太原科技大学

硕士学位论文

# 基于OSEK/VDX的操作系统设计及任务调度算法的优化

姓名：毛成勇

申请学位级别：硕士

专业：计算机应用技术

指导教师：高慧敏

20090701

# 摘 要

随着科技的进步及微电子技术的迅速发展，微处理器的价格逐渐下降，性能与可靠性日益提高。汽车应用软件的开发变得更加复杂和困难。各汽车生产商在开发汽车应用软件时采用的标准及接口不一致，造成汽车应用软件移植十分困难，甚至于不可能移植。

为了解决以上问题，欧洲汽车行业于1995年提出了OSEK/VDX规范。基于该规范的汽车应用软件可以实现现有软件的重用，具有良好的可移植性及扩展性。我国已经将符合OSEK/VDX规范的汽车嵌入式软件平台的开发及其关键技术的研究作为重大研究项目。在汽车软件开发中符合OSEK/VDX规范，促进我国汽车行业迅速发展并打入国际市场，是我国汽车业界所面临的紧迫任务。

本文在研究OSEK/VDX规范的基础上，完成了实时操作系统内核的设计和主要的数据结构。分析了实时系统中有代表性的静态以及动态调度算法，在此基础上，提出了一种任务管理机制，和任务调度算法。该算法要求在建立任务时确定任务的时限，当任务的时限到达时，提高其优先级，保证该任务得以运行；从而使系统在保证高优先级任务得以运行的同时，保证低优先级任务和同优先级下的多个任务得以运行，保证了系统的实时性。

关键词：OSEK/VDX，系统设计；任务调度；优化

# ABSTRACT

With the rapid development of science and micro-electronics technology, the price of microprocessor is declining, while its performance and reliability is improving. Therefore, the development of automotive application software is more and more complex and difficult. Different standards and interfaces adopted by different automobile manufacturers when developing automotive application software have caused great difficulty when porting the software or even make it impossible.

In order to resolve the issues discussed above, the European automobile industry put forward OSEK/VDX in 1995. OSEK/VDX embodies the advanced technology of international automobile manufacturing. Automotive application software conforming to this standard has better portability, extend ability, as well as reusability. China has already referred to the development of automotive embedded software platform conforming to the OSEK/VDX and the research into the key technologies related as important research project. Conforming to OSEK/VDX when developing automotive software, promoting progress in national automotive industry, and putting our automobiles into the international market are emergent tasks faced by national automotive industry.

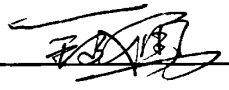
This paper presents the design and the main data structure of real-time operating system kernel conforming to OSEK/VDX. In embedded system, to a large extent, performance of the system depends on the quality of the task scheduler. The paper analyzes static and dynamic scheduling algorithm which is representative in the real-time system. This paper proposes one kind of task management mechanism and the task scheduling algorithm, which set up a list structure, this structure can avoid overhead when the task scheduling, and establish the time limit of the task when it was established. When the time limit of the task approaches, its priority will be enhanced, so the task can run successfully. Using the algorithm in this paper, the high priority task of

the system can run, meanwhile the low priority task can run in time, and so the real-time of system can be guaranteed.

**Key words:** OSEK/VDX; Design of system; Task scheduling; Optimization

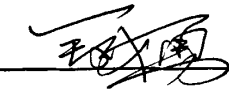
## 声　　明

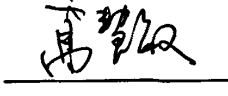
本人郑重声明：所呈交的学位论文，是本人在指导教师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

作者签名：  日期：2009.4.23

## 关于学位论文使用权的说明

本人完全了解太原科技大学有关保管、使用学位论文的规定，其中包括：①学校有权保管、并向有关部门送交学位论文的原件、复印件与电子版；②学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③学校可允许学位论文被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换学位论文；⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

