延缓时间算法

一种利用周期任务的可延缓时间的调度算法[21]。可延缓时间是指周期任务在能满足自己的截止期的前提下，可延缓执行的时间。这种方法是在使用固定优先级的调度环境中，利用周期任务的可延缓时间为非周期任务服务。它被证明是特定条件下的最优算法，因此被称为一种固定优先级下针对软非周期任务的最优调度算法（An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Fixed-Priority Preemptive System）。这个算法使用可延缓时间表记录一个超周期（即组内所有任务的周期的最小公倍数）时间，每个周期任务的每个作业的延缓时间。使用一个计数器记录当前已处理的优先级高于或等于这个任务的周期作业量，另一个计数器记录当前非周期任务已经使用的总时间量。由此计算当某个时刻非周期任务到达时，周期任务的可延缓时间。虽然这是最优算法，但可延缓时间表可能会随着超周期的增大而开销很大。两个计数器也造成了额外的开销。另外，既然固定优先级下的处理机使用率受到限制，因此以固定优先级为前提的该算法用于为非周期任务服务的可延缓时间也会受限[22]。

这些算法为解决不同类型的实时任务共存情况下的调度做出了努力，但从以上分析中我们也可以看到它们往往都是针对某些特定条件，各个方法也并不能同时使用[23]。

2.4 自适应调度

操作系统中的任务调度本质上是一种决策行为，通过决策，来控制系统任务的有序、逻辑、及时的执行。控制论的引入是系统调度研究的必然，特别是对于复杂环境以及缺乏可预测性的不确定环境下的系统调度。

随着互联网技术、分布式技术、多媒体技术、移动计算技术等的发展，一方面，计算机系统越来越面对一个动态变化的环境；另一方面，实际应用对与时间因素相关的服务质量提出了需求。如何在这动态的环境下，使操作系统的任务调度具有一定的自适应能力，从而更有效、合理的进行系统资源分配越来越成为一个迫切需要解决的问题。

自适应调度是为了适应系统负载的动态变化、任务执行时间等的不确定性，利用控制论的方法，自动调节调度器等部件，以使系统资源分配适应环境变化。

2.4.1 自适应

自适应与其说是一种新技术，不如说是一种新的思想。近年来的迅猛发展与所取得的显著成果，已经引起广泛的关注。

什么是“自适应”？至今还没有一个统一的定义。

从字典中可以看到：“适应指改变其自身，使得其行为适合于新的或者已经改变了的环境。”

在我国的成语中，也有“因人制宜，因时制宜，因地制宜”之说，这些都包含了“自适应”这个哲理。人类社会和自然界的进化过程就是一个不断自适应和发展的过程[24]。

为了进一步说明什么是自适应控制，我们将引用 L.Ljung 从控制论的角度给出的定义：控制论的任务是找出合乎人们需要的控制器。控制器的设计可以看作是 S 到 R 的映射，这里 S 表示系统的集合，R 表示控制器的集合。

$$S \rightarrow R \tag{2.6}$$

设 O 空间是观察空间（即由所有观察值组成的空间），那么：

$$O \rightarrow S \tag{2.7}$$

从 O 到 S 的映射为可以称为系统辨识。自适应控制可以理解为从 O 空间到 R 空间的映射：

$$O \rightarrow R \tag{2.8}$$

Karl Johan Astrom 在＜Adaptive Control＞中给出的定义是：”An adaptive controller is a controller with adjustable parameters and a mechanism for adjusting the parameters.”

2.4.2 自适应控制系统

虽然自适应还没有统一的定义，但并没有阻挡这一理论的发展。

对于自适应系统的研究，最早是由航空问题引起。人们发现，一般的控制系统，对于在不同高度飞行的飞机和火箭，并不能都给出满意的结果。困难发生在不同高度的响应特征有着显著的不同，而一般的控制系统只能与某一高度的响应特征相匹配。因此，需要一种更成熟的控制系统，能随着响应特性的改变而改变控制系统自身，这就是自适应系统。

尽管自适应控制系统的方案千变万化，但是它们仍有一些基本的公共点，在其组成结构上有一定的相似性。也就是说在自适应控制系统中，除了具有通常反馈控制系统都有的受控对对象（P）与测量元件（M）之外，一般它还包括以下三个部分[25]，如图 2.3 所示。

其中，系统性能计算部分（E），该部分用以对受控对象进行实时在线检测，求得受控对象当前的动态特征。

控制决策（D），根据计算部分（E）提供的信息，按照预先规定的系统性能指标，实时在线地求出所需的控制器。

可调控制器部分（C），在自适应控制系统中，控制器（结构和参数）是可调的，它由控制决策部分 D 驱动而作出在线调整，以保证系统对内、外环境的变化具有自动适应的能力。

系统性能计算（参数估计）E 和控制决策部分 D 合在一起可称为自适应机构部分（A）。

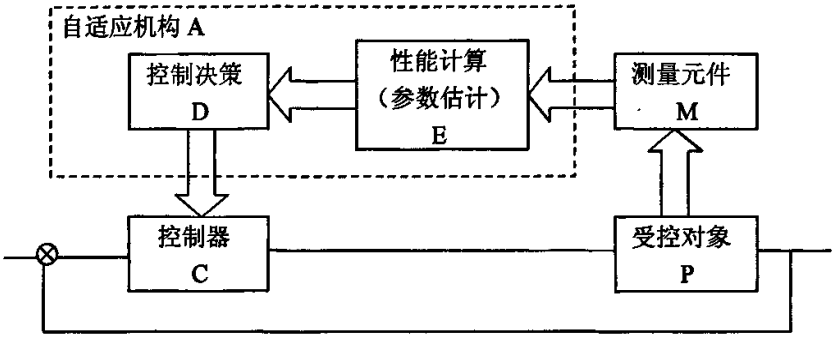

图 2.3 自适应控制系统的组成

Fig 2.3 Adaptive control system components

① 自适应系统的基本特征

自适应系统具备下列三个基本特征：

1) 过程信息的积累

这是用来降低被控系统原有的不确定性。为此可用系统辨识的方法辨识被控系统的结构和参数，直接积累过程信息；也可通过测量反映过程状态的某些辅助变量，间接积累过程信息。

2) 有一个可调的控制器

该控制器的结构参数和信号可以根据性能指标要求进行自动调整。这种可调性的要求是由被控系统的不确定性决定的，否则就无法对过程实现有效的控制。

3) 性能指标的控制

性能指标的控制可分为开环控制方式和闭环控制方式两种。若与过程动态相关的辅助变量可测，而且此辅助变量与可调控制器参数之间的关系又可根据物理学的知识和经验导出，这时就可通过此辅助变量直接调整可调控制器，以期达到预定的性能指标。这就是性能指标的开环控制，它的特点是没有根据系统实际达到的性能指标再作进一步的调整。与开环控制方式不同，在性能指标的闭环控制中，还要获取实际性能与预定性能之间的偏差信息，将其反馈后修改可调控制器，

直到实际性能达到或接近预定性能为止。

至今为止，有以下几种自适应系统类型为人们广为接受：

② 自适应系统的分类[26]

1) 模型参考自适应系统(Whitaker,Yamron 和 Kever，1958)

基本结构如图 2.4 所示。在这个系统中，采用了一个称为参考模型的辅助系统。由于指令信号，在输入到控制器的同时也加到这个参考模型的输入端，所以模型相当于输出响应的一个样板：即其输出或状态规定了系统的期望性能指标。为了比较给定性能和实测性能，可用减法器将参考模型和可调系统的过程输出或状态相减，得到广义误差信号。自适应机构再根据这个信号和一定的自适应规律产生反馈作用，以修改控制器参数；或产生一个辅助信号，促使可调系统与参考模型相一致，广义误差也趋向极小或为零。

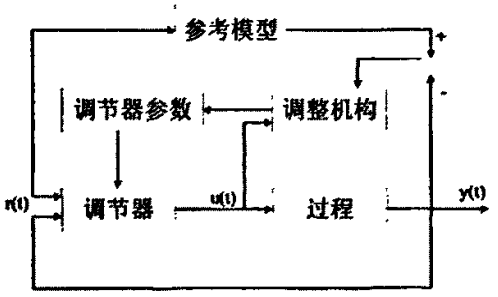

图 2.4 模型参考自适应

Fig 2.4 Model reference adaptive

2) 自校正自适应系统

基本结构如图 2.5 所示。自校正自适应系统的一个主要特点是具有被控对象数学模型的辨识环节。此类系统要根据系统运行数据，首先对被控对象进行辨识，然后再根据辨识得来的模型参数和事先指定的性能指标，进行综合控制。自校正自适应系统分为内、外两个环路。内环是由对象和控制器组成的常规反馈回路。外环由参数辨识器和控制器参数设计器组成，外环的任务是辨识过程参数，再按选定的设计方法综合出控制器参数，用以修正内环的控制器。

