桂林理工大学

硕士学位论文

嵌入式实时操作系统微内核任务调度算法的研究

姓名：张一哲

申请学位级别：硕士

专业：计算机应用技术

指导教师：程小辉

201104

摘 要

随着嵌入式系统的发展，嵌入式系统已经广泛的应用到科学研究、工业控制、军事技术以及人们的日常生活等各个方面。尤其在实时领域，基于实时的嵌入式操作系统也得到了更多的应用。正是由于实时操作系统在嵌入式系统中的地位日益提升，对实时操作系统的研究已成为嵌入式系统研究中的重要部分。

调度算法是嵌入式实时操作系统的核心所在，因此对嵌入式实时操作系统调度算法进行研究和分析，以提高实时性有着重要的意义。若要确保各个实时任务能够及时调度并安全完成，就必须对任务集在处理器上进行合理的调度，如何对实时任务进行调度同时又保障实时系统时限性和高可靠性，是实时操作系统研究的一个关键问题。不同类型的实时系统对资源要求以及对时间约束的形式不同，因此其调度算法就不同，如当任务是周期性的，其周期就很重要，当任务是非周期性的，则截至期就很重要，调度算法是系统能否实时响应的关键。

本文首先介绍了本论文的研究背景、意义以及国内外研究概况。然后给出了嵌入式操作系统和嵌入式实时操作系统的定义，介绍了嵌入式实时操作系统发展趋势。并介绍了微内核的相关概念、技术特点和体系结构和常见嵌入式实时操作系统微内核，其中包括 Vxworks，μC/OS-II，μClinux 等。基于以上基础，研究了微内核的实时调度算法，并对其调度算法做了分类和总结。最后分析比较了静态实时调度和动态实时调度其中的不足之处。

基于以上研究后，提出问题。对任务集建模，详细叙述了新的优先级的分配方案，此方案通过衡量任务价值和任务时限来决定任务优先级的高低。讨论了实时调度器和中断管理机制，叙述了新的微内核任务调度算法，算法的设计原则是任务的时限越小且价值越大，则任务的优先级越高；对于时限与价值完全相同的任务，先到达者具有更高的优先级。并给出了新的微内核任务调度算法的流程图。

然后对微内核 μC/OS-II 的调度算法做了详细的分析和研究，包括任务控制块，任务就绪表和任务调度器。分析和总结了 μC/OS-II 实时内核优缺点，通过实现时间片轮转算法在内核部分的扩展，优先级继承算法，扩充任务数，改进调度算法，使得构建的实时内核更可靠及具有更强的适用性。

最后以 μC/OS-II 作为原型，实现新的微内核任务调度算法，通过性能测试实验证明对于任务周期长的任务集，性能得到明显提升，对于任务周期短的任务集，性能也得到小幅提升，新的微内核的实时性得到了提高。

关键字：微内核，嵌入式实时操作系统，任务调度，EDF，优先级

Abstract

Along with the development of embedded system, the embedded system has been widely applied to scientific research, industrial control, military technology and People's Daily life, etc. Especially in real-time field, the embedded operating system based on real-time also got more applications. Because of the real-time operation system in embedded systems, the position rising to real-time operating system research has become an important part in the embedded system.

Scheduling algorithm is embedded real-time operating system, therefore the key to embedded real-time operating system scheduling algorithm for research and analysis, in order to improve the real-time has important significance. If you want to ensure that individual real-time tasks can prompt scheduling and safe completion, it needs to be conducted on a processor task set reasonable dispatch, how to real-time tasks scheduling simultaneously ensure real-time system time limit sex and high reliability, it is real-time operating system research a key issue. Different types of real-time system for resource requirements and to the form of time constraints are different, so its scheduling algorithm is different, such as when a task is cyclical, and its period is very important, when a task is cyclical, then by period is very important, scheduling algorithm is the key system could real-time response.

This paper first introduced this topic research background, significance and research situation. Then give the embedded operating system and embedded real-time operating system, this paper introduces the definition of embedded real-time operating system development trend. And introduced the micro kernel related concepts, technical features and system structure and common embedded real-time operating system, including micro kernel Vxworks, μC/OS-II, μClinux etc. Based on the above basis, and studied the real-time scheduling algorithm micro kernel, and its scheduling algorithm have classified and summarized. Finally, compare the static and dynamic real-time scheduling real-time scheduling in which the deficiencies.

Based on the above research, ask questions. For the tasks set modeling, this paper describes the new priority scheme, this plan through measuring task value and task time decision task priority of ruling. Discussed the real-time scheduler and interrupt management mechanism, described the new micro kernel task scheduling algorithm, algorithm design principles of the smaller is task and the time limit, the task of value higher priorities; For the same time limit and value to the task, first with higher priority. And it gives a new micro-kernel task scheduling algorithm flow chart.

Then to the micro kernel μC/OS-II scheduling algorithm has made the detailed analysis and research, including mission control blocks, mission ready table and task scheduler. Analyzes and summarizes the μC/OS-II real-time kernel advantages and disadvantages, by implementing time piece in the kernel part of the rotation algorithm expansion, priority inherited algorithm, expanded mission number, improve scheduling algorithm, making the real-time kernel building more reliable and has a strong suitability.

Finally to μC/OS-II as a prototype, to realize new micro kernel task scheduling algorithm, through the simulation experiment proof for task cycle long task set, performance are obviously to ascend, for the task set task cycle is short, performance also get modest ascent, the new micro kernel of real-time improved.

Keyword: Micro-kernel, Embedded Real-time Operating System, Task Scheduling, EDF, Priority

第1章 引言

1.1 研究背景及意义

目前计算机技术、微电子技术、嵌入式系统技术得到了突飞猛进的进步，能源、交通、工业、医疗等这些行业都使用着嵌入式系统，为了满足行业的要求，我们需要一个高实时性的系统，它的响应时间必须要控制到纳秒以下。随着人们需求的提高，实时操作系统也得到了空前的发展，如今的处理器已经达到了 64 位，数据的处理能力有了很大的提高，现在的处理器芯片含有很多的种类，而以前的只包含单独的芯片，现在的内核包含其他的很多的应用功能，例如图形化用户界面，网络协议系统。目前在全世界已经有 40 余家计算机公司推出了约 200 余种的实时操作系统[1]。

从 20 世纪 80 年代开始，出现了各种嵌入式实时操作系统，这些操作系统主要用于专有系统和开发，形成了现在各种形式的嵌入式实时操作系统相竞争的情形，如 Vxworks，eCos，μClinux 等以及本文将要讨论的 μC/OS-II。

实时系统可以通过一个单处理器系统来实现，也可以用多处理器系统来实现；实时调度是指在有限的系统资源下，为一系列任务决定何时运行，并分配任务运行除 CPU 之外的资源，以保证其时间约束、时序约束和资源约束得到满足。实时调度算法是保障实时系统的时限性和高可靠性的主要算法之一，一直是实时系统研究的热点问题。

因此在实时操作系统中，任务调度是嵌入式实时操作系统的一个重要概念，也是其核心技术之一。针对嵌入式实时操作系统任务的调度的特点，在微内核结构中，大多采用的是基于优先级的可抢占式任务调度算法。操作系统为每一个任务分配一个优先级，调度程序总是选择优先级最高的就绪任务运行[2]。

1.2 国内外研究概况

实时调度算法是实时系统的研究的核心问题之一，它的主要功能是根据可调度条件确定系统现有资源是否满足处理操作的时间确定性。对实时调度的研究开始于二十世纪七十年代中期，在三十多年的时间里，该领域的研究空前活跃，得到了飞速发展。

在我们使用的嵌入式操作系统中，它的实时性是由实时调度策略所决定的。截止目前，有很多的学者都在研究如何提高嵌入式操作系统的实时性，他们中的大多数都在研究实时操作系统的实时调度策略。单调速率算法调度算法、最小裕度优先调度算法，最小松驰时间调度算法、优先级队列调度算法、截止期最近优先调度算法都是现在比较流

行的调度算法，正在被广泛的使用，但是人们对实时任务调度的需求也是与日俱增，这些算法也很难满足人们的需求。

Anderson 等提出基于 EDF 的限制迁移算法[1]，任务的时限是相对自由的，适用于软实时应用。Abdelzaher 等提出了基于许可控制的反馈调度方法[5]。Spuri 和 Buttazzo 对改进了 EDF 算法[6]，提出了基于 EDF 的 5 种新调度算法，它们分别是 DPE、DSS、TBS、EDL 及 DPE 等。为了减少 EDF 算法中任务的响应时间，那么我们就要对它进行改进，国内外有很多的研究人员都在研究这方面的内容，他们提出了一种自适应实时操作系统，它能根据系统的不确定性来自动的改变自己执行任务的时间。还有学者设计了一个实时的容错调度算法[7]，它能调度一些实时任务，有容错和无容错的任务都可以。张杰等通过分析和计算 EDF 算法调度偶发任务所占用的空闲时间和挪用时间，以及调度后对空闲时间和最大可挪用时间的影响，提出一种采用 EDF 算法统一调度硬实时周期任务和偶发任务时的可调度性充分判定算法[8]。萧伟等提出一种改进型 EDF 调度算法[9]，充分利用总线带宽，对总线报文进行最优化调度。罗钧等针对嵌入式实时系统任务调度问题，讨论综合截止期和关键度两种特征参数的任务调度策略，提出一种动态截止期关键度调度算法[10]。国内外学者提出的这些改进算法能在很大程度上提高了实时任务调度的性能，但是还有许多不足之处，如应对复杂的应用需求等。

目前对嵌入式实时操作系统的研究有如下几个方面[11]：

1、改进现有的调度算法

可以在非实时任务的交互性和实时任务调度方面进行改进，内核中任务的响应时间是有调度算法决定的，对调度的改进包括对实时任务的调度改进和交互式非实时任务的改进，调度算法将直接影响到内核对任务的响应时间。目前，实时操作系统中常采用的调度算法有、速率单调调度算法、时限单调调度算法、最早截止期优先调度算法、最小裕度优先调度算法等。

2、改进操作系统的时钟周期

时钟周期是操作系统中最基本的时间单元，它表示这计算机的处理数据的处理能力。系统时间的长度是由时钟中断频率所决定的。所以实时操作系统对时钟频率的要求也是非常高的，要选择合适的时钟频率，过高或者过低都会导致让系统产生很大的开销，那么在选择的时候就要兼顾系统各方面的要求。

3、内核抢占的改进

系统模式和用户模式是内核的主要工作模式。如果工作在系统模式下，那么会拥有更多的权限，而在用户模式下，运行在其中的任务不能对操作系统的数据，只能在进入系统模式的情况下才能进行相应的操作。用户模式的任务是可以被抢占的而系统模式的任务不可以被抢占。如果非实时任务使用着 CPU，但它又不释放资源，那么现在运行的实时任务就不会响应，我们可以在内核中加入抢占点，让实时任务去抢占资源，从而可

以有效的节省相应时间，但是这种方法也存在一些不如意的地方。即也可能被其他任务而非实时任务抢占，使实时任务一直等待。因此可抢占内核要根据具体应用制定具体实现策略。

1.3 论文的主要研究内容

基于目前嵌入式实时操作系统在各个领域的广泛应用，本课题主要研究嵌入式实时操作系统的调度算法。主要工作包括：

1、分析嵌入式实时操作系统的国内外研究概况。分析微内核的体系结构和技术特点，介绍常见嵌入式实时操作系统微内核。研究微内核的实时调度算法，并对其调度算法做分类和总结。最后分析比较静态实时调度和动态实时调度其中的不足之处。

2、对最早时限优先（EDF）调度方法进行分析和研究，给出针对实时性改进的微内核实时调度算法，在基于新算法的设计过程中，首先对任务集建模，然后对改进算法进行描述，包括优先级的分配、实时调度器和中断管理机制。最后制定相应算法流程图。

3、对微内核 μC/OS-II 的调度算法进行详细的分析和研究，包括任务控制块，任务就绪表和任务调度器。分析和总结 μC/OS-II 实时内核优缺点，通过实现时间片轮转算法在内核部分的扩展，优先级继承算法，扩充任务数，改进调度算法，使得构建的实时内核更可靠及具有更强的适用性。

4、针对目前广泛运用的嵌入式微内核 μC/OS-II 存在的不足之处进行改进和实现。并以 μC/OS-II 作为原型，实现新的微内核任务调度算法，通过性能测试实验，证明新的微内核实时性得到提高。

1.4 论文的章节安排

本论文共分六章，具体各章节内容如下：

第 1 章为引言，介绍了本课题的研究背景、研究意义及国内外研究概况，阐述论文主要研究内容。

第 2 章介绍嵌入式实时操作系统及其体系结构。首先介绍了嵌入式操作系统和嵌入式实时操作系统，及嵌入式实时操作系统发展趋势。然后介绍了微内核的相关概念，技术特点和体系结构。重点介绍了微内核的体系结构及常见嵌入式实时操作系统微内核。

第 3 章分析嵌入式实时操作系统微内核任务调度算法。研究了微内核的实时调度算法，并对其调度算法做了分类和总结。最后分析比较了静态实时调度和动态实时调度其中的不足之处。

第 4 章是新的微内核任务调度算法。首先提出问题，并对任务集建模，详细叙述了新的优先级的分配方案。然后讨论了实时调度器和中断管理机制，叙述了新的微内核的调度算法，最后给出了新的微内核任务调度算法的流程图。

第 5 章是新的微内核任务调度算法的实现与测试。首先针对目前广泛运用的嵌入式微内核 μC/OS-II 存在的不足之处进行了改进和实现。并以 μC/OS-II 作为原型，实现新的微内核任务调度算法，通过性能测试实验证明微内核的实时性得到了提高。

第 6 章是总结与展望。对本文的所做的工作进行了系统的总结，提出下一步工作方向和目标。

第 2 章 嵌入式实时操作系统及其体系结构

2.1 嵌入式操作系统

嵌入式操作系统是一种应用于嵌入式系统上的系统软件，是嵌入式系统的核心部分，承担着任务调度、任务控制、任务间通信与同步、资源管理、存储管理等重要任务。嵌入式操作系统具有管理系统资源以及把硬件虚拟化等一般操作系统所具有的特征，它同样能够提供编程所需的库函数，驱动程序等应用程序，使程序员能够较为简单高效的实现程序的移植和维护工作。

相比于一般的操作系统，嵌入式操作系统具有以下特点[3]：

① 体积小。嵌入式操作系统受限于它所应用的环境，它只有少量的闪存作为存储介质。因为没有大容量的存储介质，这就对嵌入式操作系统的大小提出了要求，它只能在有限的闪存中运行，这也使得中断服务受到了影响。因此，嵌入式操作系统必须体积小。

② 实时性。嵌入式系统一般都必须是实时系统，这同时也就是需要对应的嵌入式操作系统必须具有实时性。这也就是要求嵌入式操作系统必须要有效率较高的实时调度算法、可调度性和死锁等。

③ 可裁剪性。根据应用需求对系统的功能模块进行配置是嵌入式操作系统最突出的特点之一。可裁剪性带来的最直接的好处是硬件成本降低，并且操作系统也只包含应用程序需要的功能，使得系统简单，可靠。

④ 固化代码。在嵌入式系统中，嵌入式操作系统和应用软件被固化在嵌入式系统的 ROM 中。辅助存储器在嵌入式系统中很少使用。

⑤ 统一的接口。针对不同的 CPU，如 ARM、PowerPC、x86 等，嵌入式操作系统都提供了统一接口。而且很多的嵌入式操作系统还支持 POSIX 规范，如 Nucleus、Vxworks、RTlinux 等。

⑥ 开发调试环境特殊。编译/连接器、内核调试/跟踪器以及集成图形界面开发平台都是开发一个嵌入式系统所需要的工具。其中调试工具、编辑工具、监视工具和软件仿真工具等也是集成图形界面开发平台所需要的软件。

目前流行的嵌入式操作系统可分为两种：一种是运行在 PC 机上的操作系统移植到嵌入式系统中，开发出的嵌入式操作系统，如微软公司的 Windows CE、朗讯科技公司的 Inferno 和 Linux 等。这一类操作系统经过 PC 机或高性能计算机等的长期运行改进，技术比较成熟，其相关标准和软件开发方式已被用户普遍接受，同时具备多种开发工具和应用软件等资源。另一种是实时操作系统，如 WindRiver 公司 VxWorks、QNX 系统

软件公司的 QNX、ISI 公司的 pSOS、ATI 公司的 Nucleus 以及本论文将讨论的 μC/OS-II 等，这类产品在操作系统的体系结构和实现上都针对应用领域，对实时性和可靠性等性能做了精巧的设计，同时提供独立的、完整的系统开发和测试工具，使其更多地应用在交通、能源、科学研究和工业控制等领域中。

2.2 嵌入式实时操作系统

嵌入式实时操作系统的定义是，当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对系统做出快速响应，并使所有任务实时地、协调一致的运行的嵌入式实时操作系统。

嵌入式实时操作系统对系统的实时性有着很高的要求。尤其在嵌入式技术广泛应用的交通、能源、科学研究和工业控制等领域，对嵌入式操作系统的实时响应能力要求是相当高的，无论时间偏差有多小，都可能造成灾难性的后果。这也是大多嵌入式系统都采用实时操作系统的原因之一。如果说嵌入式系统是以应用为中心的，那么嵌入式操作系统则是嵌入式系统应用中的核心。

2.2.1 嵌入式实时操作系统的特点

嵌入式实时操作系统并不是单纯的嵌入到操作系统中，它不同于一般而言的操作系统。嵌入式实时操作系统同一般的操作系统一样，主要是负责系统所有软、硬件资源的调度和分配工作，使系统的各个部门协调一致的运行。

嵌入式实时操作系统具有如下特点[36]：

① 移植性好。嵌入式系统的硬件平台具有多样性，CPU 芯片更新换代速度快，这就要求嵌入式操作系统对硬件平台具有良好的适应性。嵌入式操作系统对运行平台的要求较小，能够支持一般的运行平台，所以一种嵌入式操作系统要能很方便的运行在不同的硬件平台上，这就要求嵌入式操作系统移植性较好，方便系统的开发以及应用，降低生产成本。

② 内核体积小。由于通常嵌入式系统所能提供的资源有限，并且由于运行平台所具有的 CPU 的处理能力有限、内存容量较小，这就要求嵌入式操作系统应该具有较小规模的内核，一般情况下只有几百 K，甚至是几十 K 字节。