作者签名：  日期：2009.4.23

导师签名：  日期：2009.4.23

# 第一章 引言

## 1.1 国内外汽车电子的现状

近 30 年来,汽车产品发展和技术进步所取得的重大成就，几乎每一项都和汽车电子技术的应用相关。可以说,汽车电子化程度的高低已成为衡量汽车综合性能和现代化技术水平的重要标志。自 1973 年美国通用汽车公司第一次在汽车上采用电子点火装置以来,汽车电子技术的发展十分迅速,已由少数零部件扩大到许多总成及整车。如今,汽车电子产品已广泛应用于发动机、传动、悬架、制动、仪表等。

### 1.1.1 国外汽车电子的现状

（1）发动机

■ 汽油机电子控制

汽油机采用电子燃油喷射技术[1](EFI)是现代汽车提高功率、降低油耗、减少污染的有效措施之一。EFI 技术是一种高级的发动机电子管理系统,其基本工作原理是:由传感器将汽油机的工作运行状况,如负荷大小、转速快慢、进气温度等数据送给计算机进行处理,然后由控制执行元件来确定供油量,从而保证发动机在各种工情况下的正常运行。当今的 EFI 已由单一控制发展到多项集中控制,根据汽车速度、环境温度和发动机转速等参数,自动对发动机的燃油喷射、空燃比、点火时间、怠速转速和废气再循环(EGR)等进行综合控制。其效果为:输出功率提高 10%,油耗下降 10%,尾气排放降低 90%,起动时间缩短 50%,加速时间缩短 50%(0～100lKm/h)。

■ 柴油机的电子控制

柴油机电控技术当前主要应用于柴油喷射、废气再循环、尾气催化转化等方面。目前先进的柴油机电子技术是共轨式电控喷油系统。它由电磁阀、传感器及中央处理单元组成。与传统喷射系统不同,在共轨系统中,压力的产生和喷射互不关联。在一定限度内,喷射压力的选择不受发动机转速和喷油量的影响。

（2）变速器

汽车电控自动变速器的应用满足了人们追求操纵简便及燃油经济的要求。电控自动变速器通过电子技术实现自动换档、自动控制离合器及油门的开度,并能自动选择最佳档位和最佳换档时间。有三种操纵方式:电子控制气动操纵,电子控制液压操纵和电子控制马达操纵。

（3）制动系统

防抱死制动系统[2](ABS)是80年代后半期开始应用,90年代日益普及的一种机电一体化的高科技车用安全装置,目前有的车型装车率已达100%。ABS由中央控制器、传感器、电磁阀和电缆等四部分组成,一般采用逻辑门限控制。随着汽车电子技术的进步,ABS的性能也在不断改进,并增加了许多功能,如ASR驱动防滑系统。最新问世的ABS已是第五代产品,具有结构紧凑、体积小等特点。零件数量已减少到原来的1/3,成本降低了2/3。

(4) 转向系

电控动力转向系统能有效地改善车辆的转向控制特征,减轻低速行驶时的转向操纵力和提高高速行驶时的稳定性,还可提高车辆的经济性。典型的电控动力转向系统主要由扭矩传感器、车速传感器、控制器、机电离合器和减速机构等组成。新型电控动力转向系统,能自动改变方向盘的操作手感,使手动扭力之比能视车速的变化随时调节,低速或弯道时转向变轻,加速或高速时转向变重。

(5) 悬架

在汽车的悬架方面,已有以微机控制的车身高度自动调节和减振器阻尼力自动调节装置,即电子控制的主动悬架系统。该系统将车速、车辆振动加速度、车身距路面的高度、转向轮角速度等参数通过传感器输送给车载计算机,计算机综合处理后输出控制信号,调整悬架的刚度和阻尼系数,从而提高汽车的操纵稳定性和乘座的舒适性,使得汽车以最小的车身姿态变化。如转向时调整悬架高度以避免侧倾,大大提高舒适性。

(6) 安全气囊

安全气囊是90年代以来汽车产品技术发展的重大成就之一。日本、美国和西欧的许多汽车厂家已逐步把安全气囊作为汽车的标准件装备。今后的技术发展是司机和前乘客正面双极起爆气囊、翻滚探测、早期预测碰撞等技术。