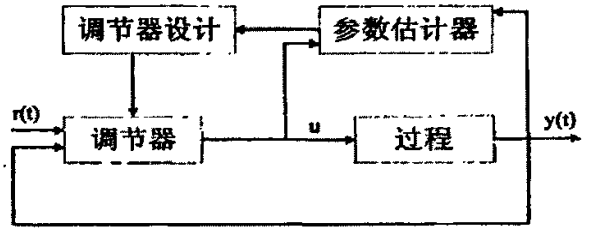

图 2.5 自校正自适应系统框图

Fig 2.5 Self-tuning adaptive system diagram

自适应控制的研究对象是具有不确定性的系统，这里所谓的“不确定性”是指被控对象及其环境的数学模型是不确定的。自适应系统需要通过不断地测量系统的输入、状态、输出或性能参数，逐渐了解和掌握对象，然后根据所得的过程信息，按一定的设计方法，做出控制决策，去更新控制器的结构、参数或控制作用，以便在某种意义下使控制效果达到某个预期的目标。

2.4.3 自适应调度

近年来人们突破了开环调度的思想，开始研究闭环的自适应调度方法。在这种自适应方法中，将系统的输出信息，返回到系统内部，根据系统当前状态及设定的目标状态的差值，作为下一步调度的依据，来提高实时系统的性能。可以说，使用自适应的调度思想，为实时系统的进程调度奠定了一个新的理论基础。

1990 年 4 月，美国洛杉矶某空军公司雷达系统组的 Muntz,A.H；Wang,K.在 IEEE First International Conference on System Integration 上，提出了有关进程调度的闭环控制[27]的论文。当时他们所研究的问题是如何在动态时间约束条件下进行进程的调度。问题来源于雷达的控制系统，当雷达的目标出现大规模转向或加减速度时，如何增加相应采样进程的频率，如何增加相应数据处理进程等，以便准确跟踪目标，避免目标丢失。从此，越来越多的学者开始关注动态环境下，基于自适应的进程调度的理论研究与方法探索。

实时系统的自适应进程调度可以理解为：实时系统的调度器能够根据系统运行中的观察值，动态的改变调度器自身，以适应系统内部情况与外界环境的变化。

实时系统的自适应进程调度的形式化定义如下。

设任务处理系统的输入向量集合为 RI,已有的性能指标向量集合为 RP_{pre}，未来的性能指标向量集合为 RP_{after}，实时系统可接受的性能指标向量集合为 $ARP_{acceptable}$，实时调度器为 RS，则：

$$\exists rf, RI \xrightarrow{rf} RP_{pre} \tag{2.8}$$

自适应进程调度的形式化定义为：

$$\forall RI,\ \forall RP_{pre},\ \exists RS, \{RI, RP_{pre}\} \xrightarrow{RP_{after} \subseteq ARP_{acceptable}} RS \tag{2.9}$$

对于自适应进程调度器，根据任务处理系统的输入向量集合 RI 和已有输出的性能指标向量集合 RP_{pre}，不断更新系统调度器 RS，使得将来输出的性能指标向量集合 RP_{pre} 落在实时系统可接受的性能指标向量集合 $ARP_{acceptable}$ 之内。也就是说，在实时系统运行进程中，进程调度器给出的系统调节量不断变化，但同时进程调度器本身也根据系统模型 rf 的不断变化处于不断变化之中。

如图 2.6 所示，实时系统分为内、外两个环路。内环是由任务处理子系统和

进程调度器组成的常规反馈回路。系统进程调度器 *RS*，根据任务处理系统的实际性能和性能指标的参考值的误差，不断计算对任务处理系统的调节量，使得实际性能指标和性能指标的参考值的误差不断减小。外环由任务处理子系统、自适应调节器和进程调度器组成，外环的任务是根据系统性能指标的参考值、任务处理系统的输入调节量和输出的性能指标，不断更新系统进程调度器 *RS*，使得在新更新的进程调度器的调度下，将来输出的性能指标向量集合 RP_{pre} 能够落在实时系统可接受的性能指标向量集合 $ARP_{acceptable}$ 之内，并不断减小。

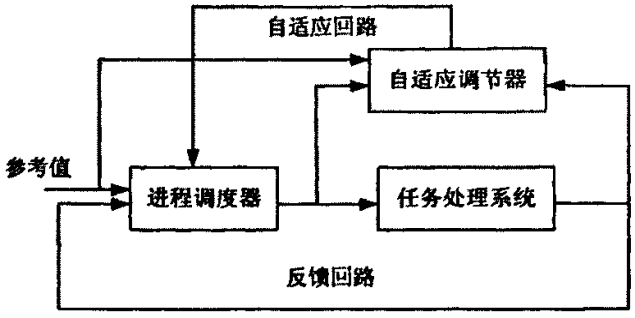

图 2.6 实时系统的自适应进程调度框架模型

Fig 2.6 Real-time system frame model of adaptive task scheduling

2.5 自适应调度应用于嵌入式系统的意义

上面介绍的都是通用的自适应调度算法，而在实际应用上，自适应调度算法在嵌入式领域有着重要的意义[28]：

2.5.1 家用网络系统

各种智能家电的个性化服务，新的智能家电的加入，设备与智能中心的连接等，必然使得各种处理器上运行任务的不确定性大大增加。而且必须能够及时处理微波炉的温度、定期检测房间温度、及时调节灯光的光线强弱等。对于这样的家庭信息网络环境的进程调度管理，就特别需要系统的个性化适应能力，以充分发挥信息家电的性能。

2.5.2 工控系统

实时系统经常作为嵌入式软件系统被嵌入在传感器、执行器及数控器中，其目的就是使得实际的设备状态接近于期望的状态。由于对工控系统的服务不断升级，由简单的实时数据采集，到实时数据采集、处理、可视化的图形显示监控，再到智能的数据处理、分析、决策等，必然引起任务调度复杂性的不断增长。目前工控系统越来越需要具有自适应调度能力的计算机系统，根据系统控制性能参数的要求，动态调整各进程的执行周期，或单位周期中的 CPU 处理时间等。

航空航天系统受空间、重量等因素的影响，用于其上的嵌入式系统处理能力有限，面对复杂的运行环境，调度算法的自适应性更为迫切。它们针对不同的任务，如：随着飞行高度的不同，某些传感器元件的测量控制明显不同。因此需要进行动态的进程调度，发挥最佳的性能，提高系统的服务质量。

2.5.3 车辆控制系统

当今汽车的科技含量越来越高，其中一个主要因素就是当今汽车的车身中含有越来越复杂的嵌入式处理控制系统。它们所面对的是动态的、无法预测的车辆运行环境。这这种环境下，对处理器中的任务调度，既要确保关键任务的执行，又要使系统中的非关键任务得到最大化调度，还要预留一定的处理器资源对付突发的任务，调度算法很难预先建立确切的数学模型。因此需要系统具备一定的自适应控制能力。

2.6 本章小结

本章首先介绍实时系统调度算法方面的基本概念和相关术语，包括实时系统、实时调度、进程调度过程中遇到的优先级反转问题；然后介绍基于单处理器的经典实时调度算法；在此基础上，为便于理解，介绍了自适应系统、实时自适应调度的形式定义、以及实时系统的自适应进程调度框架模型；最后提出了本文所要研究的内容，并提出了将自适应调度应用于嵌入式系统的意义。

3 基于嵌入式自适应调度的设计

本章阐述基于嵌入式实时系统的自适应调度算法，以下简称 ERTAS（Real-Time Embedded Adaptive Scheduling Algorithm）。先介绍嵌入式实时系统的需求，然后在此基础上，提出本文所要研究的 ERTAS 算法的体系结构。

3.1 嵌入式实时系统的需求

3.1.1 最佳实时性

对于实时系统，除了要满足应用的功能需求外，更重要的是满足应用提出的实时性要求。此外，实时任务之间可能还会有一些复杂的关联和同步关系，如：共享资源的互斥访问等，这为系统实时性的保证带来了很大的困难。因此，实时系统应遵循的设计原则是：采用合适的调度算法和资源访问控制协议，始终保证系统行为的可预测性。

3.1.2 有限的内存资源

许多嵌入式应用运行于 SoC(System on Chip)架构之上，目的是减少最终产品的价格。在嵌入式系统中，RAM 是稀缺资源，对产品的最终价格影响很大。有时为了节省几十个字节，不得不重新设计。同时这些嵌入式芯片的处理器主频一般都比较低，这样做也是为了降低成本，不浪费处理器资源。

表 3.1 嵌入式芯片内存容量

Table3.1 Memory capacity of embedded chip

芯片类型	芯片构成	处理器主频	ROM 大小	RAM 大小
AT91M40800	Atmel 32 bit ARM7 SoC	40MHz		8Kb
AT91F40816	Atmel 32 bit ARM7 SoC	40MHz	2Mb	8Kb
P89LV51RD2	Philips 8 bit microcontroller	33MHz	64Mb	1Kb
H8/3297	Hitachi 8 bit microcontroller	10MHz	16Mb	512b
ST10F269	Siemens 16 bit microcontroller	40MHz	256Mb	12Kb

表 3.1 显示了一些常用嵌入式芯片中内存的容量。可以看出 ROM 内存空间一般比 RAM 内存空间大 10 多倍。SoC 芯片上这些较少的内存资源要求应用软件必须尽可能少的使用 RAM 内存。要求内核使用的调度算法减少对内存资源，尤其是 RAM 资源的消耗。考虑到嵌入式芯片的处理能力一般比较低，还要求调度算法能充分利用处理器的处理能力。