③ 调度抢占机制。当任务执行的时候，有一个优先级更高的任务要求获得 CPU 的控制权的时候，就会抢占当前任务的 CPU，要求当前任务放弃系统资源，从而得以执行。抢占调度机制是实现嵌入式操作系统实时性的基础。

④ 功能模块可裁剪。由于嵌入式设备往往是针对不同的具体应用，它的功能要求也不尽相同，这就需要嵌入式操作系统能很方便的对不同功能需求的支持，为了节省系统资源开销，降低成本，就需要针对不同的嵌入式设备添加相应的功能模块，裁剪掉不需要的功能模块。

⑤ 高可靠性。由于嵌入式操作系统自身特定的用途，决定了它要有很高的可靠性，系统必须尽可能的将出错率降到最低，甚至没有，在飞机的飞行控制等对操作精度要求较高的应用上时，不能有任何的差错，否则会造成不可估量的严重后果。

2.2.2 嵌入式实时操作系统的发展阶段

嵌入式实时操作系统的发展经历了以下四个阶段[36]：

1、无操作系统的嵌入式算法阶段

这一阶段的嵌入式操作系统是基于可编程控制器形式，以单芯片为核心的，并且具有与监测、伺服、指示设备等功能的操作系统。当时的操作系统只是通过汇编语言编写的程序对系统进行控制，任务运行完成之后就对内存进行清除。这一阶段的嵌入式操作系统的主要特点是该系统的体系结构和功能都比较单一、存储容量小、针对性强、处理效率较低、几乎没有用户接口。

由于这种嵌入式操作系统易于使用而且价格相对低廉，所以在当时的国内工业控制领域应用比较普遍，但是这种低效的、存储容量小的系统并不能满足需要高效、大容量存储介质的现代工业和信息家电领域的要求。

2、简单监控式实时操作系统

这一阶段的嵌入式系统是基于嵌入式处理器，以简单监控式操作系统为核心的操作系统，主要用来控制系统负载以及监控应用程序运行。这一阶段操作系统的主要特点是该系统的处理器种类较多而通用性较弱；系统开销小且效率高；配备系统模拟器，具有一定的扩展性和兼容性；用户界面不够友好等。

3、通用的嵌入式实时操作系统阶段

这一阶段的嵌入式系统主要以通用嵌入式操作系统为核心，其主要特点是该系统能运行在多种不同的微处理器上，具有文件管理、目录管理、网络支持等功能的模块化和扩展性。

4、基于互联网的嵌入式操作系统

目前大多数嵌入式操作系统并不能够连入互联网，但是随着互联网和物联网的发展，嵌入式操作系统也更多的与互联网相结合起来。

2.2.3 嵌入式实时操作系统的发展趋势

嵌入式实时操作系统在操作系统的研究中是一个非常重要的部分。随着互联网以及计算机、通信等技术的发展，嵌入式操作系统的发展速度也在不断的提高。

嵌入式操作系统主要分为两种。一种是强实时操作系统，他们主要是面向控制、通信等领域，如 VxWorks、PSOS 以及 Nucleus 等，其中 VxWorks 在国内市场中有较大的影响力。另一种是弱实时操作系统，他们主要是面向消费电子产品，例如手机、PDA、电子书等。其中微软公司的 Windows CE 系统具有较大的影响力。

现在，存在于市场中的实时操作系统分为商业性的和非商业性的两种。非商业性的实时操作系统内核是开源的内核，用户可以在互联网上下载，根据个人的爱好进行修改，如 μClinux 和 μC/OS-II 等。商业性的内核是不提供源代码的，需要购买。如 VxWorks 和 Windows CE。嵌入式实时操作系统的发展趋势如下[38]：

① 嵌入式产品将与互联网应用相互促进，快速发展，嵌入式产品将成为互联网的主要终端之一，网上将出现大量的服务于嵌入式产品的软件，并有专门服务于嵌入式产品的内容。

② 随着微电子技术的快速发展，芯片功能更加强大，SOC 将成趋势，这不仅能降低成本，缩小产品体积，还将增强产品的可靠性。同时，软件硬件的紧密结合，嵌入式软件与硬件界线更加模糊，嵌入式软件时常以硬件形态存在，这种方式可提高实时性，增强可维护性。

③ 无线通讯产品将成为嵌入式软件的重要应用领域，一方面，已有无线产品将借助芯片技术和嵌入式软件来提高性能，另一方面当前许多嵌入式产品都将增加无线通讯功能。因此，未来几年，蓝牙等相关技术会与嵌入式软件相互促进，共同发展，使更多的产品具有通讯功能，使更多的通讯产品更好地为用户服务。

④ 嵌入式操作系统会与嵌入应用软件协同发展。嵌入式系统中的重要角色包括嵌入式应用软件，嵌入式系统应用领域千差万别，只有充分重视应用软件的发展，才能满足丰富多彩的应用要求。

⑤ 嵌入式操作系统是在多种硬件平台上发展起来的，随着嵌入式系统的广泛应用，信息交换、资源共享机会增多，由此相关的标准问题也将日渐突出，如何建立相关标准成为业界关注的问题。

2.3 嵌入式实时操作系统的体系结构

嵌入式实时操作系统主要是针对实时性的，在不同的平台和结构中，所需要的功能和结构是不同的，但是总体理论体系结构是相同的。如需要设备驱动，硬件初始化和处理器相关的硬件抽象层等的支持，在最上层还需要有面向用户的应用程序。

嵌入式实时操作系统主要由硬件抽象层、微内核和系统主要模块等三部分组成。嵌入式实时操作系统的体系结构如图 2.1 所示[52]。一般对于要求较低和功能单一的嵌入式系统可以不使用操作系统，而对于要求功能多样而且功能经常变化、可扩展的嵌入式系统，应选择使用操作系统以简化设计。

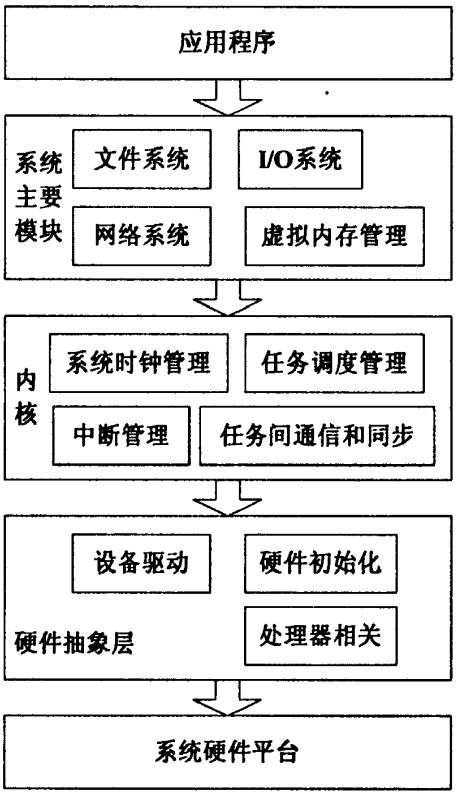

图 2.1 嵌入式实时操作系统的体系结构

1、硬件抽象层。硬件抽象层包含如设备驱动，硬件初始化和处理器相关的与硬件平台相关的模块。它位于实时操作系统体系结构的最底层，主要功能负责与硬件平台间的通信和控制硬件，并对上层的内核提供访问和控制服务。它的优点是可以简化系统内核的移植工作，在移植的时候，除了需要修改设备驱动程序代码外，只需要修改 HAL 的代码。

2、嵌入式实时操作系统内核。它是用来为多数程序和操作系统建立一系列在抽象的文件上工作的抽象机，使用户程序及上层操作系统模块对系统设备透明。

嵌入式实时操作系统内核的实现可以采用微内核技术。微内核技术指的是将必要的功能，如任务调度管理、系统时钟管理、中断管理、任务间通信和同步等放在内核中，而将那些不太重要的服务，如文件系统、网络系统、虚拟内存管理和设备驱动管理等作为内核上可调配的模块。这样，操作系统是由提供必要功能的微内核和其余服务模块组成。

目前主流的嵌入式操作系统都采用微内核结构。微内核使得内部程序精简与模块化，它的优势是良好的可移植性和可裁剪性。在微内核中包括系统时钟管理、任务调度管理、中断管理和任务间通信和同步等基本模块。并且微内核模块可以实现扩展功能，这些功能模块是以运行库的形式提供给上层用户的，这样的微内核短小精悍，具有安全性和可靠性，同时也提高了系统的效率。

3、在提供的嵌入式实时操作系统接口上需要有对用户程序提供的函数接口，专门为用户定制网络、图片、视频等接口，并且提供驱动程序开发界面，方便开发人员对不同需求的设备定制驱动程序。

嵌入式操作系统的体系结构可具有实时性、灵活性、可靠性、可扩展性和可移植性。目前嵌入式操作系统的体系结构可以分为三类，分别是单一内核结构、层次内核结构和微内核结构。

2.3.1 单一内核结构

早期的嵌入式实时操作系统是单一的，也可以说是没有结构的。整个操作系统以过程调用集合的方式编写，只要需要，每一个过程都可以调用任何其他的过程。使用这种技术时，该系统中的每个过程都有一个给定参数和返回结果给出良好定义的接口，而且每个过程都可以自由调用其他过程，只要后者提供了前者所需要的一些有用的计算工作。

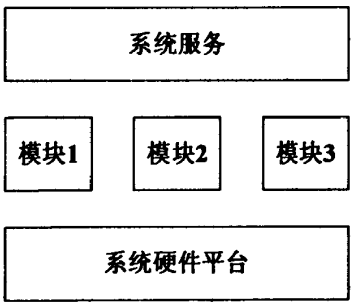

图 2.2 单一内核结构

单一内核结构由若干个模块组成，各个模块协同工作，完成整个系统的功能。系统通过各个模块的对外接口完成模块之间的信息传递工作。模块是透明的，也就是说其他模块看不到它内部的实现过。模块间也可以相互调用。其结构模型如图 2.2 所示。

这种结构的操作系统的结构紧密、接口简单直接、系统的效率相对较高。但是各个模块间转接具有随意性，模块具有较弱的独立性。在对模块进行添加或修改时，很有可能会影响其他模块。而且随着模块数量的增加和模块之间通信的越来越频繁，系统会显得更加混乱，任何一个错误都可能带来严重的后果，使得系统难以恢复。

2.3.2 层次内核结构

随着操作系统不断增大，同时要弥补单一结构中模块间调用存在的不足之处，人们大量研究应当构造何种最优结构。层次结构就是为了克服单一内核结构的不足而发展起来的，它改变了模块间的调用关系，由之前的无序变为了有序的结构，那么模块间的调用关系变的有规则了。

操作系统的分层次的调用各个模块，这就是内核的层次结构，它使各个模块的通信方式改变了，系统的设计使用了层次化的方式，每一层都使用下层提供的服务并且为上层提供服务。层与层间通过单向调用的方式通信，同层模块之间不存在互相调用的关系。不过在某些嵌入式操作系统的实际应用中，也会出现同层模块之间互相调用的情况。其结构如图 2.3 所示。

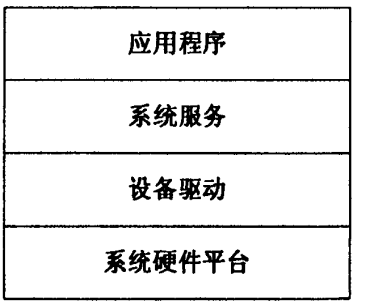

图 2.3 分层内核结构

层次内核结构具有模块间的调用和依赖关系非常清晰等优点，下层模块为上层模块提供了良好的基础，可靠性和可读性都得到了加强，更适应于各种实际应用。修改或替换内核某层时，受到影响的只有邻近两层，易于模块的扩充和修改。此外，为了适用于更广泛的应用系统，常把与系统硬件联系紧密的模块，如 I/O 管理和中断处理等放在最底层，将最常用的模块放在次底层，而把随着最底层和次底层模块操作更改而变化的模块放在最上层。

2.3.3 微内核结构

微内核是在内核中只保留了最核心的功能，如任务调度管理、系统时钟管理、中断管理、任务间通信和同步等几个组成部分，使内核尽量的小，而把那些不太重要的服务和功能，如文件系统、网络系统、虚拟内存管理和设备驱动管理放到用户模式运行。其结构模型如图 2.4 所示[54]。

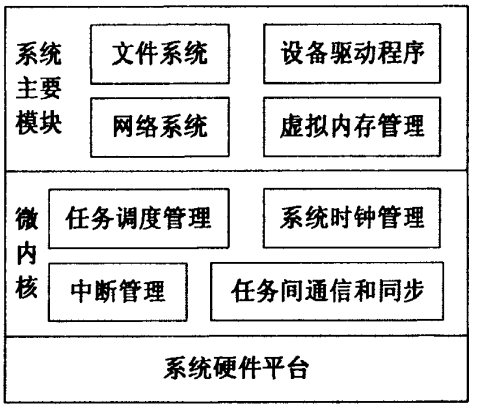

图 2.4 微内核结构

微内核的目标是将系统服务的实现和系统的基本操作规则分离开来。例如，进程的输入/输出锁定服务可以由运行在微核之外的一个服务组件来提供。这些非常模块化的用户态服务用于完成操作系统中比较高级的操作，这样的设计使内核中最核心的部分的设计更简单。一个服务组件的失效并不会导致整个系统的崩溃，内核需要做的，仅仅是重新启动这个组件，而不必影响其它的部分。

微内核操作系统具有良好的灵活性。它使得操作系统内部结构简单清晰。程序代码的维护非常之方便。但是也有不足之处。微内核系统由于核心态只实现了最基本的系统操作，这样内核以外的外部程序之间由于独立运行使得系统难以进行良好的整体优化。另外，进程间互相通信的开销也较单一内核系统要大许多。从整体上看，在当前的硬件条件下，微内核在效率上的损失小于其在结构上获得的收益，故而选取微内核成为操作系统的一大潮流。

2.3.3.1 微内核的特点

操作系统研究领域最近十几年突出的成就应该是微内核技术。微内核的研究动机是为克服已有的操作系统内核由于功能的增加而逐渐变大的缺点。微内核体系结构具有如下特点[34]：

① 内核非常小，一般只有几十到几百 KB。

② 稳定性。微内核多是在内存中对数据进行简单地读入，读出操作，微内核封装的也是最基本的核心服务，因此微内核经过精心设计和测试，具有极高的稳定性。一个外部模块的异常，不会影响到其它模块的运行。

③ 可移植性。微内核具有极高的移植性，由于与目标硬件相关部分被放到微内核的底层部分和驱动程序中，当需要移植到新的软件或硬件环境中时，只需对微内核中硬件相关的部分稍加修改，易于实现不同平台间的移植。

④ 灵活性和扩展性。微内核具有很好的灵活性和扩展性，这是微内核结构的最大优点之一。在不影响系统应有的处理功能情况下，可以随意增加或减少外部模块，来扩展或舍弃功能，而不会影响已存在的模块运行。

⑤ 基于客户/服务器体系结构。在微内核结构的操作系统中，任务间通信机制是系统的基础，操作系统的各种功能都以服务器方式实现，向用户提供服务。用户对服务器的请求是以消息传递的方式传给服务器的。

⑥ 有一个硬件抽象层，内核能方便地移植到其它的硬件体系结构中。因为当需要移植到新的软件或硬件环境中时，只需对与硬件相关的部分稍加修改即可把微内核嵌入到新的硬件环境中，在多数情况下并不需要移植外部服务器或客户应用。

2.3.3.2 常见嵌入式实时操作系统的微内核介绍及其比较

① VxWorks[37]

VxWorks 是美国 WindRiver 公司开发的一款嵌入式实时操作系统，从上世纪 80 年代进入嵌入式实时操作系统市场以来，凭借高性能的实时内核、持续的发展潜力及独立的用户开发环境等特性，被广泛的用于嵌入式实时操作系统领域。它由 400 多个短小精悍并且相互独立的模块组成，用户可以根据自己的需要选择适当的模块来裁剪和配置系统；提供基于优先级的任务调度、多任务通信、多任务同步、中断处理、定时器和内存管理等操作系统必备的基本功能，同时也能实现了符合 UNIX 可移植操作系统接口规范的内存管理，以及多处理器控制程序；还可以具有简明易懂的用户接口。如果只保留核心模块的话，该操作系统甚至可以压缩到 8KB。

② μC/OS-II[39]

μC/OS-II 是由美国 Micrium 公司开发的一款实时操作系统。它是一个源码开放、可移植、可固化、可裁剪、具有可剥夺实时内核的实时多任务的操作系统。μC/OS-II 已经在各个领域得到广泛的应用，包括医疗设备、网络设备、自动取款机、机器人及工业控制等。其实时性和稳定性已在大量的实际应用中得到了确认。它能同时运行 64 个任务，并实现了任务调度管理、内存管理、任务间同步与通信、时间管理和中断服务等功能。并且具有执行效率高、占用空间小、实时性能优良和可扩展性强等优势。基于上述特性

也使得成 μC/OS-II 特别适合于对实时操作系统的学习、研究和开发。另外，对于高校科研的应用，该内核是完全免费的，其迅猛的发展也证明了开放源码软件的巨大生命力。

③ μClinux[12]

随着嵌入式应用的日益普及，人们迫切需要更加小巧的、无需庞大内存运行环境的迷你型的操作系统，GPL 组织开发了针对微型控制领域的 Linux 操作系统，这就是 μClinux 操作系统。μClinux（micro-control Linux）是嵌入式 Linux 中优秀的代表。虽然 μClinux 内核和通用 Linux 的比要小很多，但它仍然继承了通用 Linux 操作系统的最主要特性，尤其继承了良好的稳定性和移植性、强大的网络功能、出色的文件系统支持、标准丰富的 API，以及 TCP/IP 网络协议等特点。值得注意的一点是它没有内存管理单元，所以在 μClinux 上实现多任务机制时，需要撰写更多的代码。

④ eCos[12]

eCos（embedded Configurable operating system），是 Redhat 公司于 1997 年开发的一个开源、可移植、可配置、可裁剪、高度集成的嵌入式实时操作系统，其全部代码使用面向对象语言 C++编写。由于采用了模块化设计，它的核心部分由各种不同的组件构成，包括内核、C 语言库和底层运行包等，其中每个组件都可以提供灵活的配置。目前发行的 eCos 版本采用的任务调度方法主要有位图调度和多队列调度两种方法，eCos 还支持对称多处理器 SMP。位图调度和多队列调度都是基于优先级的抢占调度，另外用户还可以根据需要在系统配置的时候加入自己的调度方法，非常方便。

eCos 最大的特点是内核可配置，可实现原代码的配置，并且同时支持命令符和图形界面配置环境，它可以配置的很小，最小可以配置裁剪到几 K。eCos 采用模块化设计，由不同的功能组件构成。它最大的优势就是配置选项较多，给开发者提供了更多的选择。