(7) 车内温度和气流的电子控制

电子控制自动空气调节可对车厢内的温度预选,并可对局部温度、车内气流等控制作自动调节,从而大大提高乘员的舒适性。

(8) 通讯和导航

电子控制的高性能导航系统(GPS),装有电子地图、车载电话,可向驾驶员提供相关的行驶资料,快捷地达到目的地。

目前汽车电子产品的应用数量正不断增多,许多工业发达的国家都已形成了独立的汽车电子产品。电子产品在整车中的价值也逐步上升。1991年,一辆汽车上电子

装置的价值为 825 美元,1995 年上升到 1125 美元,现已达 2000 美元,占整车成本的 30%以上。国外各大汽车公司为进一步提高产品水平和竞争力,在汽车电子产品的研究与开发上十分重视,投入了大量资金,如日本丰田公司就高达 15 亿美元的投资。

今后汽车电子技术的重点发展将是系统模型、电源系统、多通道信息处理系统、汽车电子软件及故障自诊断、智能化技术等。

### 1.1.2 国内汽车电子的现状

(1) 量价双增,市场规模增长较大、结构相对稳定

2003 年我国市场上共销售汽车电子产品 2488 万套,销售金额达到 300.4 亿元,同比增长分别达到 33.7%和 41.5%,高于同一时期 29.2%的全国汽车总销量增长率[2]。

汽车电子产品的快速发展主要得益于中国汽车产业近两年的高速发展。继 2002 年汽车产业的井喷之势后,2003 年中国汽车产量依然保持着高速发展,全年汽车产量突破了 430 万辆。而在汽车价值构成中,轿车产品电子产品所占比例约为 30%-40%,卡车电子产品所占比例约为 10%~20%。

另一方面,随着电子信息技术的不断升级和应用使得机械系统与电子系统之间的转换加速,动力总成性能大大提高,网络在车辆中的高速发展以及消费者消费标准与档次的提高(如对安全、先进性要求的提高)等使得汽车电子产品占整车中的比重在不断提高,从支出费用来看,2003 年平均每辆新车的汽车电子产品支出从 2002 年的 1854 美元提升至 2025 美元,增幅创出近几年来的新高,达到 9.2%;从应用比例上看,从 1989 年至 2000 年,汽车电子产品的成本在整个汽车制造成本中所占的比例已由 16%增至 23%,而到了 2003 年,这一比例提升至 26.2%。

从市场结构上看,高附加值汽车电子产品在整体汽车电子产品中所占比重乃至汽车整车中的应用比重有了显著的提升。同时,这部分高附加值产品的国内产业供给还极为有限,多类产品大量依赖进口,导致这部分产品的价格在一段时期以内持续看涨。2003 年汽车电子产品呈现出“量价双增”的增长特征,这是不同与于汽车市场的。

此外,与 2002 年相比,汽车电子产品的市场结构相对稳定。2003 年,整车消费侧重于中低端市场,汽车电子产品自身的结构升级要求在短期内受到了一定程度的压制。部分技术含量与附加值较高的产品(如发动机电子控制、防抱死制动系统、安全气囊等)在汽车电子产品整体中的比重有所下滑。但诸如 GPS 导航系统以及其他信息娱乐系统(主要是车载电视)等新兴汽车电子产品的比重 2003 年分别比 2002 年

增加0.7%和0.5%。

（2）产业规模增长速度略有下降，产业结构快速升级

2003年我国共生产汽车电子产品3559万套，产值达到313.5亿元，同比增长分别达到26.8%和37.2%，低于同一时期全国汽车电子产品市场增长率。值得注意的是，产值增幅远大于产量增幅，“量价双增”特征比市场需求更为显著，这主要是由于国外企业在中国加大了对高附加值汽车电子产品的投入力度，产业结构快速升级所导致的。