3.1.3 有限的系统优先级

静态优先级调度算法采用静态调度测试为任务集合分配优先级。现有的静态

优先级调度算法都假设系统提供无限多的优先级，而实际上，底层实时操作系统能支持的优先级个数往往是有限的。表 3.2 列出了常用实时操作系统能提供的优先级个数。

当任务个数多于系统支持的优先级个数时，只能给几个任务分配同一优先级，这就会引起新的问题[29]：

① 如何将多个不同的任务优先级映射成一个系统优先级。

② 因为重新分配了任务优先级，所以还需要推导出判定映射后任务可调度性的公式，这就是优先级映射问题。

表 3.2 实时系统支持的优先级个数

Table3.2 Priority number of Real-time system

实时系统名称	Vxworks	μC/OS-II	RT-Linux	Real-Time POSIX	QNX
支持的优先级个数	256	64	256	32	256

3.2 ERTAS 的体系结构

传统的实时调度算法，都属于“开环型”的调度算法。开环调度意味着调度策略一旦确定，整个调度过程将严格按照调度策略来执行，而不会根据任务的实际执行状况对调度策略进行调整。但在最近几年中，越来越多的任务工作在开放和不可预测环境中，以前的调度算法就不能很好的满足实际应用的需求，自适应的调度思想得到了快速发展。

3.2.1 问题分析

① 任务就绪列表的分析

应用中任务个数不确定，如果是一个任务占用一个优先级，会浪费大量的优先级，任务数目也会受到限制，所以考虑一个优先级对应一个任务列表。

同时，实际应用中的任务可分为两类：周期任务和非周期任务。周期任务的发生一般有固定的周期。非周期任务大多由随机外部事件驱动，这类任务的到达时间是随机的。希望能够提出一种调度算法，对周期与非周期任务采用不同的调度方式。

② 时间片长短设置的问题

时间片是一个数值，它表明进程在被抢占前所能持续运行的时间。当一个进程的时间片耗尽时，就认为进程到期了。没有时间片的进程不会再投入运行，除非该轮其它所有的进程都耗尽了它们的时间片。

这里考虑到优先级高的进程，假定也是重要性高的进程，希望其执行的频率高，执行时间长。即，对应于当前任务中的 n 个优先级，每个优先级任务列表的

时间片均不相等。低优先级设计时间片短，为 5 毫秒；高优先级设计时间片长，为 100 毫秒，其它优先级的时间片在此间均匀分布。如图 3.1 所示。

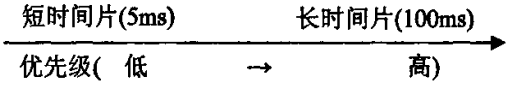

图 3.1 时间片长短设置

Fig 3.1 The length of set-up time slice

③ I/O 消耗和处理器消耗型任务的处理

任务可以被分为 I/O 消耗和处理器消耗型两类。前者指任务的大部分时间用来提交 I/O 请求或是等待 I/O 请求。因此，这样的任务经常处于可运行状态，但一般都只运行短暂的一会，因为它在等待更多的 I/O 请求时总会阻塞。为了保证这类交互式进程的响应速度，希望能够优先调度 I/O 消耗型进程。对这类任务来说，调度策略希望适当提高它们的优先级。

相反，处理器消耗型任务将时间大多用在执行代码上。除非被抢占，否则它们通常都一直不停地运行，因为它们没有太多的 I/O 需求。但是，因为它们不属于 I/O 驱动类型，所以从系统响应速度考虑，调度器不应该经常让它们运行。对于这类任务，希望尽量降低它们的运行优先级。

当然，这两种划分方法并非是绝对的。进程可以同时展示这两种行为。调度策略通常要在两个矛盾的目标中寻找平衡：进程响应迅速（响应时间短）和最大系统利用率（高吞吐率）。

3.2.2 体系结构

基于前一节中，三个方面的考虑。在此，提出一种基于动态优先级的自适应调度算法，其体系结构如图 3.2 所示。

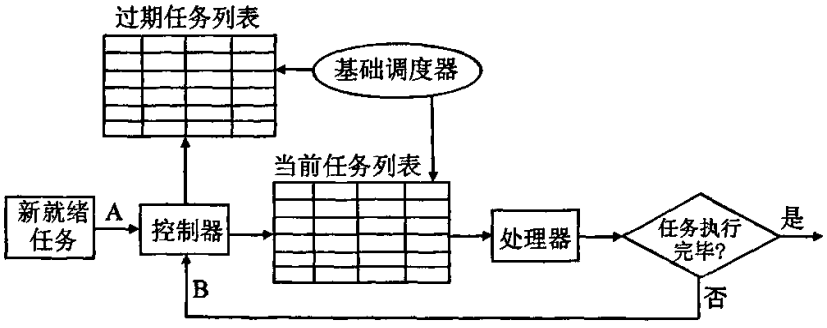

图 3.2 自适应调度的体系结构

Fig 3.2 Architecture of adaptive scheduling

在图 3.2 中，当有新的就绪任务(A)到达时，通过控制器，将其插入到当然任务列表中。基础调度器从当前任务列表中选择优先级最高的任务运行，选择出来的任务，经过处理器运行之后，如果任务运行完毕，则退出就绪队列；否则，如果是由于时间片到，CPU 使用权被剥夺而退出的任务(B)，该任务经过控制器，进入过期任务列表中等待。基础调度器的功能，除了从当然任务列表中选择出优先级最高的任务运行之外，在当前任务列表中所有任务都执行一遍时，还要负责将过期任务列表中的内容放入当前任务列表，并且清空过期任务列表，从而开始执行下一轮循环。

① 任务列表结构

图3.2中当前任务列表与过期任务列表，都与图3.3中就绪任务列表结构相同。前两者中的内容，均由控制器插入，具体插入算法可参见②。

在图 3.3 中，将连续的两个优先级划分为一个等级，则 n 个任务可分成 n/2 个等级（n 为偶数），每个等级中偶数位为周期性任务列表，存放周期任务；奇数位为非周期性任务列表，存放相应优先级的非周期任务。

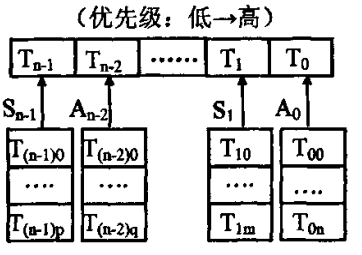

图 3.3 就绪任务列表的二维结构图

Fig 3.3 2D chart of ready task list

T_0~T_{n-1}：优先级为 0~n-1 的就绪任务列表，每个列表中又包含若干个任务，例：T_0 包含 T_{00} 到 T_{0n} 个任务；当 i<j 时，任务列表 T_i 中所有任务的优先级均高于 T_j 任务列表中所有的任务，例：任务列表 T_0 中所有任务优先级均高于 T_1 任务列表中所有任务的优先级。

A_0~A_{n-2}：经过控制器出来的自动划分的周期任务列表，优先级为 0~n-2(偶数)。其中 A_i 代表与 T_i 相同的任务列表，i 为偶数，例：A_0 代表与 T_0 相同的列表。

S_1~ S_{n-1}：经过控制器出来的自动划分的非周期任务列表，优先级为 1~n-1(奇数)。其中 S_j 代表与 T_j 相同的列表，j 为奇数，例：S_1 代表与 T_1 相同的列表。

T_{xy}：x 为任务 T 的优先级，取值范围 0-n-1，y 为在优先级为 x 的任务列表中的第 y 个任务。当前任务队列中的 n 个优先级的每个优先级对应一个任务列表，在这个列表中，每个任务都用一个字节 pribyte 来代表其二次优先级，T_{xy} 中 y 值

越小，其二次优先级越高，当选定的某个优先级任务运行时，在其对应队列中选择其二次优先级最小的任务运行即可。

② 控制器功能

本文所研究的自适应调度策略，主要体现在控制器的实现。

在这里，控制器可认为是一个任务插入算法。即，如果是新就绪任务到达(如图中 A 所示)，则由控制器插入到当前任务列表中，如果是 CPU 使用权被剥夺而退出的任务(如图中 B 所示)，则由控制器插入到过期任务列表中。

插入所依据的算法如下所示。当一个任务到达时，不管这个任务是新到达就绪任务，还是 CPU 使用权被剥夺而退出的任务，都进行以下的处理：

1) 周期与非周期任务的处理

控制器的一个主要功能就是判断出到达的任务是否是周期任务，并且针对周期与非周期任务不同，采取不同的插入方式。

判断是否是周期任务的机制如下：每个任务都记录了它的到达时间，存放于任务结构体的 time 域中。当任务再一次就绪时，利用此时的时间就可算出两次就绪时间的间隔 space，利用连续几次的间隔时间来自动判断是否是周期任务。

考虑到 RM 算法是最理想的单处理器基于静态优先级的可抢占式调度算法，并且条件 A4 中任务是周期任务(参见 2.2.2 节)，基于此，在自适应调度策略中，希望对周期与非周期任务采用不同的调度算法。对周期任务采用 RM 调度算法；而对于非周期任务，为了尽量减少系统开销则采用最简便的 FIFO 调度算法，以此来实现对周期与非周期任务的不同处理。如果是新任务到达，默认是非周期性任务。该部分具体的实现见本文 4.2 节所示。