表 2.1 四种常间嵌入式实时操作系统微内核任务调度策略的比较

操作系统		任务数量	优先级数	优先级变化	是否同优先级调度	内核是否抢占式	调度方式	时间是否可确定
VxWorks		256	256	动态	有	是	基于优先级抢占式调度	是
μC/OS-II		64	64	动态	无	是	基于固定优先级抢占式调度	是
μClinux		无限制	100	动态	有	否	实时任务先来先服务	否
eCos	位图调度器	32	32	动态	无	是	基于固定优先级抢占式调度	是
	多级队列调度器	无限制	32	动态	有	是	基于优先级抢占式调度	是

表 2.1 对四种常见嵌入式实时操作系统微内核任务调度策略的进行比较[37]。

综上所述，嵌入式实时操作系统在嵌入式系统的应用非常广泛，但是又具有各自的特点。由以上介绍的嵌入式操作系统微内核可以看到目前嵌入式实时操作系统种类较多，而且都各有所长。有的操作系统专用性强，其代码比较精简，体积小，微内核，在硬件资源很少的情况下，依然可以满足实时性要求。有的操作系统通用性好，其功能比较复杂，可以支持网络环境等。

2.4 本章小结

本章首先介绍了嵌入式操作系统和嵌入式实时操作系统，及嵌入式实时操作系统发展趋势。然后介绍了微内核的相关概念，技术特点和体系结构。重点介绍了微内核的体系结构，及常见的嵌入式实时操作系统微内核，其中包括 Vxworks，μC/OS-II，μClinux 等，比较了 VxWorks，μC/OS-II，μClinux，eCos 四种常见嵌入式实时操作系统微内核任务调度策略。

第3章 嵌入式实时操作系统微内核任务调度算法分析

3.1 实时任务调度算法

实时任务是指任务的结束时间有严格约束即任务执行必须在时限内完成。将进入系统的任务按实时性分为实时任务和非实时任务。本论文主要针对实时任务进行讨论，非实时任务不做讨论。实时任务调度算法方法有多种分类方法，下面给出了常见的几种分类[11]。

1、按照任务的实时性要求，分为硬实时任务调度和软实时任务调度。

硬实时任务调度要求进入系统的任务必须在规定的响应时间内完成，否则可能导致灾难性后果。硬实时任务的调度实时性必需保证。

软实时任务调度虽然也有一个截止期，并希望满足该截止期的要求，但并不是强制的，即使过了最后期限，任务的执行仍然是有意义的，不会给系统带来致命的错误。其目的是使各个任务尽快完成，对响应时间有一定的灵活性，在一定的时限范围内，时间的延迟是可以接受的。

2、按照任务的执行序列生成的时间和方式，分为静态调度和动态调度。

静态调度是在编译的时候就确定就绪任务的执行顺序，这类调度需要预先知道任务的实时特征参数。

动态调度是在系统运行过程中根据实际情况决定就绪任务的执行顺序，这类调度不需要预先知道任务的实时特征参数。

3、按照被调度的任务是否可以被抢占，分为抢占式调度和非抢占式调度。

在抢占式调度中，当前正在运行的任务会被其它更高优先级的就绪任务中断，这种调度方法灵活性高，利于实时性实现。

在非抢占式调度中，某个任务一旦占据 CPU 资源，就不会中途被中断，直至执行完毕，这种调度方法灵活性差，但资源利用率高。

4、按照任务到达时刻规律的不同，分为周期性调度和非周期性调度。

周期性调度的任务到达时间间隔是确定的；非周期性调度的任务到达时间间隔是不确定的，或者偶然的。

5、按照实时系统运行环境的不同，分为单处理机调度和多处理机调度。

单处理机调度主要运行在单处理机环境下；多处理机调度主要运行在多处理机环境下。

6、按照调度器体系结构的不同，分为集中式调度和分布式调度。

集中式调度是由主机负责调度的；分布式调度是根据局部范围内的结点主机的负载信息来进行调度的，不再有一个集中的调度主机。

实时任务调度分类如表 3.1 所示。

表 3.1 实时任务调度分类

分类标准	分类结果
任务的实时性	硬实时任务调度、软实时任务调度
任务的执行序列生成方式	静态调度、动态调度
任务是否可以被抢占	抢占式调度、非抢占式调度
任务到达时刻的规律	周期性调度、非周期性调度
实时系统运行环境	单处理机调度、多处理机调度
调度器体系结构	集中式调度、分布式调度

3.1.1 静态实时调度算法

静态实时调度算法是在系统编译的时候就确定任务的就绪队列，同时产生一个任务调度表，并以此来安排任务的调度顺序，系统运行时调度器严格按照任务调度表所提供的信息来调度任务。

静态实时调度算法主要包括：先来先服务（First Come First Serve, FCFS）调度算法、时限单调（Deadline-monotonic, DM）调度算法和速率单调（Rate-monotonic, RM）调度算法等。目前已经证明，速率单调调度算法是最优的静态调度算法。

1、FCFS 调度算法

FCFS 调度算法是指对于按先后顺序到达的任务，开始运行的任务将一直占用处理器，直到任务运行完成，才开始运行下一个任务。FCFS 调度算法可以用于周期任务的调度，对于相同优先级的任务，采用先来先服务进行任务轮转。同时也可以用于非周期任务的调度。FCFS 调度算法的模型为图 3.1 所示。

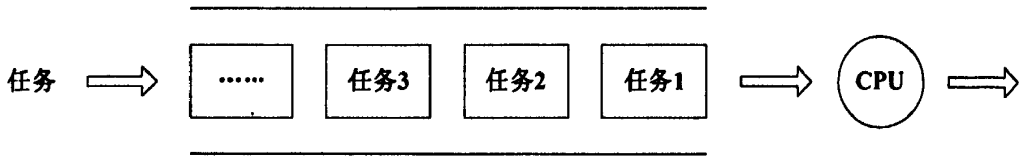

图 3.1 FCFS 调度算法的模型

FCFS 调度算法是一种最普遍和最简单的方法。它优先考虑在系统中等待时间最长的任务，而不管要求运行时间的长短。此调度算法使响应时间的变化较小，因此它比其

他大多数调度都可预测，由于这种调度算法不能保证良好的响应时间，在处理交互式用户时很少用这种调度算法。

2、RM 调度算法

RM 调度算法是一种非常经典的静态调度算法，该算法是根据不同的任务周期灵活地给每个任务分配一个确定的优先级。RM 调度算法首先通过对任务周期的长短排序的方法为每一个任务指定一个优先级，任务的周期长短越与其优先级高级成反比，余下的各个任务的优先级顺序排列。RM 调度算法有一个明显的特点，即任务的可调度性通过计算系统资源的利用率来体现，即任务集能否满足时限通过任务优先级、任务时限、最坏执行时间、任务周期等来验证。

RM 调度算法对实时任务做出三种假设：

① 所有实时任务都是已知周期的；

② 各任务之间没有资源共享、数据交换等问题，是独立的；

③ 采用可抢占式调度算法。

根据 RM 调度算法，设相互独立、非抢占的实时任务构成的任务集 $S=(r_1,r_2,\cdots,r_n)$，相应的任务周期分别是 $T_1, T_2,\cdots, T_n$，任务的执行时间为 $C_1, C_2,\cdots, C_n$。L(n)为 n 个任务的 CPU 利用率的最小上界，对于随机的 T_i 和 C_i，$L(n)=n(2^{1/n}-1)$。公式（3.1）已被证明是 RM 算法可调度的充分条件，一旦这个任务集的 CPU 利用率满足此式，那么就说明它可以用 RM 调度。

$$\frac{C_1}{T_1}+\frac{C_2}{T_2}+\cdots\frac{C_n}{T_n}\le L(n) \qquad \text{公式（3.1）}$$

表 3.2 列举出了任务数为 1 到 9 及无限个任务根据公式（3.1）所计算出的处理器利用率的最小上界 L(n)。当任务数 $n\to\infty$ 时，最小上界 L(n)的值趋近于 0.693；当 n 为 2 时 L(n)是 0.828；若任意两个任务的周期相差整数倍，则 L(n)可达到 1。对于一组任务来讲，所有任务在可调度的前提下对处理器的利用率不会超过相应的界限值。

表 3.2 处理器利用率最小上界

任务数 n	最小上界 L(n)	任务数 n	最小上界 L(n)
1	1.000	6	0.734
2	0.828	7	0.728
3	0.779	8	0.724
4	0.756	9	0.720
5	0.734	∞	0.693

从以上分析可以看出RM算法的潜在处理器利用率较低，因为RM调度算法没有考虑任务的等待时间，使有些周期长但急需要执行的任务无法被调度。但在实际系统开发中，开发人员经常选用此算法，因为它也具有自己的优势：

① 具有较小的调度开销，这是最重要的一点。

② 具有较高的处理器平均利用率。但实践表明，RM 调度算法可调度处理器利用率高于0.9的周期任务集，并且这不是特殊情况下得到的数据，处理器的平均利用率也能达到0.88。

③ 稳定性较高。即使系统出现超载等异常原因，对所有任务的时限要求不能满足时，系统对一些它可以调度的基本的或重要的任务要保证其时限要求。在RM调度算法中，通过缩短基本任务的周期或修改算法的优先级来保证任务顺利实现。

3、DM 调度算法

DM 调度算法是另一种事先把优先级固定的调度算法，任务的时限决定优先级的高低。并且时限大小与它的优先级成反比。优先级较高的任务通过抢占优先级较低的任务的 CPU 资源来保证急需处理的任务能提前调度，保证了系统的实时性。它的调度模型与RM调度模型相同。

3.1.2 动态实时调度算法

动态调度是在系统运行过程中根据实际情况决定就绪任务的执行顺序，这类调度算法主要有：最早截止期优先（Earliest-deadline First，EDF）调度算法、最小裕度优先（Least Slack First，LSF）调度算法等。最早截止期优先调度算法已被证明是动态最优的单一处理器调度算法[10]。

1、EDF 调度算法

EDF 调度算法[6]为最早截止期的周期任务分配一个最高优先级，在每一个可调度时刻，根据任务的最早截止期限动态改变任务的优先级，处理器总是运行所有任务中最早达到最后时限的任务。显然，为了有效的进行调度，EDF 调度算法必须基于抢占式调度。对于一个抢占式调度而言，EDF 调度算法的调度原则可以使得超过最后期限的任务数最少。此算法是根据任务的资源需求动态地分配任务的优先级，达到在资源分配和调度时有更大灵活性的目的。

EDF 调度算法的实现步骤如下：

① 事件发生时，对应的任务进入就绪队列。

② 比较所有任务截至期限，根据任务截至期限进行排序，将具有截至期限最近的任务给予最高优先级，形成优先队列。

③ 执行优先队列队首的任务。

设各任务相互独立的任务集$S=(r_1,r_2,\cdots,r_n)$，相应的任务周期分别是$T_1, T_2,\cdots, T_n$，任务的执行时间为$C_1, C_2, \cdots, C_n$，式（3.2）已被证明是EDF算法可调度的充分条件。如果任务集中所有任务的CPU利用率U之和满足公式（3.2）的条件，则这个任务集被证明用EDF调度是可行的。EDF作为一种动态优先级调度算法是最优的，即对于在任何其他动态优先级算法下可调度的任务集，在EDF算法下也是可调度的。

$$U=\sum_{i=1}^{n}\frac{C_i}{T_i}\leq 1 \qquad \text{公式（3.2）}$$

EDF调度算法的任务优先级表示为$d_i(t)-t$，其中$d_i(t)$表示任务时限，t表示系统当前时间，任务时限与系统当前时间的差值表示任务的紧迫程度。任务时限越小，任务的优先级也就越高，反之任务时限越大，任务的优先级也就越低。系统在每个单位时间，都要计算出下一刻系统中时限最小的任务，并调度此任务，所以系统在下一刻调度哪个任务具有不确定性，使得当前算法能较好的适应系统。

虽然EDF调度算法是公认的动态最优调度算法，但是它不能解决任务过载问题，在超载时系统性能不稳定[9]。如果一个实时任务在时限内结束运行，那么在剩余空闲时间里其它实时任务也不能运行，此时CPU是空闲的；如果一个实时任务没有在时限内结束运行，那么它将继续以高优先级运行下去，这样就会使得就绪队列后面的任务比预定的时间晚运行，形成多米诺效应，使得大量任务超出时限。在实际应用中发生过载情况时，会导致CPU时间大量花费在任务调度上，性能退化很快，不能满足系统的实时性要求。

EDF调度算法的优势在于效率高、容易计算和推断，调度的实现相对容易。但是EDF调度算法在实际应用中也存在过载情况下会出现不稳定、优先级翻转、动态调度系统开销太大等一些问题。

2、LSF调度算法

目前应用最为广泛的一种典型的动态优先级调度算法是LSF调度算法。LSF调度算法调度时根据任务的裕度（即空闲时间）的多少来动态分配优先级。裕度越小，说明该任务就越紧急，越需要尽快执行，优先级越高，反之优先级越低，这样便保证了紧急任务的优先执行。

设各任务相互独立的任务集$S=(r_1,r_2,\cdots,r_n)$，相应的任务周期分别是$T_1, T_2,\cdots, T_n$，任务的执行时间为$C_1, C_2, \cdots, C_n$，公式（3.3）已被证明是LSF调度算法可调度的充分条件。LSF调度算法可调度条件与EDF调度算法相同。如果任务集中所有任务的CPU利用率U之和满足式（3.3）的条件，则这个任务集被证明用LSF调度是可行的。

$$U=\sum_{i=1}^{n}\frac{C_i}{T_i}\leq 1 \qquad \text{公式（3.3）}$$

对比公式（3.1）和公式（3.3）可以看出，由于动态调度算法可以实时调整资源合理分配，动态调度算法的系统利用率比静态调度算法的系统利用率高，但是 LSF 调度算法是根据任务裕度这个唯一的时间参数来确定优先级的，并且要精确测量各个时间值，大大的影响算法的性能。

理论上，单处理器下的最优调度策略有两种，分别是 EDF 调度算法和 LSF 调度算法。由于 EDF 和 LSF 根据计算结果改变任务的优先级，在每个调度时刻都要计算任务时限或任务裕度，因此开销大，应用受到一定的限制。

3.1.3 静态实时调度算法与动态实时调度算法的比较

静态调度算法适用于需求明确的应用领域，如工业控制等。它的优点是可预测性好、稳定性较强和额外开销较小，如果证明任务是可调度的，便能确保所有的任务在时限内都能运行完成，所以得到了广泛的应用。它的缺点是不够灵活，难以适合于不可预测环境下的调度。由于静态调度算法一旦开始调度，调度器严格按照任务调度表来调度任务，调度顺序再也无法更改。

动态调度算法能根据系统的变化及时做出反应，它的优点是比较灵活，适用于任务不断生成的环境。由于动态调度算法只是考虑就绪队列中现有的任务特性，当前的调度任务的序列由队列中的任务来决定，没有考虑到即将要到达的任务的特性。它的缺点是动态调度算法的可预测性差且运行开销较大。

3.2 嵌入式实时操作系统任务调度算法

调度算法是嵌入式实时操作系统的核心所在，因此对嵌入式实时操作系统调度算法进行研究和分析，以提高实时性有着重要的意义。若要确保各个实时任务能够及时、安全完成调度，就必须对任务集在处理器上进行合理的调度，如何对实时任务进行调度同时又保障实时系统时限性和高可靠性，是实时操作系统研究的一个关键问题。不同类型的实时系统对资源要求不同，而且对时间约束的形式也有一定的差异，因此其调度算法就不同，如周期性的任务对周期的设计要求很高，非周期性任务要求截至期比较多，调度算法的关键是系统能否实时响应。

嵌入式实时调度算法主要可以分为三类：时间片轮转调度、优先级抢占式调度、CPU 使用比例共享调度。

3.2.1 时间片轮转调度算法

时间片轮转调度算法分配给处于就绪队列的所有任务一个相同的执行时间片，在当前任务已经结束或当前任务的时间片已经用完的情况下，内核把 CUP 控制权交给下一个就绪任务。

可以输入一个参数来指定时间片轮转调度时间片的长度，在系统中使用的是KernelTimeSlice( )函数，任务的运行方式就是时间片轮转方式。每个任务都有一个时间计数器，当任务运行时，时间片计数器加 1，这样达到了任务的时间控制功能。当任务运行了一个时间片之后进行任务之间的切换，当前运行的任务停止执行，将其放入就绪队列尾部，等待下一次的调度，而且将运行时间计数器清零，并开始执行就绪队列中的下一个任务。当运行任务被更高优先级的任务抢占时，此任务的运行时间计数器被保存，直到该任务下次运行。时间片轮转调度算法如图 3.2 所示。

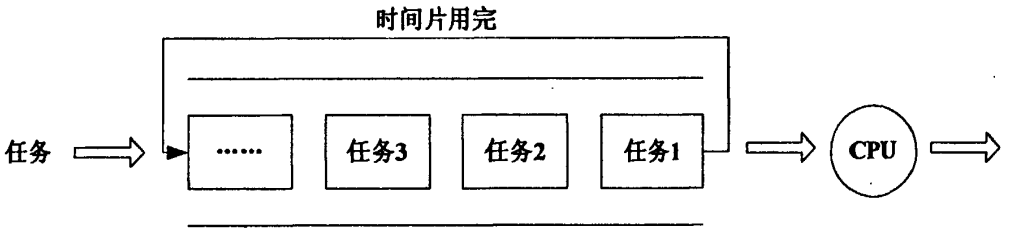

图 3.2 时间片轮转调度算法

在时间片轮转调度算法中，时间片长度的选取非常重要，时间片长度的选择会直接影响系统开销和响应时间，如果时间片长度过短，则调度任务抢占处理机的次数增多，这将使任务上下文切换次数也大大增加，加重了系统开销。如果时间片长度选择过大，大到一个任务足以完成其全部运行工作所需的时间，那么时间片足以完成其全部运行工作所需的时间，那么时间片轮转调度算法就退化为 FCFS 调度算法。最佳的时间片量值应能使响应时间达到最短。