在产业结构方面，按照增长速度可以划分为三大梯队：2003年产值增长较快的产品依次为GPS导航系统、其他信息娱乐系统（主要是车载电视）、自动变速控制系统、发动机电子控制系统、防抱死制动系统、以及防盗系统等，它们的产值增长率均高于汽车电子产业37.2%的平均增长水平；而安全气囊、汽车空调、电动车窗、电子组合仪表等产品的产值增长率虽然均略低于汽车电子产业的平均增长水平，但增长速度也较快，均高于30%；仅有汽车音响和中控锁的产值增长率较低，不足20%。

（3）应用层次落后，与国外差距明显

与国外汽车电子产品相比，我国汽车电子产品的应用层次十分落后。国内平均每辆新车的汽车电子支出为885美元(折合人民币7237元)，占整车价值的4.8%；而同期全球平均每辆新车的汽车电子产品支出为2025美元，占整车价值的26.2%，差距相当明显。

（4） 自主开发刚刚起步，合资合作企业成为竞争的主体

我国汽车电子企业现正处于起步发展阶段，据统计，我国汽车电子企业或涉及汽车电子生产的企业有1000多家，但绝大部分企业规模都太小，产品单一并且技术含量低，与世界汽车电子业跨国公司的差距在不断拉大。目前，国内在整车及大总成的电子控制系统方面有一些自主开发生产的产品，如重庆宏安ABS技术有限公司、维克交通高技术有限公司、西安国力博华机电公司等生产了商业化的ABS产品；重庆欧翔、深圳欣源展实业公司都自主开发生产了自动变速器；北京赫达、石家庄久乐、山西大恒通用都少量生产了安全气囊产品。虽然我国企业也能自主开发出一些产品，但总体水平与国外相比差别很大，与国际先进水平相比，要落后10～15年。主要差距是在电子控制单元的软硬件、系统的可靠性和控制精度方面，企业的技术主要来源于国外，而且虽然某些产品已形成一定的生产能力，但是规模化大生产还未形成，除了技术和规模两个明显的差距，与先进的跨国公司相比，不断拉大的差距还表现在：盈利水平、投融资能力、产品管理水平、生产效率、人才素质、发展

战略、持续改进能力、市场开拓、知识积累、研究开发能力等多个方面。

同时，面对我国潜在的巨大汽车电子产品市场，国外汽车电子供应商已积极介入我国汽车电子领域。近几年新成立的汽车电子企业中，合资企业约占七成，合作伙伴中有许多是汽车电子零部件行业的跨国公司，这些企业已经成为我国汽车电子产业发展的主要力量，可以说，一个中外合资合作的汽车电子零部件产业竞争发展格局正在形成，而且这种竞争程度在愈演愈烈。

（5） 配套体系相对封闭，行业进入壁垒较高

尽管全球汽车电子产业分工体系呈现出日益专业化的趋势，但目前，我国汽车工业条块分割的封闭配套体系还没有根本性的转变，我国汽车电子企业产品销售渠道依旧单一、封闭，对整车厂依赖程度高。而目前国内整车厂与外方合资已成主流，引进外国新车型，越来越多地采用原配套厂生产的进口零部件。尽管国际汽车厂商在进入中国之前都承诺，零部件在中国生产，实行全球化采购。但一旦进入，与其配套的零部件厂商就带进来了，对于缺乏统一质量标准的国内零部件企业要想进入它的配套体系甚至比插针还难。

## 1.2 汽车电子软件所面临的挑战

随着全球汽车行业竞争的加剧，汽车电子产品和软件市场已经成为国际汽车巨头开展竞争的新战场;而同时随着汽车电子系统复杂度的提高，以往一个企业单打独斗、“垂直综合型”产品开发模式已经很难满足复杂开发任务和成本目标的需要。汽车电子化进程带来的新问题可概括如下[2]:

（1）随着车载控制系统日益先进和复杂化，软件开发工程师所面临的工作量和压力也越来越大。据专家估计，每辆汽车投入的软件开发工数（时间及人力）在2010年将达到2002年的5－10倍。