2) I/O 消耗和处理器消耗型任务的处理

控制器的另一个主要功能，就是自动判断任务类型（I/O 消耗型和处理器消耗型），并且根据判断出的类型，适当地加、减优先级，再插入到任务列表中相应位置上。

推断一个进程到底是 I/O 消耗型还是处理器消耗型的方法如下：

每个任务都记录自身用于休眠和用于执行的时间。用于执行的时间存放于任务结构体的 runtime 域中。它的范畴从 prio_runslice(优先级为 prio 的时间片长度)到 0，当任务从休眠恢复到执行状态时，runtime 会根据运行的时间长短减少，直到小于 0。同时，任务每运行一个时钟节拍，如果上次 runtime 小于 0，则 task_runprio_number 就作相应的增加，当该值大于 3 就表示这个任务已经连续三次运行时均用完了时间片，刚将其优先级加 2。反之，则优先级减 2。当一个任务优先级的改变量已超过正负 10 后，则优先级不再改变。

③ 基础调度器功能

为了区别于 μC/OS-II 内核原有调度器，对此时自适应调度算法中的调度器改称为基础调度器。在此处，基础调度器主要实现两个功能。

一是从当前就绪任务列表中选择出优先级最高的任务列表，再从该任务列表中选择出二次优先级最高的任务开始执行。

二是当前任务列表中所有任务都运行一遍时，负责将过期任务列表中的内容放入当前列表中，过期任务列表清空，从而开始执行下一次循环。

基础调度器与 μC/OS-II 内核中原有调度器具体实现上的差别可见本文 4.3 节所示。

3.3 本章小结

本章首先介绍嵌入式实时系统的设计需求，然后在此基础上，提出本文所要研究的基于嵌入式系统的自适应调度算法的体系结构。该体系结构自动进行周期与非周期任务、I/O 消耗型和处理器消耗型任务的划分，并在此基础上，对该两类自动划分的任务进行相应的反馈处理，使调度算法具有一定程度上的自适应功能。具体的实现过程可参见下章。

4 ERTAS 调度策略在 μC/OS-II 内核中的实现

4.1 μC/OS-II 内核

μC/OS-II 操作系统是一个源码开放、结构小巧，可移植、可固化、可裁减、可剥夺的实时多任务操作系统。适用于小型控制系统，具有执行效率高、占用空间小、实时性能优良和扩展性强等特点，最小内核可编译到 2k[30]。

μC/OS-II 已经在世界范围内得到广泛的应用，包括许多领域，如手机、路由器、集线器、医疗设备及工业控制设备等。现在已经有成千上万的开发者把它成功地应用于各种系统，安全性和稳定性已经得到确认，并且通过美国的 FAA 认证。

总的来说，μC/OS-II 操作系统是一个很优秀的、实时性能优良的嵌入式操作系统，由于它对学校和教育是免费的，所以特别适合教学和科研。

4.1.1 内核整体结构

在对 μC/OS-II 调度算法改进之前，必须了解它的内核，这样才能有效的对其进行修改。μC/OS-II 的内核模块可以分为：任务管理、时间管理、事件控制、内存管理等。OS_CORE.C 主要实现与内核加载运行相关的模块，OS_TASK.C 与任务管理模块相关，OS_TIME.C 与时间管理模块相关，OS_MEM.C 与任务管理模块相关，而事件控制模块主要由 OS_MBOX.C、OS_Q.C、OS_SEM.C、OS_MUTEX.C、OS_FLAG.C 来实现。μC/OS-II 的系统结构图如图 4.1 所示[31]。

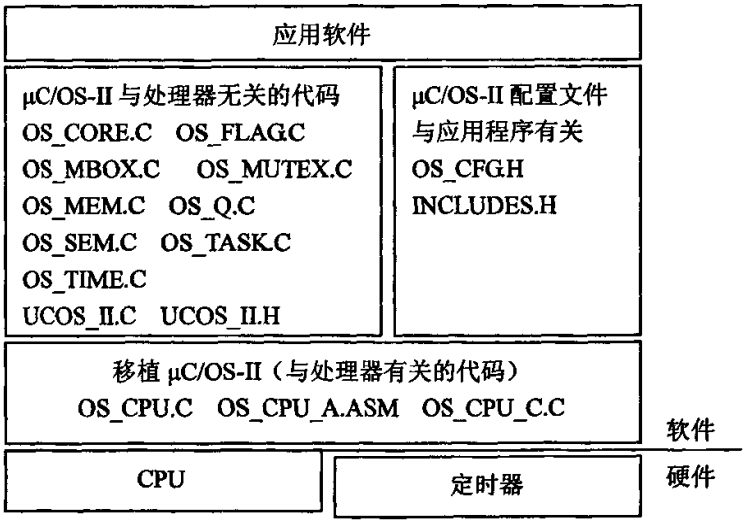

图 4.1 μC/OS-II 的系统结构图
Fig 4.1 System chart of μC/OS-II

4.1.2 任务状态分析

μC/OS-II 控制下的任务状态转换图，如图 4.2 所示。一般的，每个任务都是一个无限的循环，在 μC/OS-II 中每个任务任何时候都处于以下五种状态之一：休眠态、就绪态、运行态、挂起态和中断态，各个状态关系转换的函数如图 4.2 所示。

其中：休眠态，任务驻留程序空间，还没交给 μC/OS-II 管理；就绪态，任务本身具备运行条件，等待高优先级的运行态任务释放 CPU 后进入运行态；运行态，任务占用 CPU，正在运行；等待态，等待除 CPU 之外的其它资源，不能运行；中断态，正在运行的任务被中断，进入中断服务子程序。

任何任务状态的转变都是由系统调用(system call)或中断服务例程(Interrupt Serve Routine：ISR)引起，因为这样执行权才可能回到系统核心态，我们的内核才能履行调度的职能，所有的多任务系统都遵循这条规则。

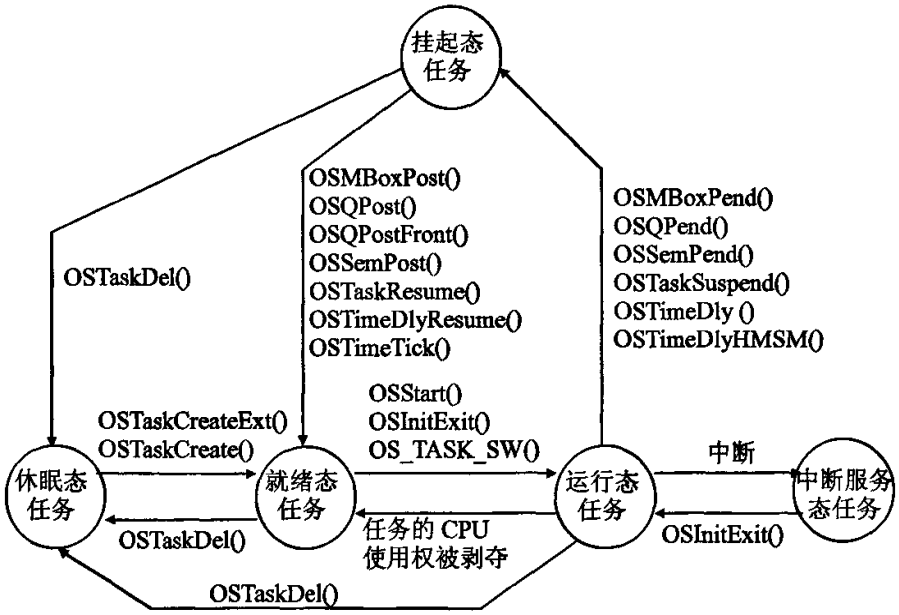

图 4.2 任务状态及转换函数图

Fig 4.2 Task state and transition function

4.1.3 任务控制块 TCB 数据结构

一旦任务建立了，任务控制块 OS_TCB 将被赋值。任务控制块是一个数据结构，当任务的 CPU 使用权被剥夺时，μC/OS-II 用它来保存任务的状态。当任务重新得到 CPU 使用权时，任务控制块能确保任务从被中断的那一点开始继续执行。

OS_TCB 全部驻留在 RAM 中，此数据结构的代码描述如下：

```
Typedef struct os_tcb
{
    OS_STK OSTCBStkPtr;            /* 是指向当前任务的指针 */
#if OS_TASK_CREATE_EXT_EN
    void OSTCBExtPtr;              /* 指向用户定义的任务控制块扩展 */
    OS_STK OSTCBStkBottom;         /* 是指向任务堆栈的指针 */
    INT32U OSTCBStkSize;           /* 存有栈中可容纳的指针数目 */
    INT16U OSTCBOpt;               /* 传给 OSTaskCreateExt 的"选择项"*/
    INT16U OSTCBId;                /* 识别码，保留未用*/
#endif
    struct os_tcb *OSTCBNext;          /* 任务控制块的双链接指针 */
    struct os_tcb *OSTCBPrev;
    OS_EVENT OSTCBEventPtr;            /* 指向事物控制块的指针 */
    void OSTCBMsg;                     /* 指向传给任务的消息的指针 */
    INT16U OSTCBDly;                   /* 任务延时节拍参数 */
    INT8U OSTCBStat;                   /* 任务状态字 */
    INT8U OSTCBPrio;                   /* 任务优先级 */
/* 以下四个用于加速任务进入就绪态的过程或进入等待事件发生状态的过程 */
    INT8U OSTCBX;
    INT8U OSTCBY;
    INT8U OSTCBitX;
    INT8U OSTCBitX;
    BOOLEAN OSTCBDelReq;           /* 用于表示该任务是否需要删除 */
} OS_TCB;
```