3.2.2 优先级抢占式调度算法

优先级抢占式调度算法是一旦较高优先级的任务处于就绪态，它就剥夺当前较低优先级任务的 CUP 使用权。它的实现机理是：如果中断服务程序使得一个高优先级的任务进入就绪队列，那么中断服务结束后，被中断的低优先级任务被挂起，高优先级的任务使用 CPU。优先级抢占式调度如图 3.3 所示。

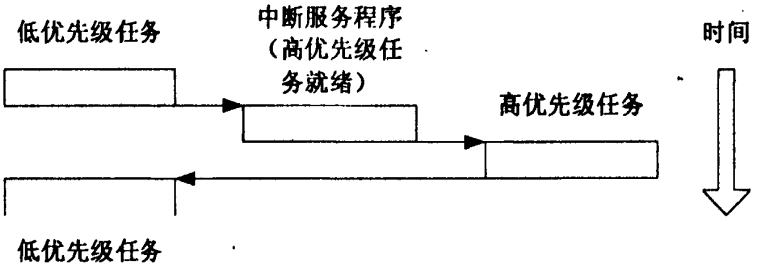

图 3.3 优先级抢占式调度

优先级抢占式调度算法的优点是实时任务的响应时间可以达到最小，缺点是系统不能直接调用不可调入函数，因为如果不可调入函数被低优先级的任务调用，此时，高优先级的任务剥夺 CUP 的控制权，可能会损坏函数。μC/OS-II 采用的就是优先级抢占式调度。

优先级抢占式调度算法更适用于调度硬实时任务，这种调度算法并不适合很多嵌入式系统，例如软实时任务更适合使用 CPU 比例共享调度算法。

3.2.3 CPU 使用比例共享调度算法

CPU 使用比例共享调度算法就是按照某个比例调度任务队列中的任务，使任务的执行时间与其权重成正比，是一种加权轮转调度。

CPU 可以通过两种方式实现比例共享调度算法。一种是调节就绪任务在就绪队列队首的出现频率，在执行任务时调度就绪队列中队首的任务；另一种是按顺序调度就绪队列中的每一个任务，在任务的运行过程中，根据时间分配的比例调节每个任务运行时间片的分配。

这个调度算法有一个缺点，就是没有定义优先级，所有任务的调度次序都是以它们申请的 CPU 使用比例作为依据进行资源共享，这样的话，假如系统出现过载状态，所有任务运行都会变慢，而且变慢是按照比例进行的。所以在实际的应用当中，常会采用到调度算法是动态调节任务权重，可以保证实时系统中的任务能够获得一定的 CPU 处理时间。

以上介绍的三类嵌入式实时操作系统任务调度算法各有优缺点，时间片轮转调度、优先级抢占式调度侧重于硬实时任务，CPU 使用比例共享调度更为适合于软实时任务，在网络系统中应用较多。

在嵌入式操作系统中，任务调度算法通常采用其中一种调度算法或其中几种调度算法结合的方式，如在 Linux 中，采用了时间片轮转和 CPU 使用比例共享两种调度算法。在嵌入式操作系统设计中，应该根据实际系统的具体需求进行优化选择。

3.3 实时任务调度算法的性能评价标准

实时任务调度算法有多种性能评估标准[11]，其中常见的有：响应时间、生存时间、任务吞吐量、任务的接受率、资源利用率、可调度性、调度的复杂性、容错性和可扩展性。

① 响应时间：实时系统从识别出一个外部事件到做出响应的时间。

② 生存时间：数据的有效等待时间，数据只有在这段时间内才是有效的。

③ 任务吞吐量：在单位时间内处理的任务总数。

④ 任务的接受率：指到达系统的任务可以被调度的比率。调度的任务总数与到达系统的任务总数相除得到。

⑤ 资源利用率：指的是系统中所有任务的资源利用率之和，用于描述任务占用资源的比率。用实时任务的最坏执行时间与实时任务活动时间相除得到。资源利用率决定单位时间内可调度的任务，即系统的吞吐量。

⑥ 可调度性：指的是调度算法能够保证实时任务的所有实例均能满足其时限要求。

⑦ 容错性：指处理器发生故障后任务可以恢复运行的能力。

⑧ 调度的复杂性：指实时调度的各种复杂度，也就是复杂性之和。

微内核在实时操作系统中起着重要作用，其性能的好坏将直接影响到整个实时操作系统的性能。嵌入式实时操作系统微内核的时间性能指标主要通过三个性能指标来衡量系统的实时性，即响应时间、生存时间和任务吞吐量。其中响应时间是系统实时性最直观，最重要的指标。而系统响应时间与中断延迟、调度延迟和任务切换时间都有关系。

1、中断延迟

中断延迟是指当高优先级任务发出中断请求，CPU 可能正在处理另一个中断请求，处理完上一个中断请求，再处理此时的中断请求，因此中断请求会延迟一段时间。中断延迟的长短由系统关闭中断时间的长短决定。

在 Linux 中，部分微内核程序是不能中断的，当任务进入到微内核时，它将运行直到完成或者被阻塞，对于实时系统来说，这种长的延时是不能接受的。在 μC/OS-II 中，系统允许中断嵌套，当中断处理时不需要关中断，关中断主要发生在一些原子操作和代码临界区保护的时候，并且都非常短，因此 μC/OS-II 的中断延迟很短。

2、调度延迟

调度延迟是指中断处理结束到任务被调度完成且完成任务切换的这段时间。在中断结束后发生的首次调度中，如果目标任务能被选中而进入运行，那样调度延迟的大小只取决于调度算法的复杂，以及系统中就绪任务的多少；但是如果不能选中，调度延迟就不能估计了。

在 Linux 中，非抢占式内核决定了系统的调度延迟的时间比较长且难以预测。μC/OS-II是基于优先级的可抢占式调度，而且内核的调度算法非常简单，因此在μC/OS-II中，调度延迟比较短且可以预测，适应实时应用的要求。如果要使重要紧急的任务一经唤醒便被优先调度运行，系统就必须要有基于优先级的可抢占式调度策略。这样才能保证调度延迟是可以估计的。

3、上下文切换时间

任务切换的实现是把当前任务的上下文保存到堆栈中，而把将要运行的任务从堆栈中进行上下文恢复。因此，任务的切换时间主要与 CPU 的寄存器数量和操作系统的实现有关。在μC/OS-II 中，任务都有单独的堆栈，因而任务的切换操作非常简单，用十多条 CPU 指令就可完成，因此μC/OS-II 任务切换产生的延迟很小且是可以预测的。

3.4 本章小结

本章首先介绍了实时系统的分类及实时任务调度算法，并对静态实时调度（FCFS 调度算法、DM 调度算法、RM 调度算法等）和动态实时调度（EDF 调度算法、LSF 调度算法）进行详细讨论及比较两者的优缺点。然后分别讨论了时间片轮转调度、优先级抢占式调度、CPU 使用比例共享调度。最后给出实时调度算法的性能评价标准。

第 4 章 新的微内核任务调度算法

4.1 问题的提出

在嵌入式实时操作系统中，为了保证实时的需要，最重要的目标就是满足系统内实时任务的时间约束，合理地调度并执行它们，使其在各自的时限内完成。这就使实时任务调度成为嵌入式实时操作系统的核心问题之一，同时也是计算机科学领域研究的热点问题之一。

任务调度是微内核的主要职能之一，调度算法的好坏起着至关重要的作用。大多数实时微内核使用的是基于优先级抢占式任务调度算法。系统为每一个任务分配一个优先级，调度程序总是选择优先级最高的就绪任务运行。内核运行中频繁地进行任务调度，且任务调度属于系统的临界资源，调度时需要独占 CPU，不允许外部中断和任务切换。所以调度速度缓慢时，会影响整个系统的响应速度和处理能力。因此，关于最高优先级就绪任务的查找算法是实时系统的核心内容之一。

为提高微内核任务调度的实时性，本论文基于以上研究，设计一种新的微内核任务调度算法，就优先级分配、任务调度器、中断管理机制、任务调度算法流程等方面进行设计。任务调度的状态图如图 4.1 所示，核心模块中的任务调度算法是新的微内核任务调度算法的关键。

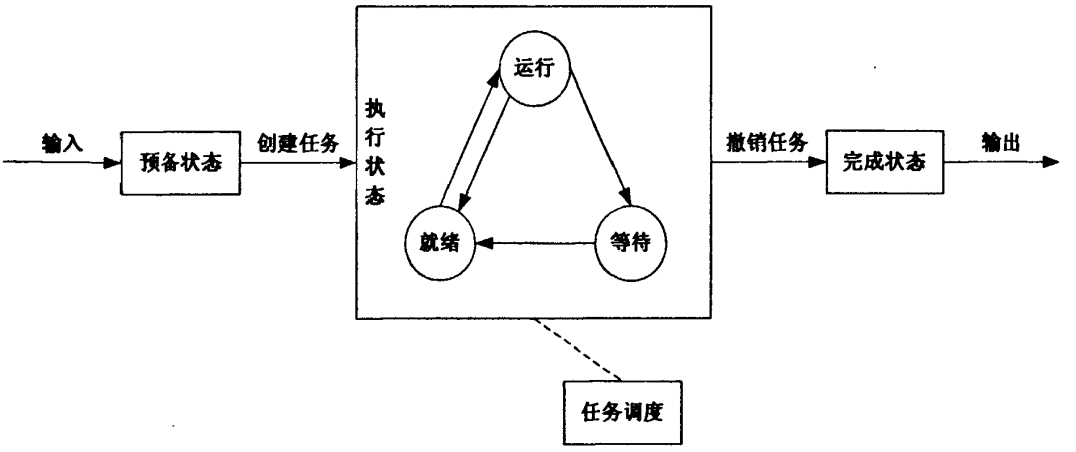

图 4.1 任务调度状态图

4.2 任务模型

假设任务集 S，且各任务相互独立，$S=(r_1,r_2,\cdots,r_n)$，相应的任务周期分别是 T_1，T_2，…，T_n，任务的执行时间为 C_1，C_2，…，C_n，任务到达的时间 A_1，A_2，…，A_n。任务的时限为 $DTime_1$，$DTime_2$，…，$DTime_n$，任务的价值为 V_1，V_2，…，V_n，任务的时间片为 S_1，S_2，…，S_n，任务在就绪队列中的优先级为 P_1，P_2，…，P_n。

在新的任务调度算法中进行如下假设：

① 所有的任务具有周期性的，且周期大小确定；

② 所有任务的相对截止时间与他们的运行周期相等，这意味着每个任务必须在它下一次启动前完成；

③ 各个任务彼此完全独立，任务的请求不依赖于其他任务的状态；

④ 系统开销、中断处理时间和上下文切换的时间可以忽略；

⑤ 所有任务都是可被抢占的；

⑥ 调度算法的程序运行在单处理器上；

由于调度系统需要实时更新任务的价值、时限和优先级，所以整个调度系统由优先级计算单元、调度器和采集器组成，如图 4.2 所示。

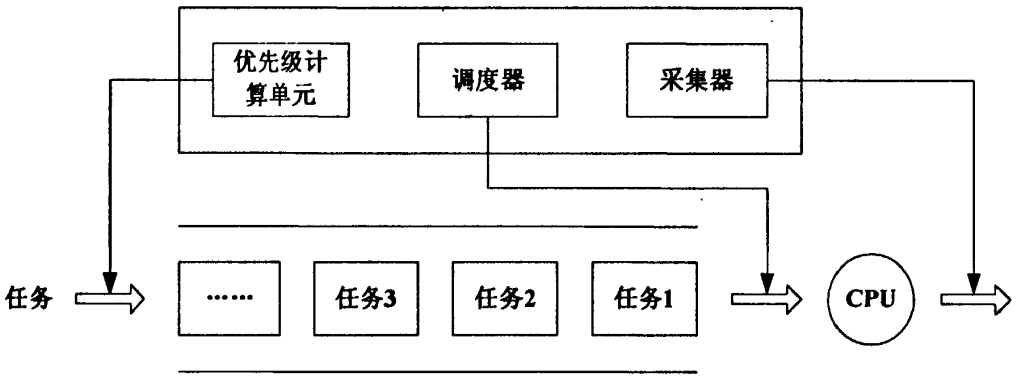

图 4.2 调度系统结构图

其中，优先级计算单元负责根据采集器采集的结果实时更新所有任务的优先级，并按优先级顺序排列就绪任务；调度器负责按照一定的全局调度算法，实时调度任务的执行；采集器负责对影响任务优先级的诸多因素进行实时采集，并根据采集结果实时更新任务的执行价值等参数，使任务优先级的计算和更新与环境的实时变化密切相关。

4.3 新的优先级分配方案

在嵌入式实时操作系统中，特别强调任务优先级的确定性与实时性。通常采用任务队列的方式进行管理。当有新任务进入队列时，一种方式是直接放在任务队列的末尾，在进行任务调度时，需遍历队列，来获得就绪队列中的最高优先级；另一种方式是把所有任务的优先级按顺序排序，当有新任务进入队列时，把新任务按次序放在就绪队列中的合适位置，在进行任务调度时，直接从队头调取任务。这两种方式所花费的时间都与任务数量有密切的关系，时间具有不确定性，不能达到实时性的要求。为了提高实时内核的确定性，通常采用这种称为优先级位图的就绪任务处理算法，通过对表示优先级位图的数据类型的改变，达到了扩展优先级数的目的。

就绪时间、运行时间、等待时间等时间因素往往与其任务的优先级高低起着决定性作用。实时任务的优先级随着时间因素的变化而变化，为实时任务动态调整其优先级可以提高系统的实时性。要求不但要考虑系统的响应速度，而且还要考虑实时任务能否在时限内完成。

实时任务调度算法尽可能的综合考虑任务价值和任务时限，以保证实时任务在时限内尽可能多地完成。

本论文设计一种新的优先级分配方案，在任务调度时通过衡量任务价值和任务时限来决定任务优先级的高低，设计原则是任务的时限越小且价值越大，则任务的优先级越高；对于时限与价值完全相同的任务，先到达者具有更高的优先级。

任务 Ti 的优先级 $Pnew_i$ 计算式为：

$$Pnew_i = \frac{V_i}{DTime_i} \qquad \text{公式（4.1）}$$

由公式(4.1)可知，实时任务的优先级高低与价值成正比，与时限成反比。其中 $DTime_i$ 表示时限，V_i 表示任务价值，$Pnew_i$ 表示优先级大小，但并不表示实际优先级的值，而是表示在优先级队列中排序的前后。

在此方案中，首先要知道当前任务的时限和优先级，然后累计任务的运行时间和任务的时限。且正在运行的任务，其时限不变，而就绪队列上的任务，其时限随着任务运行时间的推移而减小，从而实现动态调整实时任务的优先级高低。

优先级方案如图 4.3 所示，其中横轴 V_i 表示任务价值，纵轴 $DTime_i$ 表示时限，表中序号表示任务的优先级顺序，序号值越小任务优先级越高。同一对角线上的任务在原则上具有相同的优先级，但在实际调度中是不同的优先级，这些原则上同优先级任务的调度顺序按照表中序号的顺序。

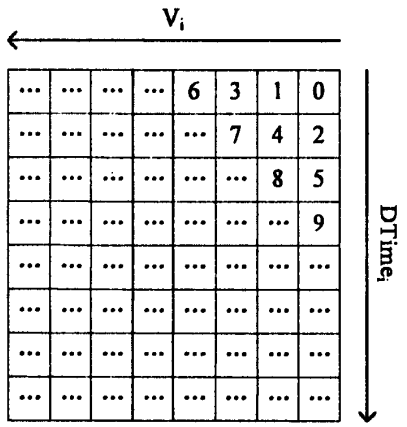

图 4.3 新的优先级分配方案

在这种优先级分配方案中，任务的时限序列按照升序排列，即任务的时限越小，其优先级越高；任务的价值序列按照降序排列，即任务的价值越大，其优先级越高。对于具有同样时限的任务，价值越大，任务的优先级越高；而对于具有同样价值的任务，时限越小，任务的优先级越高。新的优先级分配方案的任务就绪表如图 4.4 所示。

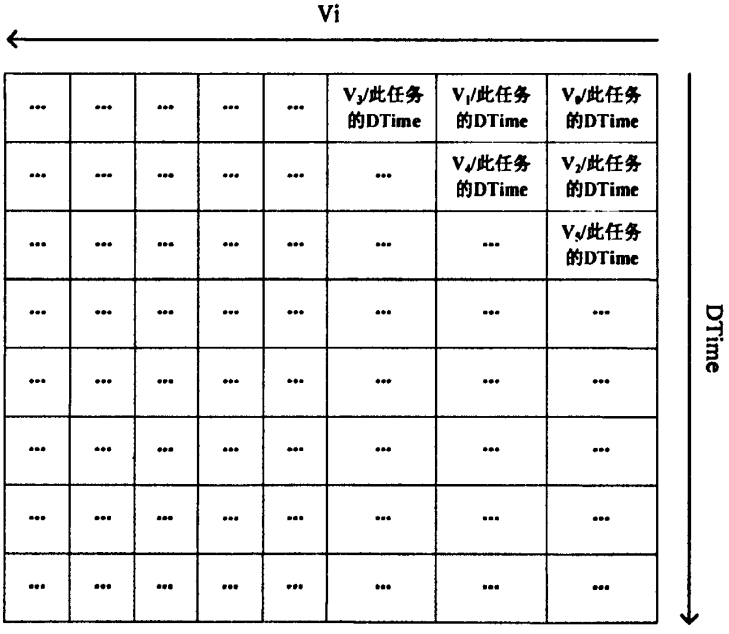

图 4.4 新的优先级分配方案的任务就绪表

在此调度方案中，任务调度时，任务调度器判断正在占用 CPU 的任务和就绪队列中的队首任务是否同一优先级，如果是同一优先级的，CPU 继续执行当前任务，如果不是同一优先级的，调度就绪队列中的队首任务，队首任务进入执行状态。

4.4 新的微内核任务调度算法设计

4.4.1 实时调度器

一个实时系统性能的好坏很大程度上取决于实时调度算法的选择。为了满足更广泛的各种实时和非实时任务的需求，以及更有效的协调各种不同任务以最大限度的提高系统性能，内核提供了丰富的线程调度机制。实时调度器是一个可以替换的核心组件，在最简单的固定优先级调度机制基础上，实现 FIFO 调度，时间片轮转调度以及混合时限调度等多种调度策略。