（2）汽车电子系统设计复杂化和单个企业“垂直综合型”开发模式造成了可靠性隐患，从而导致汽车因安全隐患被“招回”的现象频繁发生。一些国际顶级汽车制造商已经因此倍感痛苦。

（3）随着大量的嵌入式产品涌向市场，汽车产品开发商将面临着激烈的市场竞争。谁能在最短的时间内推出高质量的产品，谁就能抢占先机。缩短开发周期，提高开发效率，成为汽车产品开发商重点关注的问题。

上述问题归结到一点，就是车用控制软件缺乏重用性和兼容性。在开发和管理与应用不相关的控制单元软件时，尤其是像车用嵌入式操作系统内核等软件，存在

重复的巨额开销。因为不同的接口和协议，不同厂商生产的控制单元没有兼容性。

如果能规范不同供应商之间汽车电子产品和元器件的互通性，实现汽车电控软件的接口标准化，那么汽车电子行业就可建立国际间广泛的合作关系，分享汽车电子领域的技术积累，向“水平分业型”开发方式转变，从而缩短产品开发时间、降低开发成本，从根本上提高产品的可靠性。国际业界早已着手开始解决这些问题。提出于 1993 年的 OSEK/VDX 体系，它的开放式体系结构提供了标准化接口。经过十多年的发展，OSEK/VDX 已经引起了全球的强烈关注。它的主旨是从已有软件的可移植性、可扩展性和可重用性角度出发，采用基于标准化接口和协议的构件化、模块化设计。同时，OSEK/VDX 还允许不同供应商把不同的模块集成在汽车电控系统中。

另据日本媒体报道，丰田汽车公司与日产汽车公司日前宣布将联手开发汽车电子控制软件，以促进电子控制刹车及发动机燃料喷射等装置的基础软件实现标准化并提高其通用性，降低软件开发的成本。

# 第二章 嵌入式操作系统

## 2.1 嵌入式系统的基本概念

### 2.1.1 嵌入式系统

嵌入式系统是以应用为中心，以计算机技术为基础，同时结合半导体技术、网络技术和通信技术等具体应用的产物，是对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。目前，嵌入式系统已经应用到工业控制、国防军事、交通通信、医疗卫生以及人们日常生活的各个方面，如数控机床、汽车、航天器、PDA、手机、智能家电等产品中都包含嵌入式系统[3]。

### 2.1.2 嵌入式系统的定义及特点

作为20世纪人类社会最伟大的发明之一，计算机在期待着第五代出现的同时，也迈入了其另一个新阶段——后PC时代。后PC时代的到来，使得人们开始越来越多地接触到一个新的概念——嵌入式产品。其实嵌入式系统（Embedded System）很早以前就出现并得到广泛的应用，如微型计算器、PC中的键盘，都是最典型的、简单的嵌入式系统[4,5]。

如果说PC机的出现构建了信息产业的框架，推动了整个信息产业和人类文明的发展和前进，那么嵌入式系统的发展将会更加完善这个框架，并成为信息产业发展的加速器。中国拥有世界上最大的消费电子产品市场。随着经济水平的提高和消费观念的改变，人们对消费电子产品的要求越来越高，如产品的灵活性，可控性、耐用性、高性价比等，这些都可以通过合理、有效的嵌入式系统设计和优化来实现。另外，在现代化的医疗、测控仪器和机电产品中对系统的可靠性、实时性要求较高，更需要有专用的嵌入式系统的支持，这些需求都极大地刺激了嵌入式系统的发展和产业化的进程。

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种技术相结合后的产物，这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