4.1.4 任务调度

μC/OS-II 内核采用的是基于优先级的可抢占调度，目前在 μC/OS-II 中不支持时间片。系统采用 64 位静态优先级分配策略，在程序运行前由用户为每个任务指定优先级。优先级数值越高，优先级越低，优先级取值在 0～63 之间，每个任务对应一个优先级。虽然空闲任务执行率最高，但是优先级设置为最低。

调度工作的内容主要分为以下几个步骤[32]:

① 寻找最高优先级任务

在 μC/OS-II 中，最高优先级任务的寻找是通过建立就绪任务表来实现的，每个任务的就绪态标志都放在就绪表中，当发生任务切换时，内核会在就绪表中查找已经就绪的最高优先级任务进行切换。

就绪表是一个 8×8 的二维表，如图 4.3 所示。它将 64 个优先级分成 8 组，第一行（组）从左到右依次是优先级 0 到优先级 7，依次类推；OSRdyGrp 一共有 8 位，分别对应就绪表的 8 个组（第 0 行到第 7 行），当第 n 组中的第 m 个任务就绪时，则 OSRdyGrp 的第 n 位置 1，同时就绪表 OSRdyTbl[n]的第 m 位置 1；OSRdyTbl[]是在 ROM 中的（在文件 OS_CORE.C 中定义）屏蔽字，用于限制 OSRdyTbl[]数组的元素下标在 0 到 7 之间；因为就绪表是 8×8 的，所以 prio[2:0]可以用来表示任务在就绪表中所处的列，prio[5:3]可以用来表示任务的优先级在就绪表中所处的行；OSUnMapTbl[]是优先级判定表，可以通过它来找到就绪表中优先级最高的就绪任务。可以对 OSRdyGrp 和 OSRdyTbl 进行操作来改变就绪表。

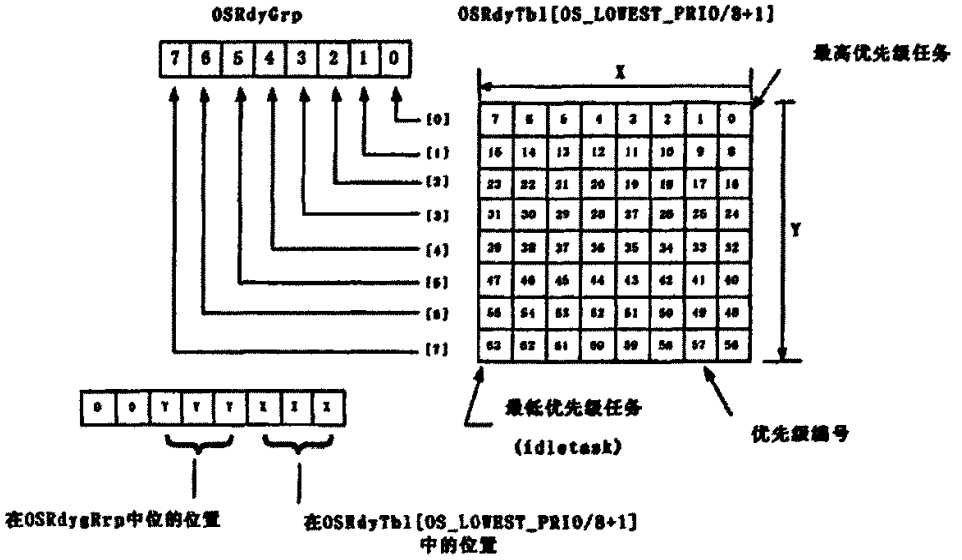

图 4.3 任务就绪表

Fig 4.3 Task ready table

② 使任务进入就绪表的代码

```
OSRdyGrp            |= OSRdyTbl[prio>>3];        //决定任务所在组
OSRdyTbl[prio>>3]|= OSMapTbl[prio&0x07];
```

③ 从就绪表中删除一个任务的代码

μC/OS-II 中所说的任务删除是指任务将返回并处于休眠状态，任务的代码不再被 μC/OS-II 调用。任务的删除是由函数 OSTaskDel()实现的。由于在 μC/OS-II 中任务的优先级同时也起着任务号的作用，所以 OSTaskDel()参数就是目标任务的优先级。如果要删除的任务是当前任务本身，刚可以使用一个特殊的优先级 OS_PRIO_SELF，即常数 OxFF。不过，系统中有一个任务是特殊的，那就是空闲任务。这个任务不允许删除，否则就有可能使任务调度无法进行。

OSTaskDel()首先检查参数的有效性，确保要删除的不是空闲任务，参数中优先级的值也应该是有效范围内，而且，占用该优先级的任务是应该存在的。另外，中断服务程序是不允许删除的，所以还要检查对函数的调用是否来自中断服务程序。如果是 OS_PRIO_SELF，就替换成当前任务的优先级。

根据给定的优先级找到目标任务以后，首先清除就绪任务组内位图中代表着目标任务的标志位，如果这是本任务组中最后一个就绪任务，则还要进一步将就绪任务分组

```
If( (OSRdyTbl[prio>>3] &= ~OSMapTbl[prio&0x07] == 0 )
    OSRdyGrp &= OSMapTbl[prio>>3]
```

④ 查找就绪表中优先级最高任务的代码

```
y = OSUnMapTbl[OSRdyGrp];
x = OSUnMapTbl[OSRdyTbl[y]];
```

使用位图方式查找就绪任务加快了查找的时间，优化性能。

⑤ μC/OS-II 中任务的调度是由函数 OSSched()完成的

```
void OSSched(void)
{
        关中断;
        if(不是中断嵌套并且系统可以被调度)
         {
                确定优先级最高的任务（即赋值给 OSTCBHighRdy）;
                if(最高级的任务不是当前的任务)    调用 OS_TASK_SW();
}
开中断;
}
```

任务调度模块首先用变量 OSTCBHighRdy 记录就绪任务表中当前最高优先级任务的 TCB 地址，然后调用 OS_TASK_SW()函数来进行任务切换。

4.1.5 μC/OS-II 调度中存在的问题

① 死锁[33]：系统不支持时间片调度，因而一旦任务进入死循环，就无法进入调度，其它的任务没有执行机会。解决办法是加入时间片轮询。

② 优先级倒置问题[34][35]：μC/OS-II 没有考虑优先级倒置问题，所以极易发生优先级倒置现象。考虑到实际应用中，需要创建数量较多的并发执行任务的复杂嵌入式应用，解决的办法可以是修改 μC/OS-II 底层数据结构，在抢占式调度策略的基础上使 μC/OS-II 支持同优先级任务机制，由此可突破创建任务数目的限制，进而还可以实现优先级继承协议。

μC/OS-II 本身存在的问题，我们希望在改进的调度算法中，能够尽量改进。

4.2 控制器的实现

对于任务 T_x（x为任务的优先级），控制器对其周期性处理的算法流程图，如图 4.4 所示。任务依据 RM 算法归入 Ai 队列的算法流程图，如图 4.5 所示。归入 Si 队列中，用 FIFO 算法插入。