目前绝大多数的嵌入式实时操作系统的调度策略实现相对比较简单，简单的优先级调度结合相同优先级上的时间片轮转调度。通过在 EDF 调度算法的基础上加以改进，对于每一个实时任务，系统通过增加少量的开销，将运行在系统中的硬实时任务、软实时任务和非实时任务统一调度，调度策略既满足了所有硬实时任务的时限要求，最大限度的按时完成实时任务，最大程度上缩短了实时任务的响应时间和周转时间。

4.4.2 中断管理机制

随着嵌入式实时系统的发展，为了方便对中断的处理，系统内核常接管中断的处理，比如提供一些系统调用接口来安装用户的中断，提供统一的中断处理接口等。

微内核的中断处理程序可以分为两类，一类是低级中断处理，另一类是高级中断处理。低级中断处理即中断入口程序，中断发生后，经过核心的中断分配器直接进入的程序。由于进入中断入口程序后，处理器处于关中断状态，因此这段程序的执行时间应控制在 10us 内，只用来处理最紧急的 I/O 操作，而将不太紧急的操作放在高级中断处理程序中。高级中断处理即中断处理程序，处理中断入口程序没有处理完的 I/O 操作。中断入口程序执行最紧急的 I/O 操作后激活中断处理程序。所有的高级中断处理程序在系统中一个高优先级线程中执行，按照先入先出方式调用。高级中断处理程序不允许进入阻塞状态，可以等待互斥锁，其前提是该互斥锁是该设备驱动程序独立使用的，并且所有程序在拥有该互斥锁的期间不会进入阻塞状态。

中断服务线程是在必要时替代高级中断处理程序的核心线程。例如，当一个设备驱动程序的中断处理比较复杂，可以使用中断服务线程来代替高级中断处理程序，在驱动程序初始化时由驱动程序的初始化过程创建，其优先级可以根据需要选择一个适当的数值。中断服务线程处理完中断后，进入阻塞状态，等待下一个中断，并由中断入口程序在适当的时候将其唤醒。

微内核把中断服务程序的执行分为两个部分，调度中断服务程序使用优先级的调度策略，相当于在硬件和操作系统之间添加了两个软件层：软中断控制层和优先级中断调度层，其结构如图 4.5 所示。

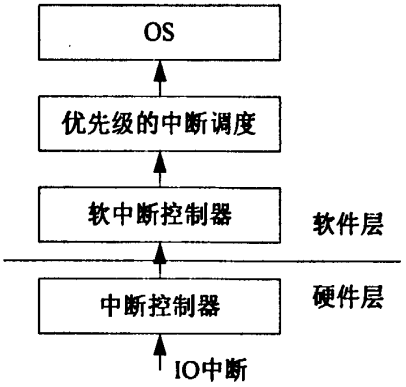

图 4.5 中断模型

在非抢占式微内核中，中断服务程序可以使一个高优先级的任务就绪，但中断完成后仍返回到被中断的任务，直到其主动放弃 CPU 的使用权，然后才能调度高优先级的就绪任务，这种系统的性能比前/后台方式的要好，但实时性还是不高，因为高优先级的任务就绪了也不能马上得到执行。非抢占式微内核任务调度中断模型如图 4.6 所示。

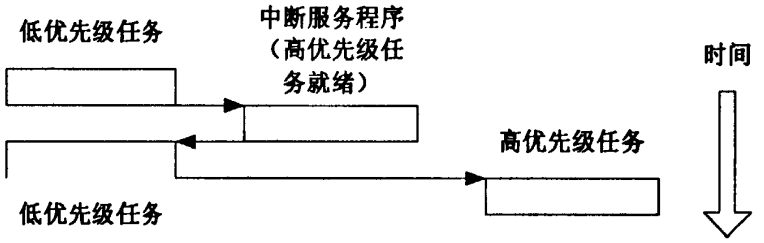

图 4.6 非抢占式微内核任务调度中断模型

在可抢占式微内核中，中断服务程序退出时会有两种情况，如果中断服务使高优先级任务就绪，则内核挂起被中断的低优先级任务，使高优先级的任务执行；如果中断服务没有使高优级任务就绪，则中断后仍返回被中断了的任务。可抢占式微内核任务调度中断模型如图 4.7 所示。可抢占式微内核中断处理模式下的时序如图 4.8 所示。这种处理模式有利于高优先级任务的处理，但相应地延长了被中断的低优先级任务的执行时间。

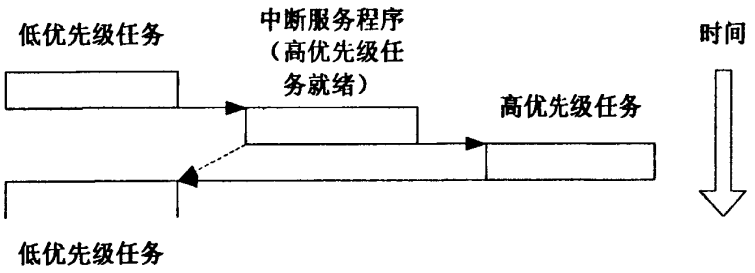

图 4.7 可抢占式微内核任务调度中断模型

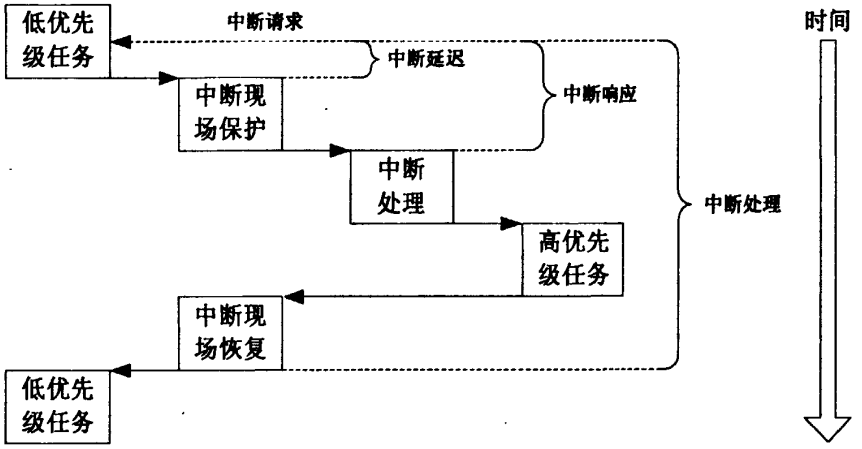

图 4.8 可抢占式微内核中断处理模式时序

4.4.3 时间片轮转与 EDF 调度算法

在嵌入式实时操作系统中对任务按其重要程度和响应要求的不同赋予不同的优先级，这样能够使紧急的任务优先得到响应，提高系统的实时性。然而，在实际的应用中，往往会有多个任务具有同等的重要性，它们应该被赋予相同的优先级，在系统调度时得到同等的对待，如果对两个重要程度相同的任务强行的赋予不同的优先级，很容易出现一个任务长时间的占用处理机，而另一个同等重要的任务却得不到处理的情况，造成进程饥饿，甚至饿死，这同样会影响系统的实时性，降低系统性能，甚至会引起灾难性的后果。因此要使系统的多个同等重要的任务都能或多或少得到 CPU 控制权，可以对这些任务采用时间片轮转调度减少进程饥饿发生，提高系统的实时性，增强系统性能。

根据 EDF 算法，处理器将分配给当前距离绝对时限最近的任务。EDF 算法是动态调度算法，任务的优先级不是固定的，而是根据各任务与时限的接近程度决定。动态优先级调度算法最大的优点是对于突发的实时的事件实时处理能力强，系统调度灵活，同时 CPU 的利用率较高，最大缺点在于系统的开销较大。

按照任务模型，假设任务集$S=(r_1,r_2,r_3)$，r_1，r_2，r_3是相互独立的实时任务，并对相应任务赋给时间片的值，任务的时限等于任务自身的周期，时间片的值均相等。相应的任务周期分别为T_1，T_2，T_3，任务的时限分别为$DTime_1$，$DTime_2$，$DTime_3$，时间片分别为S_1，S_2，S_3。实时任务r_1的$S_1=2$，$T_1=DTime_1=4$；实时任务r_2的$S_2=2$，$T_2=DTime_2=6$；实时任务r_3的$S_3=2$，$T_3=DTime_3=8$。

在时间片轮转调度下实时任务的运行结果如图4.9所示，在EDF调度下实时任务的运行结果如图4.10所示，在新的微内核调度下实时任务的运行结果如图4.11所示。

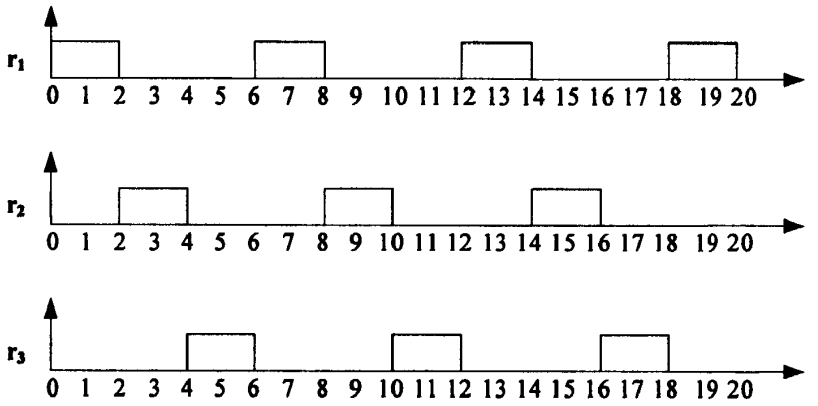

图4.9 在时间片轮转调度下任务调度情况

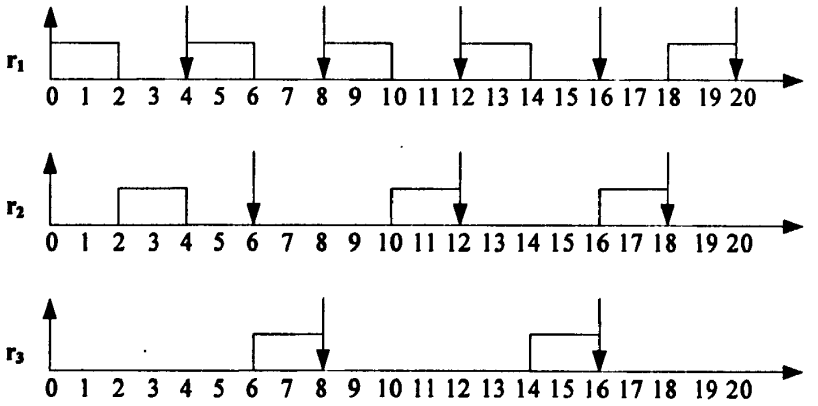

图4.10 在EDF调度下任务调度情况

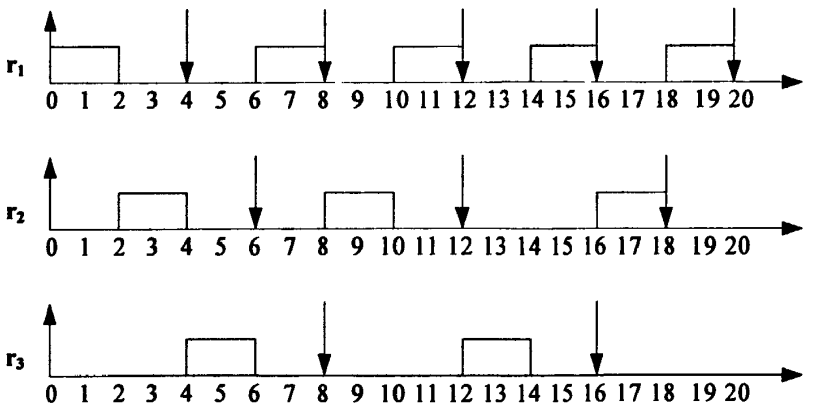

图 4.11 在新的微内核调度下任务调度情况

由上图可看出，周期最短同时时限也最短的实时任务 r_1，在时间片轮转调度下只完成了四个循环，而在 EDF 调度下和改进的微内核调度下都完成了五个循环，这是 EDF 调度的优势。周期最长同时时限也最长的实时任务 r_3，在 EDF 调度下，完成的两个循环任务都仅在时限前完成，在改进的微内核调度下，完成的两个循环任务不仅在时限前完成，还提前了两个时间片，这是时间片轮转调度的优势。

4.4.4 新的微内核任务调度算法的流程

基于以上讨论和研究，新的微内核调度算法首先通过衡量任务价值和任务时限来决定任务优先级的高低，由此对就绪队列排序，然后对就绪队列队首任务进行时间片轮转调度。此调度算法的设计原则是任务的时限越小且价值越大，则任务的优先级越高；对于时限与价值完全相同的任务，先到达者具有更高的优先级。

新的微内核任务调度算法步骤如下：

① 初始化 TCB 和各个任务的 T_n、C_n、$DTime_n$、V_n；

② 检查任务队列中所有就绪任务，根据式（4.1）比较 PRIOnew 的值（任务的时限等于任务自身的周期），并以此来确定任务的优先级，将任务的优先级由高到低排列，任务优先级的值越小，任务的优先级越高，在队列位置越前面；

③ 更新计数器；

④ 调度队列中的第一个任务，即优先级最高的任务，并获得当前任务的时间片；

⑤ 判断该任务时间片是否为 0，若不是，则转到步骤⑥，若是，则转到步骤⑦；

⑥ 执行当前任务，并将当前任务的时间片置为 0；

⑦ 判断任务是否完成，若不是，将当前任务放入就绪队列的队尾，若是，从任务队列中将任务删除；

⑧ 队列中已经没有任务则调度完成；

新的微内核任务调度算法的流程图如图 4.12 所示。

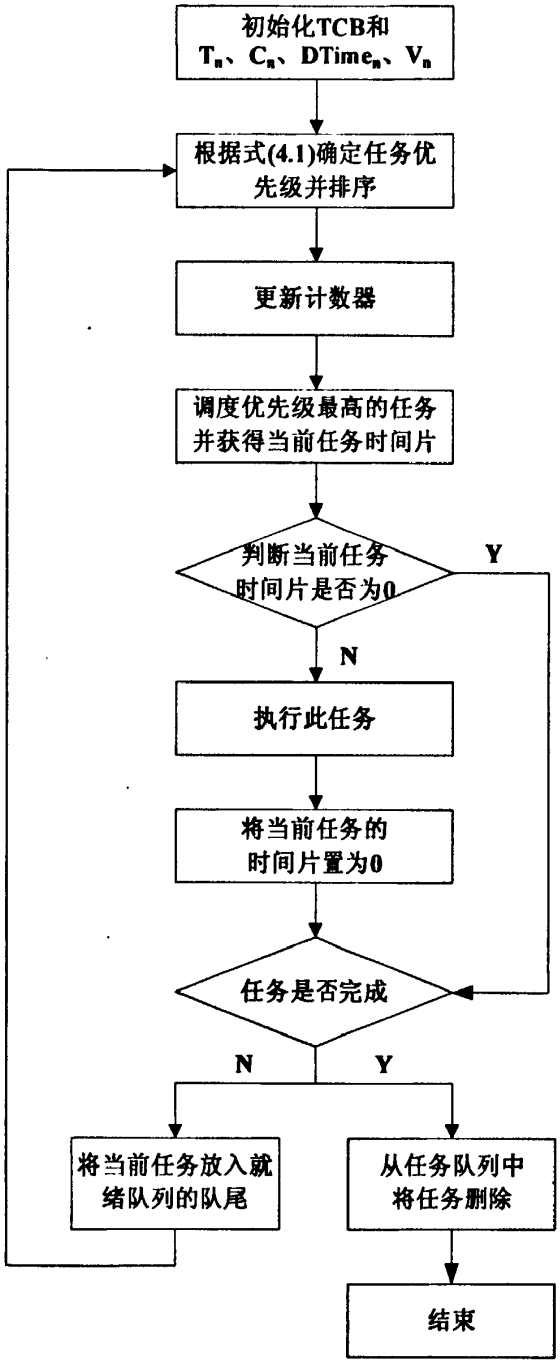

图 4.12 新的微内核任务调度算法流程图

此调度算法是时间片轮转调度算法与基于时限与价值的优先级调度算法的结合。若跳过步骤②和步骤④，则算法退化为时间片轮转调度；若跳过步骤③、步骤⑤和步骤⑥，则算法退化为基于时限与价值的优先级调度算法。

4.5 本章小结

本章首先提出问题，并对任务集建模，详细叙述了新的优先级的分配方案，此方案通过衡量任务价值和任务时限来决定任务优先级的高低。然后讨论了实时调度器和中断管理机制，叙述了新的微内核任务调度算法，算法的设计原则是任务的时限越小且价值越大，则任务的优先级越高；对于时限与价值完全相同的任务，先到达者具有更高的优先级，最后给出了新的微内核任务调度算法的流程图。

第 5 章 新的微内核任务调度算法实现与测试

5.1 微内核 μC/OS-II 简介

μC/OS-II 是由美国 Micrium 公司出品的一个实时操作系统，它是一个源码开放、可移植、可固化、可裁剪、具有可剥夺实时内核的实时多任务的操作系统。适用于任务多，对实时要求较严格的场合。μC/OS-II 已经在世界范围内许多领域得到广泛的应用，如手机、路由器、集线器、飞行器、医疗设备及工业控制等。现有许多开发者把它成功地应用于各种系统，并已经确认了其安全性和稳定性。

μC/OS-II 的内核具有许多功能，如任务调度与管理、时间管理、任务间同步与通信、内存管理和中断服务等。任务管理、任务之间的通信与同步、时间管理、中断管理和内存管理是 μC/OS-II 的微内核结构的五个模块。用户通过微内核函数的形式调用这些模块功能。需要说明的是，用户多任务是建立在 μC/OS-II 的各种系统服务之上，系统并没提供与硬件的接口程序，需要用户直接与硬件打交道。μC/OS-II 微内核体系结构如图 5.1 所示。

<table>
<tr><td colspan="5">用户多任务</td></tr>
<tr><td rowspan="4">I/O管理和
驱动程序</td><td>任务管理</td><td>任务通信与同步</td><td>时间管理</td><td>内存管理</td></tr>
<tr><td colspan="4">μC/OS-II与硬件无关代码</td></tr>
<tr><td colspan="4">μC/OS-II与硬件相关代码</td></tr>
<tr><td colspan="4">硬件层</td></tr>
</table>

图 5.1 μC/OS-II 微内核体系结构

μC/OS-II 包括用于应用任务的例子和辅助代码，μC/OS-II 核心代码约 3500 行，包括与处理器无关的代码 OS_CORE.C、OS_FLAG.C 等，与应用相关的代码 OS_CFG.H、INCLUDES.H，与处理器相关的代码 OS_CPU.H、OS_CPU_C.C 等，分布在图 5.2 所示文件中。

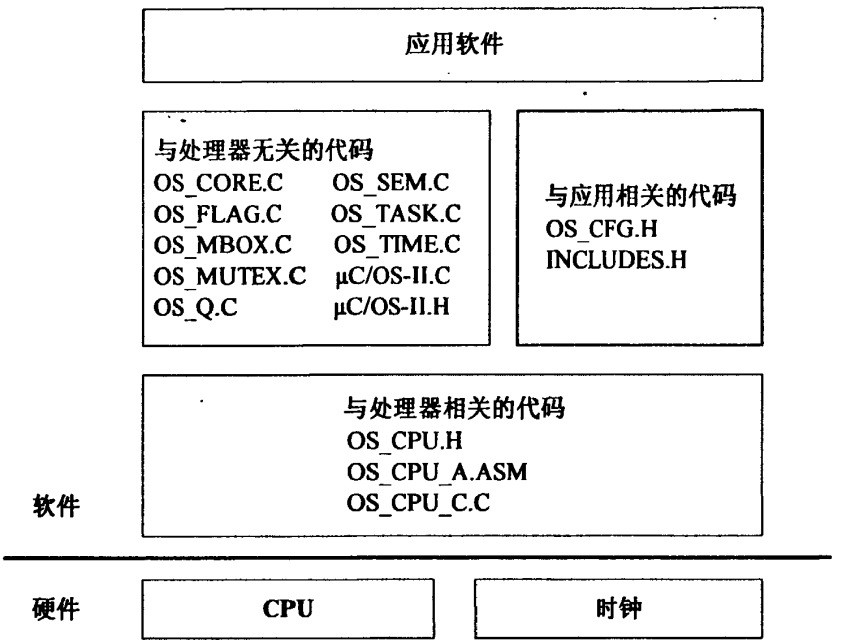

图 5.2 µC/OS-II 文件组织结构

µC/OS-II 具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点，具体特点如下：

① 源代码公开：源代码全部公开，通俗易懂，注释详细，机构清晰。与其他封闭式的实时内核不同，µC/OS-II 具有透明性和可用性，就使得它在各种嵌入式平台上容易运行。

② 强移植性：µC/OS-II 的源码绝大部分是用 C 语言写的，带有少量的汇编程序，并且有详细的说明和示例，所以 µC/OS-II 能运行在绝大多数微处理器或者数字信号处理器上，并且可以在有堆栈指针的处理器通过 CPU 内部寄存器入栈出栈来移植。µC/OS-II 的使用对象主要是嵌入式系统，并且很容易移植到不同构架的微处理器上。

③ 裁剪性：在具体应用中，用户需要什么样的服务，µC/OS-II 就可以设计相应的条件进行编译，通过编译来实现相应的程序。

④ 可抢占性：可抢占性是 µC/OS-II 等嵌入式操作系统的微内核的主要特点之一，所谓可抢占性是指具有高优先级的任务，在处理紧急事件时，需要抢占低优先级的任务的 CPU 控制权，直到处理事件完毕，才释放 CPU 控制权和所占用的资源。

⑤ 多任务性：为了同时实现多个不同的事件同时被处理，多个不同优先级的任务同时执行，多任务是 µC/OS-II 微内核的一个重要特征。

⑥ 时间可确定：µC/OS-II 的执行时间是确定的，不管应用程序任务数量是多少，所有 µC/OS-II 的函数调用和系统服务的执行时间是可知的、不变的。

⑦ 任务栈：系统服务 μC/OS-II 根据功能的不同把任务放在不同的栈中，每个任务在系统服务 μC/OS-II 占有不同的栈空间。

⑧ 中断机制：系统服务 μC/OS-II 采用了中断嵌套层，挂起正在执行的低优先权的任务，执行高优先权的任务，以达到实时性。

5.2 μC/OS-II 任务调度算法分析

任务调度算法是 μC/OS-II 中最重要的算法之一。μC/OS-II 任务调度算法通过建立 OSUnMapTbl[]和 OSMapTbl[]两张表，使任务切换执行时间恒定，不会随着任务数目变化而变化，从而保证了系统的确定性和实时性。μC/OS-II 中存在众多的全局变量，如 OSRdyTbl[]、OSEventTabl[]和 OSTCBTbl[]等，这种设计思想避免了动态初始化。对于一个任务调度十分频繁的操作系统，这一点空间相对其换取的 CPU 时间是微不足道的。μC/OS-II 正是采用这种策略使其性能得到了大大的提升，这种处理方法也是 μC/OS-II 任务管理效率高的关键因素。

对于 μC/OS-II 定义的每一个任务，在创建任务时就会为其分配一个合适的优先级，但是任务的优先级是可变的，即支持动态优先级。由于 μC/OS-II 是基于优先级的抢占式调度，因此每个任务的需要拥有不同的优先级。当任务就绪之后，就能够获得 CPU 的控制权。如果在中断程序的执行过程中，得到一个具有较高优先级的任务，那么这个任务将在中断程序执行完成之后执行。

任务调度算法主要包含以下模块：任务控制块、任务就绪表和任务调度器。下面对其分别进行介绍。

5.2.1 μC/OS-II 的任务控制块

μC/OS-II 中的每个任务都有一个任务控制块 OS_TCB（OS_Task Control Block）。任务控制块记录任务的优先级、任务的堆栈指针、相关事件控制块指针等任务的执行环境。在任务建立之后，任务对应的任务控制块 OS_TCB 就会被赋值。任务控制块用于保存和恢复该任务的状态，处于运行中的任务被剥夺 CPU 的使用权时，任务控制块立即保存当前任务的状态，在当前任务获得 CPU 使用权之后恢复当前任务的执行状态。

μC/OS-II 的 TCB 数据结构简单，内容容易理解，保存最基本的任务信息，同时还支持裁减来减小内存消耗，TCB 是事先根据用户配置，静态分配内存的结构数组，通过优先级序号进行添加，查找，删除等功能。减少动态内存分配和释放。因为依靠优先级进行 TCB 分配，每个任务必须有自己的优先级，不能和其他任务具有相同的优先级。

5.2.2 μC/OS-II 的任务就绪表

μC/OS-II 的是通过任务就绪表来进行任务的调度和任务优先级的处理。任务就绪表是一个包含所有任务的运行状态以及任务优先级的相应的信息的数组。μC/OS-II 的任务就绪表如图 5.3 所示。

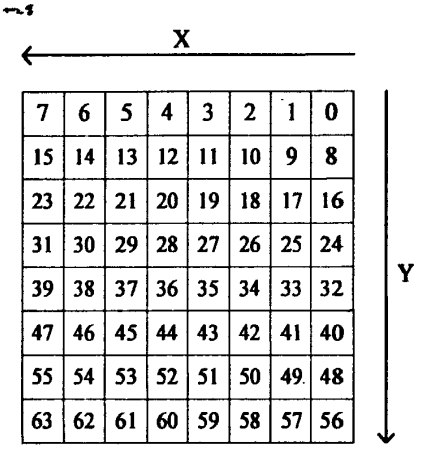

图 5.3 μC/OS-II 的任务就绪表

就绪表中包含有每一个任务的就绪态标志，就绪表中有两个变量 OSRdyTbl[]和 OSRdyGrp。

OSRdyTbl[]表示各个任务的就绪与否的状态，μC/OS-II 最大允许 64 个任务，这样任务就绪表就用 64 位指示各个任务的运行状态，调度只关心各个任务的就绪与否的状态，只有拥有较高优先级并且存在于就绪态中的任务才有机会被调度，用“1”表示该任务处于就绪态，用“0”表示该任务不处于就绪状态。每个任务的优先级的信息都是通过任务在数组中的位置的信息来表示的，具有较高优先级的任务在数组中所处的位置也相对较低。在任务进入就绪态之后，OSRdyTbl[]就绪表中的相应元素的相应位置也会产生变化。

OSRdyGrp 组变量用来表示就绪位在 OSRdyTbl[]中的位置。OSRdyGrp 用于方便查找计算，大小为一个字节。在 OSRdyGrp 中，按照任务优先级的不同分成不同的组，8 个任务为一组，OSRdyGrp 所包含的每一位是用于标志是否有任务进入队列，以便加快检索速度。

OSRdyTbl[]和 OSRdyGrp 之间的关系图如图 5.4 所示，且满足如下规则：当 OSRdyTbl[i]中任何一位被置 1 时 OSRdyGrp 的第 i 位置也相应的被置 1。

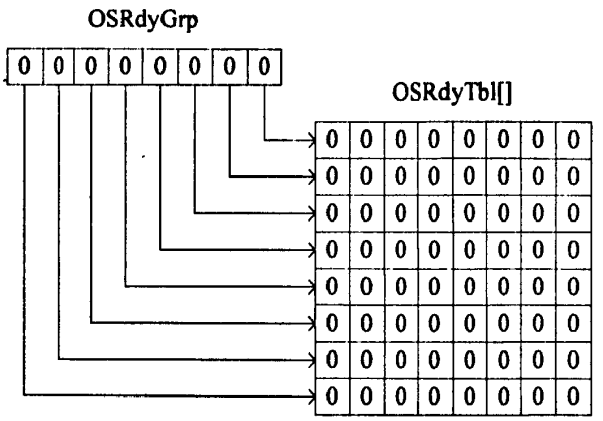

图 5.4 OSRdyTbl[]和 OSRdyGrp 间的关系图

不同的任务拥有从 0 级到最低优先级 OS_LOWEST_PRIO 的不同的优先级。在 μC/OS-II 系统进行初始化的时候，空闲任务总是获得最低优先级 OS_LOWEST_PRIO。

通过查表，可以确定 μC/OS-II 系统中的所有系统调用的时间。这样，就使得系统的时间确定下来，增加了系统的可预测性。

5.2.3 μC/OS-II 的任务调度器

μC/OS-II 总是运行进入就绪队列任务中优先级最高的任务，这项工作由调度器完成。在系统中，由函数 OSSched( )完成任务的调度；由函数 OSIntExt( )完成中断级的调度。

当进入就绪表找到优先级最高的任务之后，函数 OSSched( )可以检验当前正在运行的任务是否是这个优先级最高的任务，从而避免不必要的任务调度。嵌入式系统设备对实时性要求很高，这样可以提高任务调度的实时性。

使优先级为 prio 的任务进入就绪态的代码如下：

```
OSRdyGrp |=OSMapTbl[prio>>3]
OSRdyTbl[prio>>3] |=OSMapTbl[prio&0×07]
```

使优先级为 prio 的任务脱离就绪态的代码如下：

```
If((OSRdyTbl[prio>>3]&=～OSMapTbl[prio&0×07]==0)
OSRdyGrp&=～OSMapTbl[prio>>3]
```

可以通过以上两个步骤完成任务的切换工作，首先把即将挂起任务的微处理器和寄存器信息放入到堆栈中，然后把获得 CPU 控制权的具有较高优先级的任务的寄存器信息从堆栈中恢复。在 μC/OS-II 中，处于就绪队列的任务要通过两个过程完成任务的切换。一是恢复 CPU 寄存器中所包含的信息，二是运行中断返回指令。通过 OS_TASK_SW( ) 进行任务切换。在大多数的微处理器通过软中断指令或者陷阱指令 TRAP 来实现以上操

作。中断服务子程序或陷阱处理通过把中断向量传递给汇编语言函数 OSCtxSw( )来实现中断操作。OSCtxSw( )要通过 OS_TCBHighRdy 变量和当前任务控制块 OSTCBCur 变量指向即将被挂起的任务。

OSSched( )包含的代码都是临界段代码，需要运用汇编语言编写用于缩短切换时间。可以通过运用少量的 C 语言编写 OSSched( )函数来增加代码的可读性和移植性。

5.3 μC/OS-II 任务调度算法的改进

5.3.1 μC/OS-II 任务调度算法的局限性

μC/OS-II 的内核调度机制具有高效率和简单快捷的特性，但是任务调度算法中也有它得局限性，比较突出的局限性体现在 μC/OS-II 的优先级和任务具有一一对应的特性。μC/OS-II 的任务调度算法的不足和局限性体现在如下几点。

1、任务数受优先级数的限制

由于一个优先级只能对应一个任务，优先级数目应当能对应于实际系统中需求的任务数。μC/OS-II 中，任务优先级数为 64 个，其中从 0 到 3 和 60 到 63 这八个优先级数被系统所保留，剩下的 56 个优先级提供给用户使用。如此少的优先级数目只适用于及其简单的嵌入式系统，但如今嵌入式得蓬勃发展，投入市场应用的方面会越来越多，同时对应的相关任务数目也会日新月异得成长，因此仅仅 56 个优先级任务式是无法达到市场应用的需求。

2、缺乏时间片轮转调度

μC/OS-II 不支持时间片轮转调度，低优先级的任务没有得到 CPU 控制权不会运行，而高优先级的任务会一直运行，那么在系统中并行运行的任务只能通过信号灯和休眠的方式启动运行。

5.3.2 μC/OS-II 调度算法改进的实现

1、微内核时间片轮转调度算法的实现

系统时钟函数 OSTimeTick( )可以用来计算时间片轮转调度中时间片的消耗。OSTimeTick( )可以减少任务的时间片，若时间超过一定的值就可以转换它的状态，而这个函数可以很好的实现这一点。当 OSTSCurLen=0，表示时间片用完了，下一个任务就可以从就绪状队列进入运行状态。OSTCBReadyTbl[]中的指针向下移动一位任务，然后把另一个同优先级的任务启动，让其运行。微内核时间片轮转调度算法的流程如图 5.5 所示。由图可看出，图中的每个任务都会循环的执行，那么时间片轮转调度也能很好的在操作系统微内核中运行。

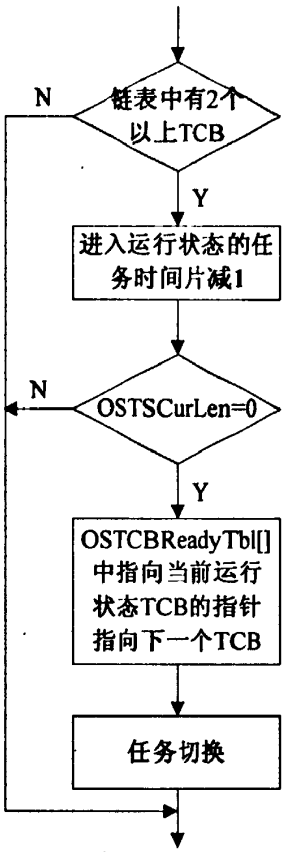

图 5.5 时间片轮转调度算法流程图

2、优先级继承的实现

优先级继承能够很好的处理互斥信号量中产生的优先级反转问题，对于高 16 位的互斥信号量，可以把它设为当前的优先级，而低于 16 位任务可以保存它之前的优先级。

当 OSMutexPend( )函数被调用执行时，当互斥信号量在使用时，先比较拥有该互斥信号量的任务和当前的这个任务的优先级。如果当前任务优先级高，那么它就能最早的继承互斥信号量的资源。

当调用拥有该互斥信号 OSMutexPost( )这个函数，可以比较低 16 位的任务和当前这个任务的优先级，如果它们不相等，说明它们曾出现了优先级继承，那么就可以把拥有互斥信号量的任务的优先级调整为之前的优先级，然后再执行下面的任务。

在以上过程需要对 OSMutexPend( )和 OSMutexPost( )这两个函数进行修改，优先级继承流程如图 5.6 所示。

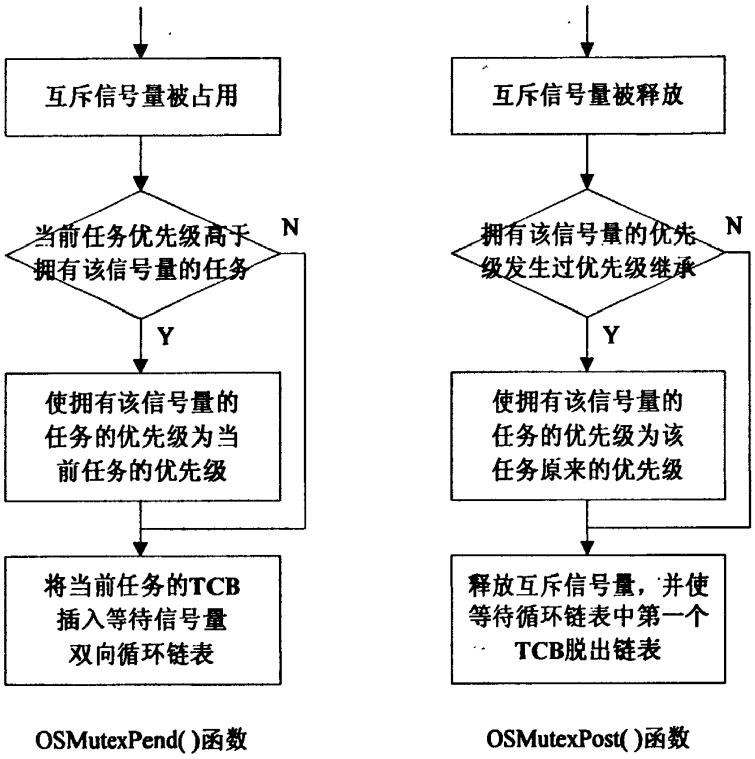

图 5.6 优先级继承流程图

3、增加任务的数量

在 μC/OS-II 中，分别用 3 个比特位来定位优先级任务就绪表中的行和列，即 0-2 位标识任务在任务就绪表中的列信息，3-5 位标识任务在任务就绪表中的行信息。因此存放任务优先级的 8 位比特只用到了 6 位，还剩下 2 个空闲的位。可以通过直接扩展定位信息所占的比特位，使其能够区分 256 个不同任务的优先级。扩展后的算法规定任务优先级字节的定义如图 5.7 所示。