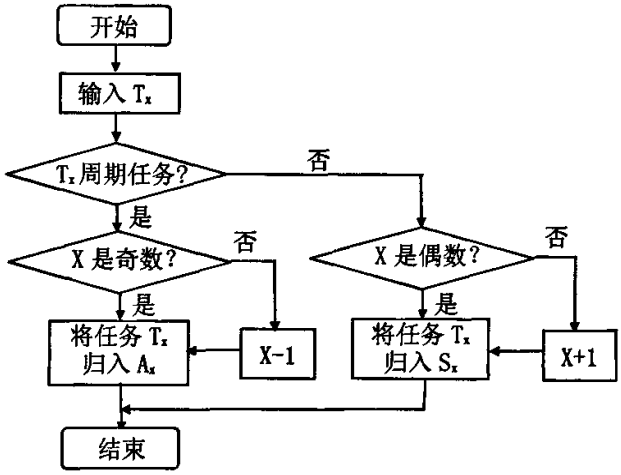

图 4.4 周期与非周期任务处理算法流程图

Fig 4.4 Process algorithm flow chart of period and aperiod task

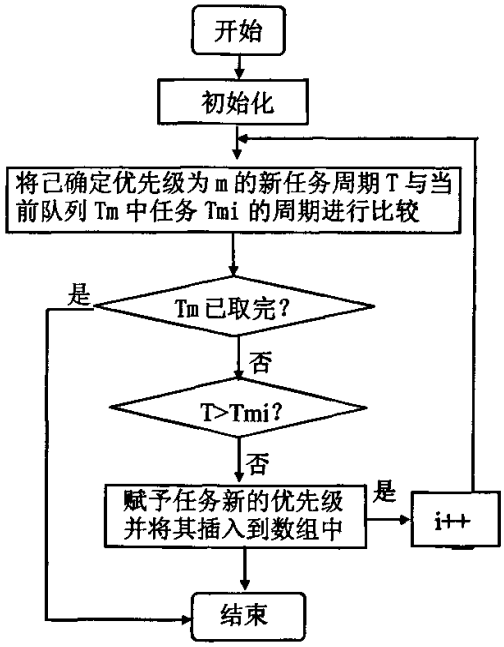

图 4.5 RM 算法流程图

Fig 4.5 RM algorithm flow chart

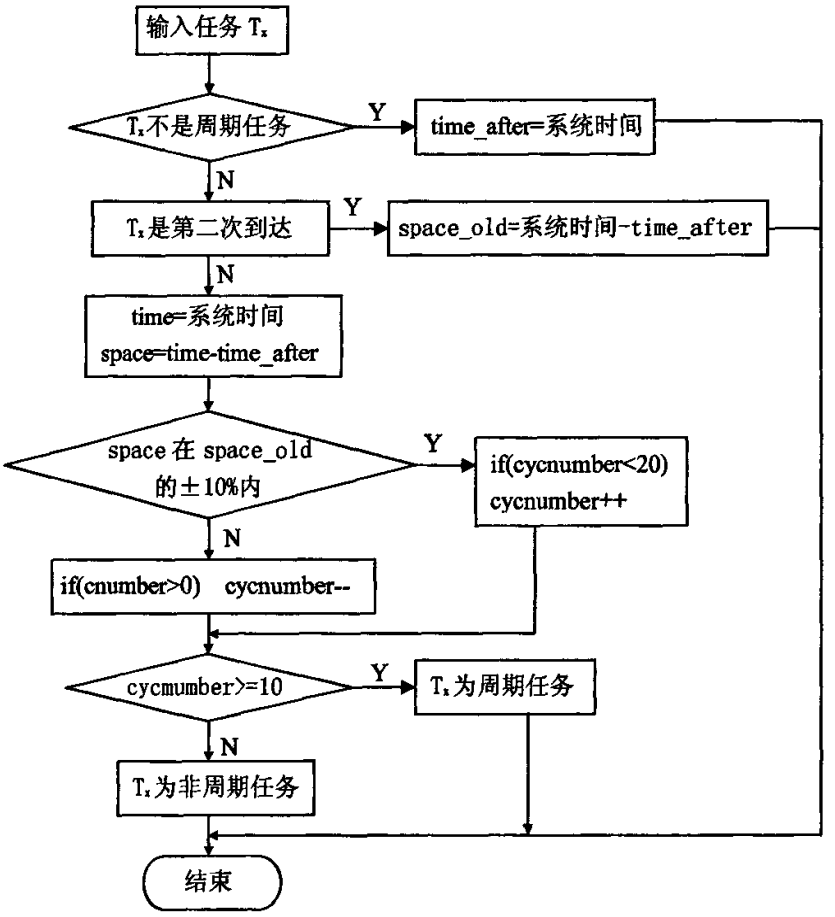

图 4.6 周期与非周期性任务的判断算法流程图

Fig 4.6 Period and aperiod task judgment algorithm flow chart

为了判断一个到达的任务是否具有周期性，现修改任务控制块 TCB 的数据结构，增加以下内容：

```
Typedef struct os_tcb
{
    ……
    INT32U time_after;
    INT32U time;
    INT32U space_after;
    INT32U space;
    INT32U cycnumber;
}
```

在周期与非周期性任务的判断算法流程图 4.6 中，默认新任务为非周期任务。如果任务是非周期任务的话，记录其此时的到达时间，存放于 time_after 中。当其下一次到达时，就可以计算出两次到达的间隔时间存放于 space_old 中，并将此时的 space_old 当作判断是否是周期任务的标尺。从该任务第三次开始，将第三次与第二次的到达时间间隔存放于 space 中，并将其与标尺相比较，如果相差不超过 10%，则将周期性指标 cycnumber++，否则将 cycnumber--。当周期性指标 cycnumber＞10 时，可认为此任务暂时具有周期性，将其认为是周期任务，否则认为其为非周期任务。

4.3 基础调度器的实现

在这里，为了简化基础设计器的实现，在 μC/OS-II 原有的调度器上进行修改。

定义调度管理结构体如下：

```
struct ScheduleStr
{
    int tasknum;
    unsigned byte bitmap[8];
    struct list_head queue[64];
}
```

在调度管理结构体 ScheduleStr 中选定下一个优先级最高的进程，只需要判定 bitmap 中优先级最高的位 i，并选择运行 T_{i0} 即可。具体执行过程可调用 μC/OS-II 自身的 schedule()函数实现。

4.4 本章小结

本章首先介绍开源免费的嵌入式实时操作系统 μC/OS-II 的内核整体结构、任务状态转换图、任务调度及其所涉及到的数据结构，并分析了 μC/OS-II 调度中存在的问题。在此基础之上，详细分析了本文在第三章提出的 ERTAS 在 μC/OS-II 内核上的关键实现技术，包括控制器与基础调度器的设计。

5 实验研究与应用前景

本章针对前两章所提出的基于嵌入式实时系统自适应调度算法，通过一系列实验对其实际效果进行测试和验证，并对所获得的实验数据进行分析，以进一步了解各种情况下的实时调度效果。

5.1 检测方法

将 μC/OS-II2.52 移植 ARM7 三星微处理器 S3C44B0X 上。

5.1.1 Samsung S3C44B0X 简介

Samsung S3C44B0X 微处理器是三星公司专为手持设备和一般应用提供的高性价比和高性能的微控制器解决方案。

Samsung S3C44B0X 最突出的特点是它的 CPU 核采用 ARM 公司的 16/32 位 ARM7TDMI RISC 结构，主频 66MHz。为了降低系统总成本的减少外围器件，这款芯片中还集成了下列部件：8KB Cache、外部存储器控制器、LCD 控制器、4 个 DMA 通道、2 通道 UART、5 通道 PWM 定时器和一个内部定时器、71 个能用 I/O 口、8 个外部中断源、8 通道 2 位 ADC 等。

5.1.2 μC/OS-II 在 S3C44B0X 上的移植

在移植的过程中，我们使用 ARM SDT 作为编译器。移植主要有三方面工作。分别编写与编译器、处理器、操作系统相关的代码[37]。

① 设置 OS_CPU.H 中与处理器和编译器相关的代码

1) 更改与编译器相关的数据类型。

因为不同的处理器有不同的字长，所以移植包括了一系列类型定义以确定其可移植性。

2) 修改与 ARM 处理器相关的代码。

```
#define OS_ENTER_CRITICAL() ARMDisableInt()   /* 关闭中断 */
#define OS_EXIT_CRITCAL() ARMEnableInt()      /* 开启中断 */
```

3) 设置堆栈的增长方向。

```
#define OS_STK_GROWTH 1      /* 堆栈由高地址向低地址增长 */
```

② 编写 OS_CPU_C.C 中 6 个与操作系统相关的函数

OSTaskStkInit()、OSTaskCreateHook(OS_TCB *ptcb)、
OSTaskDelHook(OS_TCB *ptcb)、OSTaskSwHook(void)、
OSTaskStatHook(void)、OSTimeTickHook(void)、

③ 在 OS_CPU.ASM 中用汇编语言编写 4 个与处理器相关的函数

OSStartHighRdy()：运行优先级最高的就绪任务

OSCtxSw ()：任务级的任务切换函数

OSInitCtxSw ()：中断级的任务切换函数

OSTickISR()：中断服务函数

完成了上述工作以后，μC/OS-II 就可以正常运行在 ARM 处理器上了。

5.1.3 实验方案设计

针对本文中提出的两种类型的任务，分别分为两类进行测试。一类是现场采集设备数据、数据交换区的刷新、数据计算及处理等周期性任务；另一类是控制命令执行、异常处理等非周期性任务。工业实时系统要求系统既能处理数据采集等周期任务，也要能处理不可预测的突发事件或随机事件引起的非周期任务。为此，系统测试侧重于对于周期任务与非周期任务执行情况的测试。

① 同时加载的两类任务

在被调度的任务集合中同时加载以下两类任务。并在不同的系统负载条件下，分别做了三组实验，进行比较分析。

1) 硬实时周期数据计算任务

2) 软实时非周期解码任务

其中，1)类任务定义为硬实时周期数据计算任务，优先级要高于 2)类软实时的非周期解码任务。2)类任务使用媒体播放软件 mplayer 进行实验，它是完全开放源码的软件。如果该播放器正常解码播放，每秒播放 25 帧图像，其播放任务可认为是软实时周期任务，而其解码任务只要 CPU 有空，就会解码，可看作为软实时非周期任务。这里选用解码播放任务，是考虑到解码任务是非周期，而刚好可以由播放任务看到非周期解码任务的时间偏差效果。

② 软实时非周期解码任务的时间偏差

在实验中，为使其能够将每帧图像数据的解码时间输出。我们修改了 mplayer.c 中的部分代码，在负责解码的程序代码中加入了输出解码时间的语句。

在 5.2 节的实验中，是让 1)类任务和 2)类任务并发运行，并监测 2)类任务中 mplayer 各帧图像解码的时间偏差。解码的时间偏差公式如下：

$$e_i = t_i - t_{i-1} - 40000 \qquad (5.1)$$

其中，e_i 为第 i 帧图像数据解码时产生的时间偏差，t_i 为第 i 帧图像解码完成时刻，t_{i-1} 为第 i-1 帧图像解码完成时刻，式(5.1)各变量的单位均为微秒，常量 40000 是正常情况下 MPEG4 相邻两帧图像的标准播放间隔(即每秒播放 25 帧图像)。