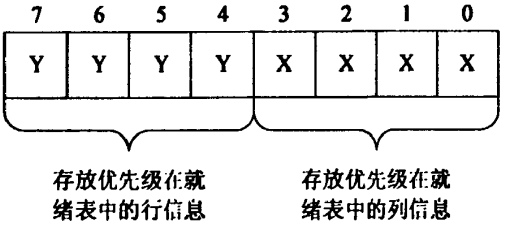

图 5.7 优先级定位字节

该算法继续使用 μC/OS-II 中所定义 OSRdyTbl[]和 OSRdyGrp 就绪表变量。就绪表中任务优先级所在的位置，是所在行通过变量 OSRdyGrp 表示，在 OSRdyGrp 中，将按照任务优先级的不同，将任务分为若干组，每 16 个任务分为一组。就绪态通过表·

OSRdyGrp 中的每一位来表示 16 组任务中是否有处于就绪态的任务，在 OSRdyGrp 中所对应的位置置为 1。优先级在就绪表中的列信息是通过 OSRdyTbl[]中的数组来表示，OSRdyTbl[]同样依据扩展规则将该数组的由 8 位扩展为 16 位，如果一个优先级任务处于就绪态，则其所对应的位的值置为 1。例如，OSRdyTbl[0]对应优先级为 0-15 的任务，OSRdyTbl[1]对应优先级为 16-31 的任务，依次类推，OSRdyTbl[15]对应优先级为 240-255 的任务。如果处于就绪状态的任务的优先级为 78，不仅要将 OSRdyTbl[4]的第 14 位置 1，而且要将 OSRdyGrp 的第 4 位置 1。只要 OSRdyTbl[n]中有一位为 1，则 OSRdyGrp 的第 n 位就为 1。变量 OSRdyTbl[]和 OSRdyGrp 之间的关系如图 5.8 所示，图中 OSRdyTbl[]下表格中的数字 0-255 仅表示 256 个任务，并非实际存放的状态信息。

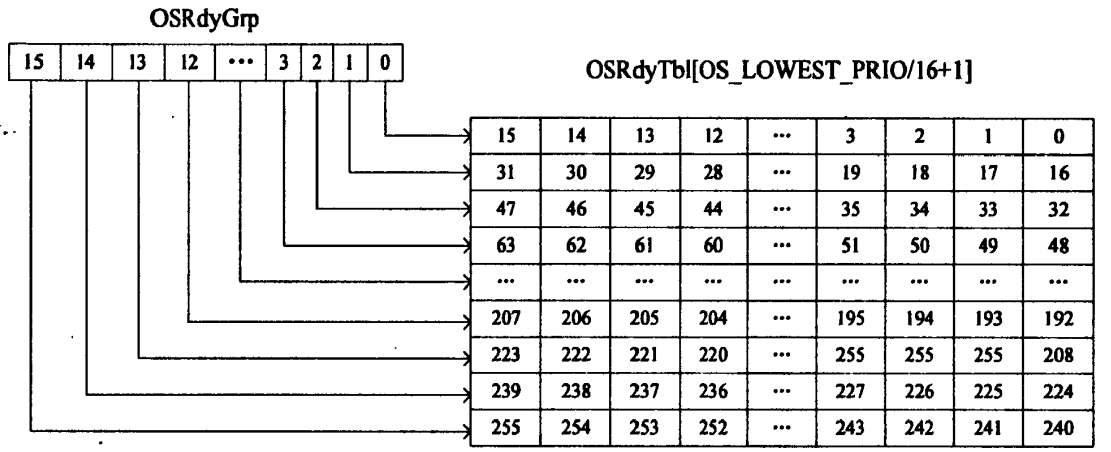

图 5.8 增加任务的就绪表

任务数增加的任务就绪表的代码如下：

```
OSRdyGrp |=OSMapTbl[prio>>4];
OSRdyTbl[prio>>4] |=OSMapTbl[prio&0X0F];
```

其中，char OSMapTbl[]={0X01, 0X02, 0X04, 0X08, 0X10, 0X20, 0X40, 0X80,0X0100, 0X0200, 0X0400, 0X0800, 0X1000, 0X2000, 0X4000, 0X8000}，用于限制 OSMapTbl[]数组的元素下标在 0 到 15 之间，prio 表示任务的优先级。

5.3.4 新的微内核和 μC/OS-II 微内核的调度算法比较

目前的 μC/OS-II 具有强实时性，结构小巧，高裁剪性，易移植的特点。通过上文的一系列扩展设计，使 μC/OS-II 除了具有以上优点外，还具有时间片轮转的相互协作调度和优先级抢占的优点。与此同时，对于上层用户应用程序新内核仍然保证了优良的兼容性，系统 API 在函数名称，功能上几乎和以前的完全一致。新的微内核任务调度算法在

实时性，模块设计，任务划分，资源分配等方面和原内核相比具有一定的优越性。表 5.1 是对新的微内核和 μC/OS-II 微内核的部分参数比较。

表 5.1 新的微内核和 μC/OS-II 微内核的部分参数比较

参数	新的微内核	μC/OS-II 微内核
支持优先级数	1024	64
支持任务数	2^{16}	64
支持同优先级任务	支持	不支持
支持优先级继承	支持	不支持
支持时间片调度	支持	不支持

说明通过实现时间片轮转算法在内核部分的扩展，优先级继承，扩充任务数，改进 μC/OS-II 的调度算法，使得新构建的微内核更可靠及具有更强的适用性。

5.4 新的微内核任务调度算法实现

5.4.1 新的微内核数据结构变量的扩展

根据本算法的任务模型，需要在任务控制块 OS_TCB 中增加变量，微内核调度算法的扩展的数据结构代码如下：

```
INT8U   tperiod;
INT8U   tcpu;
INT8U   tarrive;
INT8U   dtime;
INT8U   value;
INT8U   sli;
INT8U   priority;
INT8U   flag;
```

参数说明：tperiod 表示任务运行的周期；tcpu 表示任务的执行时间；tarrive 表示任务到达的时间；dtime 表示任务的时限；value 表示任务的价值；sli 表示任务的时间片；priority 表示任务在就绪队列中的优先级，值越大则优先级越高；flag 标志着该任务是否运行完成，true 表示运行完成，false 表示没有运行完成。

5.4.2 新的微内核任务调度器实现

任务调度器 OSSched( )函数的任务是查找就绪队列队首的任务，即优先级最高的任务，并记录这个任务的优先级，然后判定当前任务和记录的任务是否为同一任务，最后调用上下文切换的函数实现。

OSSched( )函数的工作过程如下：首先，要确定任务调度器没有被上锁 OSLockNesting==0 并且任务的没有进入服务函数 OSIntNesting==0，在满足的情况下，找出优先级最高的就绪任务，如果这个任务不是当前任务的话，那么将其任务控制块指针指向当前最高优先级就绪任务的指针 OSTCBHighRdy，然后调用任务级任务切换函数 OS_TASK_SW( )进行任务切换。退出中断服务程序 OSIntExit( )的时候，调用中断级的任务切换函数 OSIntCtxSw( )来进行任务的切换。

OSSched( )函数的结构如下：

```
void OSSched(void){
关中断;
如果（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;
  如果（最高级的任务不是当前的任务）{
    调用 OSCtxSw( );
    }
  }
开中断;
}
```

OSSched( )函数的代码改动如下：

```
Void OSSched (void)
{
  #if OS_CRITICAL_METHOD == 3
  OS_CPU_SR  cpu_sr;
  #endif
  INT8U  y;
  OS_ENTER_CRITICAL( );
  if ((OSIntNesting == 0) && (OSLockNesting == 0)) {
    y  = OSUnMapTbl[OSRdyGrp];
    OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
    if (Deadline<tcpu [i][1]) {
```

```
        int j=i;
        while(tcpu [j][1]&&(j>0)) j--;
        csRec[j-1][1]++;
        }
        tcpu [i][1]=Deadline;
        if (OSPrioHighRdy != OSPrioCur) {
          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
          OSCtxSwCtr++;
          OS_TASK_SW( );
        }
      }
      OS_EXIT_CRITICAL( );
    }
```

5.5 性能测试实验及结果

5.5.1 实验平台 Proteus 介绍

仿真实验使用 Proteus 仿真软件，Proteus 是英国 Labcenter electronics 公司出版的 EDA 工具软件，是世界上著名的仿真软件，从原理图布图、代码调试到单片机与外围电路协同仿真，一键切换到 PCB 设计，真正实现了从概念到产品的完整设计。是目前世界上唯一将电路仿真软件、PCB 设计软件和虚拟模型仿真软件三合一的设计平台，其处理器模型支持 8051、HC11、PIC10/12/16/18/24/30/DsPIC33、AVR、ARM、8086 和 MSP430 等，2010 年即将增加 Cortex 和 DSP 系列处理器，并持续增加其他系列处理器模型。在编译方面，它也支持 IAR、Keil 和 MPLAB 等多种编译器。

其功能最大的特点是仿真处理器及其外围电路，可以仿真 51 系列、AVR、PIC、ARM、等常用主流单片机。还可以直接在基于原理图的虚拟原型上编程，再配合显示及输出，能看到运行后输入输出的效果。配合系统配置的虚拟逻辑分析仪、示波器等，Proteus 建立了完备的电子设计开发环境。

为了更好地测试新的微内核任务调度算法性能，采用在 Proteus 仿真平台中进行实验测试。因为 Proteus 仿真平台可以方便地搭建起不同的硬件的平台，有利于对微内核进行可移植性测试。此外对于运行效率的测试时也可以使用此平台进行，由于性能测试的方案是对比测试，那么只需要在同一环境下对本文设计的算法与对比对象算法进行比较测试即可达到测试的目的。

5.5.2 微内核的移植

所谓移植，指的是一个操作系统可以在某个微处理器或者微控制器上运行。下面将列出新的微内核移植所的一些必须条件，处理器必须满足以下要求：

① 处理器的 C 编译器必须能产生可重入代码。

② 在程序中可以打开或者关闭中断。

③ 处理器支持中断，并且能产生定时中断（通常在 10Hz-1000Hz 之间）。

④ 处理器支持能够容纳一定量数据的硬件堆栈。

⑤ 处理器有将堆栈指针和其他 CPU 寄存器存储和读出到堆栈（或者内存）的指令。

5.5.2.1 与应用相关的代码

这一部分是用户根据自己的应用系统来定制合适的内核服务功能，包括 2 个文件：OS_CFG.H 和 INCLUDES.H。OS_CFG.H 用来配置内核，用户根据需要对内核进行修改，留下需要的部分，去掉不需要的部分，比如系统可提供的最大任务数量，是否定制邮箱服务，是否提供优先级动态改变功能等等，所有的配置更改包括头文件的增减均在该文件中进行。

INCLUDES.H 系统头文件，整个实时系统程序所需要的文件，包括了内核和用户的头文件，这样使得用户项目中的每个.C 文件不用分别去考虑他实际上需要哪些头文件。

5.5.2.2 与处理器相关的代码

这是移植中最关键的部分。内核将应用系统和底层硬件有机地结合成一个实时系统，要使同一个内核能适用于不同的硬件体系，就需要在内核和硬件之间有一个中间层，这就是与处理器相关的代码，处理器不同，这部分代码也不同，在移植时需要自己处理这部分代码，这一部分代码分成 3 个文件：OS_CPU.H，OS_CPU_A.ASM，OS_CPU_C.C。

1. OS_CPU.H

此文件包含了用#define 定义的与处理器相关的常量、宏和类型、具体有系统数据类型、栈增长方向、关中断和开中断等常量与参数的定义。

① 不依赖于编译器的数据类型

根据 Keil-C 编译器的特性，相关的数据类型定义代码为：

```
typedef  unsigned  char  BOOLEAN;
typedef  unsigned  char  INT8U;
typedef  signed    char  INT8S;
typedef  unsigned  int   INT16U;
typedef  signed    int   INT16S;
typedef  unsigned  long  INT32U;
```

```
typedef  signed     long   INT32S;
typedef  float              FP32;
typedef  double             FP64;
```

② 定义栈的增长方向

```
#define OS_STK_GROWTH 0   /*从下往上*/
#define OS_STK_GROWTH 1   /*从上往下*/
```

③ 定义允许和禁止中断宏

```
#define   OS_ENTER_CRITICAL( )  EA = 0
#define   OS_EXIT_CRITICAL( )   EA = 1
```

④ 定义 OS_TASK_SW( )宏

从低到高优先级时调用软中断直接来将中断向量指向 OSCtxSw，直接调用 OSCtxSw。

```
#define   OS_TASK_SW( )  OSCtxSw( )
```

2. OS_CPU_C.C

OS_CPU_C.C 包含了与移植有关的 C 函数，包括堆栈的初始化和一些钩子函数的实现，其中最重要的是 OSTaskStkInit( )函数，该函数是在用户建立任务时系统内部自己调用的，用来对用户任务的堆栈初始化。相应的移植相关函数如下：

```
OSTaskCreateHook( );
OSTaskDelHook( );
OSTaskSwHook( );
OSTaskStatHook( );
OSTimeTickHook( );
```

3. OS_CPU_A.ASM

移植的绝大部分工作都是在修改 OS_CPU_A.ASM 文件中进行，因为这个文件中的 OSIntCtxSw 和 OSTickISR 这两个函数的实现是移植的方法修改和与相关硬件定时器、中断寄存器的设置有关，这个文件大部是使用了汇编语言编写，包括 4 个子函数：OSStartHighRdy( )、OSCtxSw( )、OSIntCtxSw( )、OSTickISR( )，只需要进行相应的移植修改即可。

5.5.3 性能测试结果与分析

对新的微内核测试的硬件平台是 AT89C52 的虚拟平台。如图 5.9 是测试设计的电路原理图，U1 是 AT89C52 单片机，U3 是 6116 内存芯片，因为 AT89C52 的内存空间太

小，所以需要进行内存的扩展。U2 是 74LS373 地址锁存器。为了能方便看到系统运行的情况，接上了一个虚拟的显示终端。

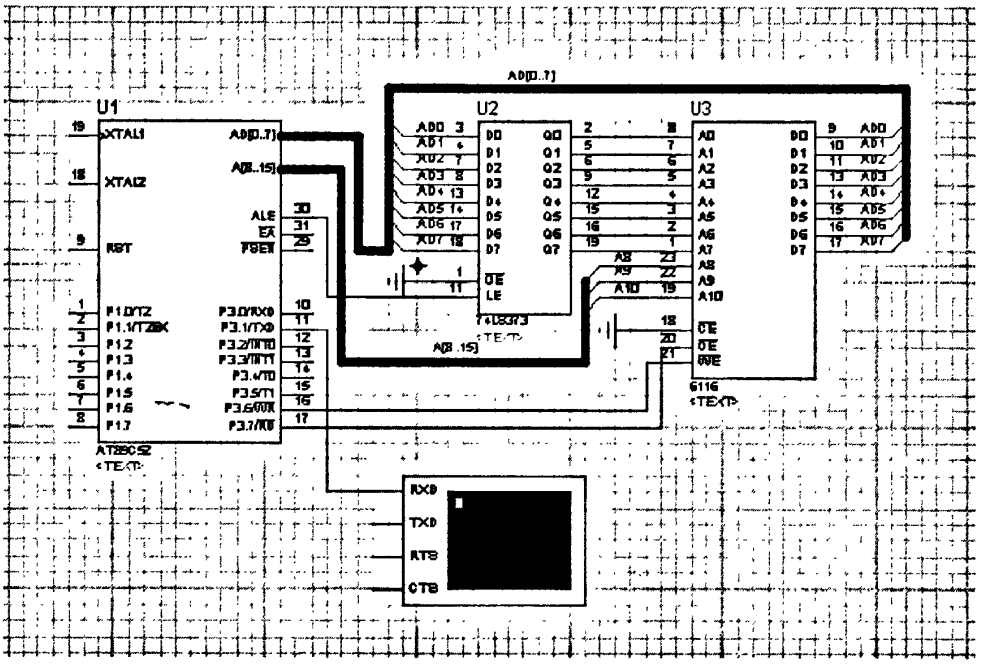

图 5.9 AT89C52 Proteus 原理设计图

为保证任务集样本的随机性，所有的任务都是随机产生的。假设两个任务集，分别是任务集 P_1 和任务集 P_2。周期和运行时间的单位为某个假定时间基的滴答（tick）数。任务的时限等于任务周期。任务执行时间在[10，30]内均匀，随机的选择。实验任务集参数如表 5.2 所示。

表 5.2 实验任务集

任务集	周期 T	时限 DTime	运行时间 C
任务集 P_1	200	200	随机
任务集 P_2	400	400	随机

该任务集根据微内核 μC/OS-II 任务调度算法和新的微内核任务调度算法分别进行调度，实验结果如图 5.10 所示，其调度结果数据如表 5.3 和表 5.4 所示。

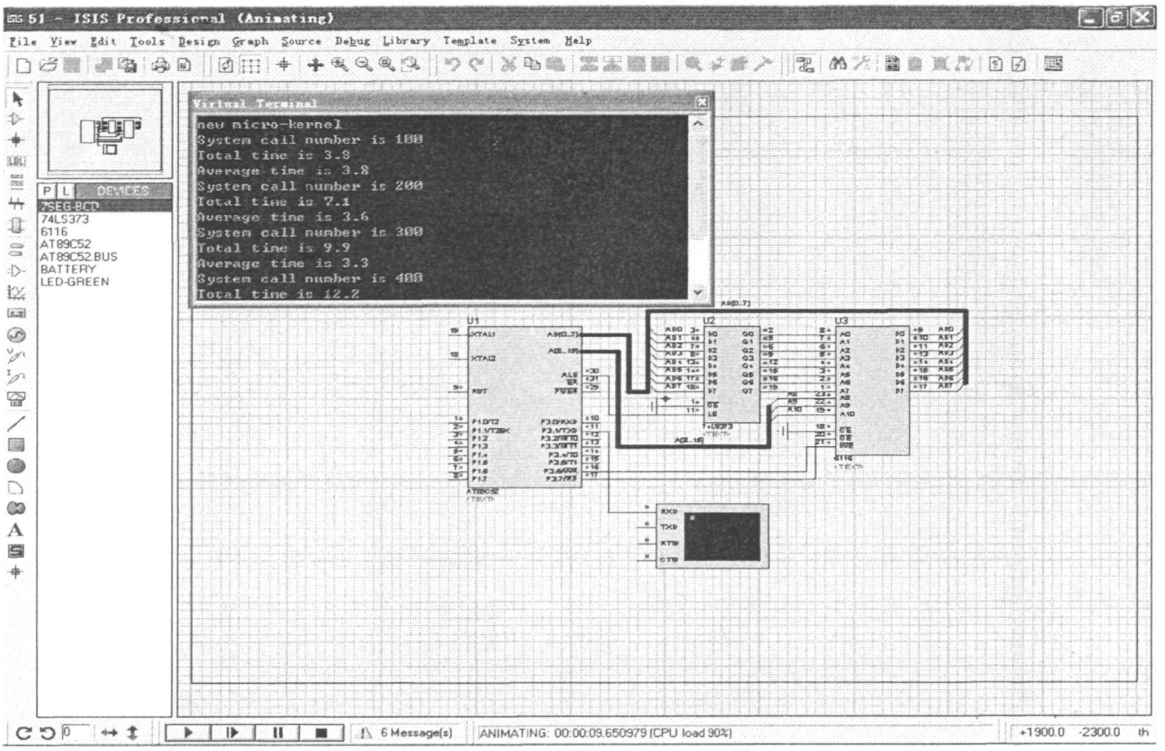

图 5.10 实验结果

表 5.3 微内核 μC/OS-II 调度结果

任务集	系统调用次数	总耗费时间(μs)	平均时间(μs)	任务集	系统调用次数	总耗费时间(μs)	平均时间(μs)
任务集 P1	100	4.1	4.1	任务集 P2	100	7.8	7.8
任务集 P1	200	7.3	3.6	任务集 P2	200	14.3	7.2
任务集 P1	300	10.2	3.4	任务集 P2	300	20.2	6.8
任务集 P1	400	12.4	3.1	任务集 P2	400	24.7	6.2
任务集 P1	500	15.1	3.0	任务集 P2	500	29.6	5.9