因此，若解码时间偏差小于或等于 0，则说明 mplayer 可以正常播放，否则，将出现图像抖动和延迟，并且偏差值越大，人眼可察觉的图像播放异常越明显。

③ 周期任务 CPU 利用率计算方法

对于周期任务，在每个周期中的每次运行所占用的处理机时间总是一个确定值。这样，我们对周期任务 CPU 利用率可以量化的得到，如下式(5.2)所示：

其中，B 为周期任务 CPU 利用率，t 为它在每个周期中运行时所需要的处理机时间，T 为它的周期。

$$B=\frac{t}{T} \tag{5.2}$$

5.2 实验

5.2.1 轻负载条件

当周期任务 CPU 利用率为 0.3 时，周期任务运行过程中的截止期错过率为 0。此时的系统负载较轻，CPU 还有足够的处理能力运行非周期解码任务。

表 5.1 为此时 μC/OS-II 解码视频文件的时间偏差和统计情况。

表 5.2 为此时 ERTAS 解码同一个视频文件时间偏差和统计情况。

表 5.1 轻负载下，μC/OS-II 的非周期解码任务时间偏差统计

Table5.1 Aperiodic decode tasks of μc/os-II time deviation statistics under light load

时间偏差(微秒)	发生次数	百分比(%)
-20000 以下	0	0.00
-20000~-10000	75	1.90
-10000~0	3813	97.07
0~10000	23	0.59
10000~-20000	12	0.31
20000 以上	5	0.13

表 5.2 轻负载下，ERTAS 的非周期解码任务时间偏差统计

Table5.2 Aperiodic decode tasks of ERTAS time deviation statistics under light load

时间偏差(微秒)	发生次数	百分比(%)
-20000 以下	0	0.00
-20000~-10000	46	1.37
-10000~0	3258	97.34
0~10000	35	1.05
10000~-20000	6	0.18
20000 以上	2	0.06

从表 5.1 和表 5.2 可以看出，两者在轻负载情况下，偏差在 0 微秒以下的帧数

百分比相关不大，mplayer 均能正常播放，实时性能差别不大。

5.2.2 中度负载条件

当周期任务 CPU 利用率为 0.6 时，周期任务运行过程中的截止期错过率为 0。此时的系统负载为中度负载，CPU 还有剩余的处理能力可满足大多数非周期任务的需求。

表 5.3 为此时 μC/OS-II 解码视频文件的时间偏差和统计情况。

表 5.4 为此时 ERTAS 解码同一个视频文件时间偏差和统计情况。

表 5.3 中度负载下，μC/OS-II 的的非周期解码任务时间偏差统计

Table5.3 Aperiodic decode tasks of μc/os-II time deviation statistics under moderate load

时间偏差(微秒)	发生次数	百分比(%)
10000 以下	3	0.00
10000~20000	2312	47.13
20000~30000	2528	51.53
30000~40000	53	1.08
40000 以上	10	0.26

表 5.4 中度负载下，ERTAS 的非周期解码任务时间偏差统计

Table5.4 Aperiodic decode tasks of ERTAS time deviation statistics under moderate load

时间偏差(微秒)	发生次数	百分比(%)
-30000 以下	0	0.00
-30000~-20000	37	0.79
-20000~-10000	1325	28.14
-10000~0	2716	57.68
0~10000	620	12.64
20000 以上	11	0.75

从表 5.3 可以看出，在中度负载条件下，μC/OS-II 原有调度机制已经不能保证 mplayer 的流畅播放，因为此时所有的时间偏差都大于 0，98.66%的帧数据都延迟在 10000 到 30000 微秒之间。每一帧图像数据都被延迟播放，人眼可以看到明显的播放抖动。

而在表 5.4 中，86.61%的图像数据解码偏差都控制在 0 以下，即使是被延迟播放的帧中有 12.64%也集中在 0 和 10000 之间分布，远比前者小。

对比表 5.3 和表 5.4 的数据，在此时的负载条件下，是否使用 ERTAS 实时调

度支持已经有了明显差别。

5.2.3 重度负载条件

当周期性任务 CPU 利用率为 0.8 时，周期任务运行过程中的截止期错过率为 0，此时的系统负载为重度负载。

表 5.5 为此时 μC/OS-II 解码视频文件的时间偏差和统计情况。

表 5.6 为此时 ERTAS 解码同一个视频文件时间偏差和统计情况。

表 5.5 重负载下，μC/OS-II 调度的非周期解码任务时间偏差统计

Table5.5 Aperiodic decode tasks of μc/os-II time deviation statistics under heavy load

时间偏差(微秒)	发生次数	百分比(%)
20000 以下	0	0.00
20000~30000	2889	82.83
30000~40000	563	16.14
40000 以上	36	1.03

表 5.6 重负载下，ERTAS 的非周期解码任务时间偏差统计

Table5.6 Aperiodic decode tasks of ERTAS time deviation statistics under heavy load

时间偏差(微秒)	发生次数	百分比(%)
-20000 以下	39	1.24
-20000~-10000	674	21.47
-10000~0	2156	68.68
0~10000	261	8.31
10000 以上	9	0.30

从表 5.5 可以看出，在重度负载情况下，μC/OS-II 原有调度机制已经不能保证解码的顺利进行。所有帧的解码时间偏差都大于 20000 微秒，98.97%的帧解码时间偏差都延迟在 20000 到 40000 微秒之间，每一帧图像数据都在较大程序上被延迟播放。

从表 5.6 可以看出，ERTAS 中 91.93%的帧解码时间偏差都小于 0，即使是被延迟播放的帧中有 8.31%也集中在 0 和 10000 之间分布。

对比表 5.5 和表 5.6 的数据，在此时的重负载条件下，ERTAS 作用较好，它在大部分时间下都将时间偏差控制在 0 以下。

5.2.4 实验分析

通过以上 3 组实验表明：本文在嵌入式实时系统 μC/OS-II 内核上所实现的自适应调度算法，可以集成调度硬实时周期任务、软实时非周期任务，达到了一般

嵌入式系统所要求的同时支持多类型实时任务的目标。

通过这 3 组实验数据的分析和比较，可得出以下结论：在系统负载较轻时，自适应调度机制与一般调度机制对实时任务的影响差别不大。但是，随着系统负载逐步加大，它们之间的差别就越来越明显。特别是从实验 3 中可以看出，在重负载条件下，当有自适应调度机制支持时，相对于 μC/OS-II 原有调度机制而言，非周期任务的实时性能得到了较大的改善。

5.3 应用前景

本文提出的嵌入式实时系统自适应调度方法，使得多类型的实时任务或非实时任务可在同一系统中并发运行，并且还具有根据环境变化自适应的调整功能，使运行时的计算资源分配更符合实时应用需求。这些功能特点较好的解决了目前嵌入式实时系统所面临的难题，迎合了实际应用的需要，从而带来了广阔的应用前景。

首先，从用户需求发展的角度，有时间约束的实时应用需求越来越突出，而目前应用于嵌入式系统的调度机制主要按公平性原则为所有应用程序分配系统资源，并未考虑到实时应用程序的时间约束需求。本文提出的自适应调度算法，在一定程度上解决了这类系统中的关键问题，

其次，在复杂工业控制、军事工业、航空航天、地理遥感等对实时性能较为敏感的应用领域，随着数据处理、控制决策等机制的日益复杂，以及人们对人机交互界面和可扩展性的要求越来越高，原有的封闭式实时调度已经难以很好的满足实际需求。因此，从实时应用软件开发角度，提高系统实时性能具有很大的实用性。

综上所述，本文提出的嵌入式实时系统自适应调度方法，丰富和完善了嵌入式实时操作系统的功能，使其更加具有实用性，使其在实时系统领域有良好的应用前景。

5.4 本章小结

实验研究是检验和评价系统调度方法的重要手段。

在此，先介绍了 μC/OS-II 内核所涉及到的调度方面的一些基本内容。

然后，针对本文第三章和第四章所提出的自适应调度方法进行了实验研究。在已实现的系统中设计了实验方案，通过同时加载多个周期与非周期性任务，创建了设计目标所要求的多类型任务并发运行环境，并检测在不同情况下的实时性能指标。实验结果表明，所提出的自适应调度方法是切实可行的、有效的，达到了设计目标。

在此基础上，对本文所提出的嵌入式实时系统自适应调度方法的应用前景做了分析，由于该调度方法针对实际需求，所以可具有良好的应用前景。

6 总结及进一步的工作

随着计算机科学技术的飞速发展，实时计算逐步兴起，实时系统也已经被广泛应用在人类社会生活中的诸多领域。目前开环的调度方式已经不能很好的满足实际应用需求，而近几年提出的自适应调度的思想却得到了大量的发展。然而，目前应用于嵌入式系统的调度算法，不论是在理论上的还是系统实现方面，实用性的成果比较少见。因此，为满足诸多应用领域对实时调度的需要，本文对嵌入式自适应方法进行了深入研究。

6.1 论文工作总结

本文对国内外各种嵌入式实时系统作了介绍，分析了经典的单处理器调度算法及各类型任务混合的调度，并对已提出的自适应调度算法作了详细研究。在此基础之上，从工控系统中任务的特点出发，将自动控制原理的思想引入到调度算法中。提出了一种针对周期与非周期性、I/O 消耗和处理器消耗型任务的自动划分策略，并从这两个方面对任务进行相应的反馈控制，使调度器具有一定程度上的自适应功能。

更进一步，对于广泛应用、但却并不完善的实时操作系统 μC/OS-II 的内核调度算法进行改进。在该内核中加入时间片，引入自适应的控制器，改进调度器，解决了 μC/OS-II 没有考虑的死锁与优先级倒置问题，最终改进后的算法具有极大的可行性。并在具体环境中，对各种不同的任务进行测试，可以看出改进后的调度算法在系统负载较重或系统过载的情况下表现良好。这为嵌入式系统 μC/OS-II 的应用前景带来巨大商机。

6.2 进一步的工作

虽然本文完成了上述工作，但是许多研究工作还处于初级阶段，还有很多问题有待改进：

① 对于周期与非周期性任务的判断，在此只提出一种基本的思想，在具体的实现过程中，其各个参数的取值还有待通过进一步的实验修正，或者是考虑用采样算法实现，使算法更加接近实际系统的需求。

② 引入对实际应用相关性的处理。目前的工作是假设实时应用之间是相互独立的，而在客观实际中，可能会遇到若干个实时应用之间有相互协作和通信的情况。因此，增加对实时应用相关性的处理会更适合分布式和大型的实时应用情况。

③ μC/OS-II 仅仅是一个实时多任务的内核，要想实现一个相对完整、实用性好

的嵌入式实时多任务系统，还需要加入相当多的扩展性工作。例如：建立文件系统、为外部设备建立驱动程序并规范相应的 API 函数、创建图形用户接口（GUI）函数、建立其它应用程序接口（API）函数等。

④ 将本文提出的调度算法的实用范围扩大。在 Internet、多媒体、工业数控等技术迅猛发展的情况下，具有实时需求的应用也逐渐普及起来。而实际上商用的实时系统大多是收费的，在此条件下，本文改进后的免费实时操作系统将存在很大的市场空间。

⑤ 考虑在多处理机的情况下实时系统自适应进程调度的研究。目前，本文的工作都是基于单处理机平台。随着硬件技术的不断发展，多处理机平台将可能与单处理机的情况有所不同，即还要考虑任务分配到哪个处理机上运行。让实时系统的自适应调度同时能适合多处理机的情况将会扩展它的应用场合。

致 谢

在重庆大学攻读硕士学位的三年时间对于我来说既是一次机会又是一项严峻的挑战，既是置身于知识的海洋又是生活在幸福的大家庭里。借论文完成之际，我由衷地感谢所有给予我关怀指导、支持和鼓励的老师、同学、朋友和亲人。

首先，衷心的感谢我的导师王铮副教授。感谢王老师为我提供了一个良好的学习环境，为我的论文工作奠定了基础。在三年的学习生活中，王老师给予我深切的关怀和热情的激励，在学术上悉心的指导，帮助我克服困难，发展思路。并以其无私的人格、严谨的学风、渊博的知识、饱满的工作热情和敬业精神和诲人不倦的师长风范，使我在学术和科学研究上取得长足的进步。

感谢我的同门李志军、杨志祥、汪平、王帅、陈小波等人，感谢他们与我所进行的讨论以及在学习过程中给予我的帮助和支持。与他们一起工作学习，共同营造的和谐气氛给我三年的学习生活增添了色彩。

还要感谢梁琰和李仙同学，特别是她们的专业背景和知识积累使我在科研项目的研究中节省了时间，在将计算机软件方面的理论与自动控制理论横向结合时，受到了一些启发。

感谢一切关心我和帮助我的领导、老师、同学和朋友们。

衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

孙 萍

2007 年 4 月 于重庆

参考文献

[1] 夏玮玮，沈连丰，肖婕，毛宇斌. 嵌入式系统关键技术分析与开发应用[M]. 单片机与嵌入式系统应用.2003.2:45-49

[2] 季志均，马文丽，陈虎，郑文岭. 四种嵌入式实时操作系统关键技术分析[J]. 计算机应用研究.2005 年第 9 期:73-78

[3] 邹勇，淮晓永，李明树. 开放式实时系统的自适应调度方法[J]. 计算机学报.2004 年 1 月. 第 27 卷.第 1 期:58-64

[4] 郑宗汉. 实时系统软件基础[M]. 清华大学出版社.2003:58-65

[5] 周洁，李正凡. 嵌入式实时操作系统的调度策略[J]. 华东交通大学学报.2005 年 08 月第 22 卷第 4 期:56-62

[6] 王琪. 嵌入实时控制系统的及应用[M]. 西北工业大学. 2004:23-31

[7] 郭江鸿，董祥和. 嵌入式实时操作系统中的调度问题[J]. 太原理工大学学报. 2004 年第 35 卷第 1 期:36-42

[8] C.M.Krishna. Kang G.Shin. Real-time Systems[M]. 清华大学出版社.2001.9:256-280

[9] 王志平，熊光泽. 实时调度算法研究[J]. 电子科技大学学报. 2000.Vol29.No2:47-52

[10] 王保进. 嵌入式实时系统的任务调度与资源共享模型及算法研究[D]. 解放军信息工程大学.2005:17-21

[11] 尹江会，刘捷，管素清. 实时系统中传统调度方式的一种改进方法[J]. 计算机工程与应用.2005.6:25-29

[12] Robert Love. Linux Kernel Development Second Edition[M]. 机械工业出版社.2006：16-48

[13] C.L.Liu，W.Layland；Scheduling algorithms for multiprogramming in a hard real-time environment [J]；Journal of the ACM，20(1):46－61

[14] Liu,C.L. J.W.Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment[J]. Journal of the ACM.20(1):46-61,1973

[15] Xu J. D.L.Parnas. Scheduling Processes with Release Times,Deadlines,Procedence,and Exclusion Properties[J]. IEEE Trans.Sotfware Engineering 16:360-369,1990

[16] Xu,J. Scheduling Processes with Release Times,Deadlines,Procedence,and Exclusion Properties[J]. IEEE Trans.Sotfware Engineering 16:139-154,1993

[17] 傅杰. 嵌入式操作系统的任务调度策略研究[D]. 湖南大学.2005:8-12

[18] Sha,L.,R.Rajkumar,and S.S.Sathaye;Generalized Rate-Monotonic Scheduling Theory:A Framework for Developing Real-Time System[J];Proc.IEEE,82:68-82,1994

[19] Baker,T.P. A Stack-Based Resource Allocation Policy for Realtime Processes[J]. Proc IEEE Real-Time Systems Symp. pp191-200,IEEE,Los Alamitos,CA,1990

[20] Muntz R R,Coffman E G.Preemptive scheduling of real-time tasks in multiprocessor systems[J]. Journal of the ACM,1970,17(2):324-338

[21] Han C C, Lin K J. Scheduling distance-constrained real-time tasks[J]. Proceedings IEEE Real-Time Systems Symposium,New York,1992:300-308

[22] Hsueh C, Lin K J,Fan N. Distrubuted pinwheel scheduling with end-to-end timing sonstraints[J]. Proceedings IEEE Real-Time Systems Symposium,1995:171-181

[23] 吕鸣松. 嵌入式工业监控系统中实时调度策略的研究[D]. 东北大学.2005:21-24

[24] 袁震东. 自适应控制理论及其应用[M]. 华中师范大学出版社.1988:1-10

[25] 李清泉. 自适应控制系统理论设计与应用[M]. 科学出版社.1990:2-10

[26] 赵国良，姜仁锋. 自适应控制技术与应用[M]. 人民交通出版社. 1991:2-16

[27] 童立靖. 实时系统的自适应进程调度方法研究[D]. 中国科学院软件研究所.2005:9-11

[28] Jeam J.Labrosse 著，邵贝贝译. 嵌入式实时操作系统 μC/OS-II(第 2 版) [M]. 北京：北京航空航天大学出版社..2003:72-143

[29] 陈冬. 工业控制领域的嵌入式系统实时性研究[D]. 武汉理工大学.2006:21-28

[30] 陈予吒. 基于 μCOS-II 的嵌入式系统设计与实现[D]. 浙江大学. 2004:54-62

[31] 谢长生，马进德，黄浩. 基于 uC/OS-II 的任务调度策略研究[J]. 计算机工程与科学. 2004 年第 26 卷第 8 期:68-72

[32] 郭勐，黄江洪，王贞松. 实时操作系统 uC/OS-II 的多任务改进方法[J]. 计算机工程与应用.2005.12:32-35

[33] 林游，韩志科. 在 uC/OS-II 上实现优先级天花板[J]. 单片机与嵌入式系统应用. 2005.4:23-27

[34] 邵志勇，张学东，马丁. μCOS-II 实时操作系统任务调度的改进[J]. 鞍山科技大学学报. 2003 年 10 月第 26 卷第 5 期:65-70

[35] 郭洋. uC/OS-II 中解决优先级反转问题的设计模式及实现方法研究[D]. 沈阳.沈阳工业大学.2006:6-11

[36] 邹勇. 开放式实时系统的调度方法研究[D]. 中国科学院研究生院.2003:89-92

[37] 王田苗. 嵌入式系统设计与实例开发-基于 ARM 微处理器与 μC/OS-II 实时操作系统[M]. 清华大学出版社.2002.9:60-76

附录：作者在攻读硕士学位期间发表的论文目录

[1] 孙萍，王铮，《基于嵌入式实时系统的反馈调度算法的研究》，计算机技术与发展，2007