表 5.4 新的微内核调度结果

任务集	系统调用次数	总耗费时间(μs)	平均时间(μs)	任务集	系统调用次数	总耗费时间(μs)	平均时间(μs)
任务集 P1	100	3.8	3.8	任务集 P2	100	7.5	7.5
任务集 P1	200	7.1	3.6	任务集 P2	200	13.9	7.0
任务集 P1	300	9.9	3.3	任务集 P2	300	19.8	6.6
任务集 P1	400	12.2	3.1	任务集 P2	400	24.4	6.1
任务集 P1	500	14.6	2.9	任务集 P2	500	29.5	5.9

由以上调度结果可以看出，对于任务周期短的任务集 P1，新的微内核调度算法耗费的平均时间较短，性能提升较明显，对于任务周期长的任务集 P2，性能也得到小幅提升。同时对于新的微内核调度算法，一次处理任务越多，平均执行时间越少，提高了微内核的实时性。

5.6 本章小结

本章首先针对目前广泛运用的嵌入式微内核 μC/OS-II 存在的不足之处进行了改进和实现。以 μC/OS-II 作为原型，实现新的微内核任务调度算法，且通过性能测试实验证明对于任务周期短的任务集，新的微内核调度算法耗费的平均时间较短，性能提升较明显，对于任务周期长的任务集，性能也得到小幅提升。新的微内核的实时性得到了提高。

第6章 总结与展望

6.1 总结

随着嵌入式系统的发展，嵌入式系统技术得到了突飞猛进的进步，嵌入式系统已经广泛的应用到科学研究、工业控制、军事技术以及人们的日常生活等各个方面。

调度算法是嵌入式实时操作系统的核心所在，任务调度算法直接影响嵌入式实时操作系统的实时性和可靠性，因此对嵌入式实时操作系统调度算法进行研究和分析，以提高实时性有着重要的意义。任务调度算法一直是目前研究的热点。本课题主要针对实时性对嵌入式实时操作系统的调度算法进行深入的分析和研究。

本论文主要工作包括：

1、分析了嵌入式实时操作系统的国内外研究现状。并分析了微内核的体系结构和技术特点，及几种常见的嵌入式实时操作系统微内核。研究了微内核的实时调度算法，并对其调度算法做了分类和总结。最后分析比较了静态实时调度和动态实时调度其中的不足之处。

2、对任务集建模，详细叙述了新的优先级的分配方案，此方案通过衡量任务价值和任务时限来决定任务优先级的高低。讨论了实时调度器和中断管理机制，叙述了新的微内核调度算法的改进方案，算法的设计原则是任务的时限越小且价值越大，则任务的优先级越高；对于时限与价值完全相同的任务，先到达者具有更高的优先级。并给出了新的微内核任务调度算法的流程图。

3、对微内核 μC/OS-II 的调度算法做了详细的分析和研究，包括任务控制块，任务就绪表和任务调度器。分析和总结了 μC/OS-II 实时内核优缺点，通过实现时间片轮转算法在内核部分的扩展，优先级继承算法，扩充任务数，改进调度算法，使得构建的实时内核更可靠及具有更强的适用性。

4、以 μC/OS-II 作为原型，实现新的微内核任务调度算法，通过实验证明对于任务周期长的任务集，性能得到明显提升，对于任务周期短的任务集，性能也得到小幅提升，新的微内核的实时性得到了提高。

6.2 展望

本课题的工作还有待进一步提高的完善。接下来还需要做的工作是对调度算法做进一步的研究，任务的调度模型进一步规范化，并在目前实时调度算法的基础上，研究出更高效的实时调度算法。并与系统其他部分如时钟中断及内核抢占等有关内容结合，进

一步提高系统实时性能。μC/OS-II 已经升级到了更高的版本，我们的系统也应该跟着更新。另外，将已有的算法更多地应用于实际应用中。

致 谢

首先感谢我的导师程小辉教授，本文的研究工作以及论文的撰写，是在导师程小辉教授的悉心指导下完成的。从论文研究方向的确立到研究工作的结束，都凝聚了导师的心血。在这段学习生活期间，程小辉老师给予我的悉心指导和谆谆教诲将使我终身受益！程小辉老师开阔的思想、渊博的学识、创新的精神和严谨的治学态度，对我在研究生求学期间所起的作用，以及在今后工作中的帮助是难以用语言来形容的。程小辉老师不仅在学习上、思想上给予我以很大的帮助，而且在生活中也给予了我无微不至的关怀。在此我向尊敬的程小辉老师致以衷心的感谢，并将永远珍惜这份师生情谊！

感谢桂林理工大学的各位老师，牛秦洲老师、蒋存波老师等在我的研究生学习期间对我的指导和帮助。

感谢李泽球、肖富元、周茂杰、陈伟、史丹和师弟师妹们他们对我的论文提出了很多建议。同时要感谢实验室里的每一位同学。我们在一起学习，一起生活。在研究遇到困难时，他们总会给我莫大的鼓励与帮助。他们各有优点，从他们身上我看到了自己的不足，并从他们那里学到了很多。

感谢我的父母，在我的人生道路上，他们永远关心和爱护我，永远地鼓励我，给予我无私的帮助。

最后，再次对所有帮助过我的人表示衷心的感谢！

参考文献

[1] 范学英, 张明新, 王登磊. 嵌入式系统概述. 自动化技术与应用, 2008, (2): 4～5

[2] 郭杨, 王新社. 嵌入式操作系统的硬实时微内核设计. 计算机工程与设计.2008, (1): 115～118

[3] 吴伟. 嵌入式实时操作系统在 ARM 系列微处理器上的移植研究: [硕士学位论文]. 昆明理工大学, 2007

[4] James H. Anderson, Vasile Bud, UmaMaheswari C. Devi. An EDF-based restricted-migration scheduling algorithm for multiprocessor soft real-time systems. Real-Time Systems, 2008, 38(2): 85～131

[5] Tarek F. Abdelzaher, John A. Stankovic, Chenyang Lu, Ronghua Zhang, Ying Lu. Feedback performance control in software services. IEEE Control Systems, 2003, 881～911

[6] GC Buttazzo. Rate monotonic vs. EDF: judgment day. Real-Time Systems, 2005, 5～26

[7] 王晓宇. 实时任务在集群计算中的自适应容错调度研究: [硕士学位论文]. 复旦大学, 2010

[8] 张杰, 阳富民, 卢炎生, 涂刚. EDF 统一调度硬实时周期任务和偶发任务的可调度性判定算法. 小型微型计算机, 2009, (12): 2383～2388

[9] 萧伟, 冯治宝, 应启戛. 改进型 EDF 调度算法的研究与实现. 计算机工程, 2009, (9): 231～233

[10] 罗钧, 吴志. 嵌入式系统动态策略任务调度算法. 重庆大学学报, 2008, (7): 792～796

[11] 吴景峰. 嵌入式实时操作系统-MKRTOS 的实现与调度策略设计: [硕士学位论文]. 华东师范大学, 2006

[12] 杨嘉, 王移芝. Linux 内核调度器算法研究与性能分析. 计算机技术与发展. 2006, (3): 95～97

[13] 冯艳红, 张玉明, 徐美华. 实时调度算法分类研究. 微型电脑应用, 2005, (21): 12～14

[14] 付耀. 嵌入式实时操作系统的进程间通信: [硕士学位论文]. 华中科技大学, 2008

[15] 鲁莹. 基于 ARM 的嵌入式 Linux 和 MiniGUI 的研究与移植实现: [硕士学位论文]. 昆明理工大学, 2006

[16] Xiao-hui Cheng, Ming-qiang Li, Xin-zheng Wang. Embedded real-time operating system micro kernel design. ICMIT2005:Information Systems and Signal Processing, Published by SPIE-The International Society for Optical Engineering, 2005,, 246～249

[17] Xiao-Hui Cheng, You-min Gong, Xin-zheng Wang. Study of Embedded Operating System Memory Management. 2009ETCS:Computer Science, Published by the IEEE Computer Society, 2009, 962～965

[18] Xiaohui Cheng, Yongjian Hu, Youmin Gong. Design Methodology of Control System with Mutual Exclusion. 2008PACIIA:Intelligent control, Published by the IEEE Computer Society, 2008, 403～407

[19] Andre Brinkmann, Dominic Eschweiler. A microdriver architecture for error correcting codes inside the Linux kernel. 2009, 1674～1685

[20] Ayse Kivilcim Coskun, Tajana Simunic Rosing, Keith Whisnant: Temperature aware task scheduling in MPSoCs. 2007, 1659～1664

[21] Shuai Wang, Yindong Ji, Shiyuan Yang. A Micro-Kernel Test Engine for Automatic Test System. Journal of Computers, 2011(1): 3～10

[22] Dario Faggioli, Giuseppe Lipari, Tommaso Cucinotta. An efficient implementation of the BandWidth Inheritance protocol for handling hard and soft real-time applications in the Linux kernel. Proceedings of the Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, 2008(12): 467～471

[23] Dario Faggioli. An EDF scheduling class for the Linux kernel. proceedings of the 11th Real-Time Linux Workshop, 2009, 1～8

[24] Luis Fernando Friedrich, Centro Tecnologico. A Survey of Operating Systems Infrastructure for Embedded Systems. http://www.di.fc.ul.pt/tech-reports

[25] François Armand, Michel Gien. A practical look at micro-kernels and virtual machine monitors. Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, 2009, 1～7

[26] Myung-Kyun Kim, Liang Shan and Wang Yu. EDF-based Real-time Message Scheduling of Periodic Messages on a Master-Slave-Based Synchronized Switched Ethernet. Communications in Computer and Information Science, 2009, 65(10): 62～70

[27] Moris Behnam, Malardalens Hogskola. Hierarchical Real Time Scheduling and Synchronization. University dissertation from Vasteras : Malardalens hogskola, 2008, 134～138

[28] Lei Sun, Hiroo Ishikawa, Tatsuo Nakajima. Kernel Data Structure-based Runtime Monitoring. 2008, 468～472

[29] 李仁发, 刘彦, 徐成. 多处理器片上系统任务调度研究进展评述. 计算机研究与发展, 2008(9): 1620～1629

[30] 周学功, 梁堰, 黄勋章等. 可重构系统中的实时任务在线调度与放置算法. 计算机学报, 2007, (11): 1901～1909

[31] 梁堰, 周学功, 王颖等. 采用预配置策略的可重构混合任务调度算法. 计算机辅助设计与图形学学报, 2007, (5): 635～641

[32] 宋丰末. 关于 RTOS 抢占式调度及优先级反转的几点探讨. 计算机工程与应用, 2007, (19): 4719～4720

[33] 王田苗, 魏洪兴编著. 嵌入式系统设计与实例开发. 清华大学出版社, 2008

[34] 邵贝贝译. 嵌入式实时操作系统 μC/OS-II (第二版). 北京航空航天大学出版社, 2003

[35] 孙玉芳主编. 嵌入式计算系统设计原理. 北京:机械工业出版社, 2002

[36] 覃中. 一种实时嵌入式多任务微内核的分析与改进: [硕士学位论文]. 西安电子科技大学, 2006

[37] 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. 计算机应用研究, 2005, (9): 4～8

[38] 李瑾. μC/OS-II 操作系统的研究与应用: [硕士学位论文]. 武汉理工大学, 2007

[39] 刘铁志. μC/OS-II 在嵌入式系统中的研究与应用：[硕士学位论文]. 成都电子科技大学, 2005

[40] 余婷婷. 嵌入式文件系统的研究与设计：[硕士学位论文]. 武汉理工大学, 2007

[41] Ani Nahapetian, Foad Dabiri, Miodrag Potkonjak,and Majid Sarrafzadeh. Optimization for Real-Time Systems with Non-convex Power Versus Speed Models. In: N. Azemard, L. Svensson,eds. PATMOS, 2007, 443～452

[42] Enrico Bini, Giorgio Buttazzo. The Space of EDF Deadlines: the exact region and a convex approximation. Real-Time. Systems, 2009, 41(1): 27～51

[43] Vincenzo Bonifaci, Ho-Leung Chan, Alberto Marchetti-Spaccamela, Nicole Megow: Algorithms and Complexity for Periodic Real-Time Scheduling. SODA, 2010, 1350～1359

[44] Ryo Watanabe, Masaaki Kondo, Masashi Imai, Hiroshi Nakamura, Takashi Nanya. MP-SOC Task Scheduling under Performance Constraints for Reducing the Energy Consumption of the GALS Multi-Processor SoC. Proceedings of the conference on Design, 2007, 234～241

[45] Mikael Asberg, Moris Behnam, Thomas Nolte, Reinder J. Bril. Towards Hierarchical Scheduling in VxWorks. Master Thesis, 2008, 897～901

[46] 袁云. 基于多核处理器并行系统的任务调度算法研究：[硕士学位论文]. 华东师范大学, 2009

[47] 张山刚. 微处理器验证平台的实现：[硕士学位论文]. 西北工业大学, 2005

[48] 王莹莹. 实时操作系统 μC/OS-II 内核分析与移植：[硕士学位论文]. 重庆大学, 2007

[49] 毛德梅, 何建忠, 汪明珠. μC/OS-II 中任务切换机理及中断调度技术研究. 微计算机信息, 2007, 23(10): 45～47

[50] 熊玉梅, 陈一民. μC/OS-II 任务调度算法的改进. 计算机应用与软件, 2008，25(6): 84～89

[51] 万柳, 郭玉东. 嵌入式 RTOS 中就绪任务查找算法和优先级反转的解决方案. 计算机应用, 2003(6): 49～51

[52] 刘云生，方丹. 嵌入式实时操作系统的内核抢占机制研究. 计算机工程,2005，31(24): 80～104

[53] 黄石，张发存. 粗粒度可重构并行计算的面向对象仿真研究. 计算机工程与设计，2009，30(11): 2737～2787

[54] 王红玲，吕强，褚亚铭. 一个微内核操作系统的设计与实现. 微电子学与计算机.2008(4): 9～17

[55] 陈杰. 一种微内核任务调度的实现方法. 机电工程，2010，27(5): 78～81

[56] 徐进辉，杨梦梦，窦勇，周兴铭. 粗粒度可重构平台中循环自流水硬件实现. 计算机学报, 2009, 32(6): 1080～1088

[57] 于淑英，黄皓，刘国斌. 微内核完整性保障研究与应用. 计算机科学, 2009, 36(1): 247～251

[58] 李向蔚，桑 楠，熊光泽. 基于软件复用的嵌入式操作系统的定制. 电子科技大学学报, 2007, 36(3): 559～562

[59] 陈文智,谢铖,石教英. 一个构件化嵌入式操作系统的精确控制内核. 计算机学报,2006, (6): 867～874

[60] 姚君兰. 增强 Linux 内核实时任务调度性能的研究. 微计算机信息, 2006, 22(5): 42～44

[61] 齐骥, 李曦, 于海晨等. 一种面向动态可重构计算的调度算法. 计算机研究与发展, 2007(8):

1443～1447

[62] 高富强, 秦昌硕, 游纪原, 邹恒. μC/OS-II 内核扩充时间片轮转调度算法的设计. 微电子学与计算机. 2009, (4): 1128～1142

[63] 苗蕾, 齐勇, 侯迪, 钟婉, 郑小梅. 基于遗传算法的片上多处理器任务调度策略研究. 微电子学与计算机. 2007, 24(6): 8～15

[64] 王婧怡, 应忍冬, 周玲玲. 微内核系统直接加载 ELF 文件机制的设计与研究. 信息技术. 2009(10): 73-76

[65] 苗硕, 马光思. 基于并行遗传算法的对称多处理器任务调度策略研究. 微电子学与计算机. 2006, 23(6): 181～184

[66] 刘勇, 尹增山, 杨根庆. 嵌入式并行系统中基于任务优化的调度算法. 计算机工程, 2008, 34(2):11～14

[67] 刘忠仕, 戴金海, 桂先洲. 单调速率调度算法的可调度性分析与仿真. 计算机仿真, 2006, (3): 78～80

[68] 贺小川, 贾焰. FPTS：一种任务间存在共享资源时的抢占阈值调度算法. 计算机研究与发展, 2009, (2): 302～309

个人简介

张一哲，女，1985年7月出生于江西省南昌市，2008年7月毕业于西安电子科技大学计算机学院计算机科学与技术专业，获工学学士学位。2008年9月考入桂林理工大学攻读硕士学位，师从程小辉教授，研究方向为嵌入式系统。

1. 攻读硕士学位期间发表的学术论文：

[1] 张一哲. 基于多核 SoC 的负载平衡任务调度算法优化策略. 软件导刊. 第一作者.2011.5

2. 攻读硕士学位期间参与的科研工作：

[1] 项目来源：广西区自然科学基金项目（桂科自 0832264）
项目名称：嵌入式操作系统微内核体系结构与实现模式研究
[2] 项目来源：广西研究生教育创新计划项目
项目名称：基于事件调度的嵌入式操作系统内核的研究与设计（2010105960812M33）