嵌入式系统具有以下特点：

（1） 嵌入式系统面向特定应用

嵌入式系统是面向用户、面向应用的，一般会与用户和应用相结合，以其中的

某个专用系统或模块出现。嵌入式系统和具体应用有机结合在一起，它的升级换代也是和具体产品同步进行，因此嵌入式系统产品一旦进入市场，就具有较长的生命周期。

（2） 嵌入式处理器受到应用要求的制约

嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用中更具有竞争力。与通用型处理器相比，嵌入式处理器的最大不同是将大部分工作用在为特定用户群设计的系统中，它通常都具有低功耗、体积小、集成度高的特点，能够把很大任务集成在芯片内部、从而有利于嵌入式系统设计趋于小型化，移动能力大大增强、与网络的联系也越来越紧密。

嵌入式处理器功耗、体积、成本、可靠性、速度、处理能力、电磁兼容性等均受到应用要求的制约[6]。例如对于嵌入式微处理器就具备以下 4 个特点：

■ 对实时多任务有很强的支持能力，能够完成多任务并且有较短的中断响应时间，从而使内部的代码和实时内核的执行时间减少到最低限度。

■ 具有很强的存储区保护功能。这是出于嵌入式系统的软件结构已模块化，而为了避免在软件模块之间出现错误的交叉作用，需要设计强大的存储区保护功能，同时也有利于软件诊断。

■ 可扩展的处理器结构，能最迅速地开发出满足应用的最高性能的嵌入式微处理器。

■ 嵌入式微处理器必须功耗很低，尤其是用于便携式的无线及移动的计算和通信设备中靠电池供电的嵌入式系统更是如此，功耗只是 mW 甚至 uW 级。

（3） 嵌入式系统要求固化、可靠

嵌入式应用软件是实现嵌入式系统功能的关键，为了提高执行速度和系统可靠性，嵌入式软件一般都固化在存储器芯片或单片机本身中，而不是存储在磁盘等载体中；软件代码要求高质量、高可靠性和高实时性。

嵌入式系统本身不具有自举开发能力，即使设计完成以后，用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发。

### 2.1.3 嵌入式系统的基本结构

一个嵌入式系统包括硬件和软件两部分。硬件包括微处理器、存储器及外围设备器件等，而软件包括操作系统软件（OS）和应用程序。为了提高系统性能，除了

需要对软件和硬件进行优化外，还会引入总线的概念。

（1） 硬件

与普通计算机系统类似，嵌入式系统仍然包括微处理器（运算器和控制器）、存储设备（内存储器和外存储器）和输入、输出设备，但是它与一般的 PC 系统还有很大的区别。一般来讲，嵌入式系统在功耗、体积、成本、可靠性、处理能力、电磁兼容性等方面均受到应用要求的制约。

（2） 软件

嵌入式软件是数字化产品的核心，PC 的出现使桌面软件得到了飞速发展，而数字化产品的广泛普及必将为嵌入式软件产业的蓬勃发展提供无穷的推动力。操作系统控制着应用程序和硬件之间的交互作用，应用程序控制着系统的运行。

随着嵌入式应用系统架构的开发越来越复杂，嵌入式系统软件的开发往往开始的很晚，导致软硬件的集成被推迟，所以，很可能导致研发出有设计错误的产品，错失市场机会，造成设计反复、成本高昂等问题。因此，软件成为设计开发的关键，软件需要在标准化平台上运行，软件发展需要工具支持，软件发展不断促进标准的建立，如 MIPI、OpenGL、OSEK/VDX 等[7,8]。

## 2.2 嵌入式操作系统

嵌入式操作系统[9,10]是嵌入式应用和传统操作系统结合的产物，是需求和复杂度增加的必然结果，是产品上市时间和可维护性的时代要求。嵌入式系统从一开始便在通信、交通、医疗、安全方面表现出强大的魅力和强劲的发展潜力。

### 2.2.1 嵌入式操作系统概述

嵌入式操作系统[11,12,13,14]就是以应用为中心，以计算机技术为基础，软件硬件可裁剪，适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。简而言之，它是一个计算机硬件和软件的集合体。软件部分包括操作系统软件和应用程序，应用程序控制着系统的运行和动作，而操作系统控制着应用程序与硬件的交互作用。嵌入式操作系统是嵌入式系统的灵魂，它的出现大大提高了嵌入式系统的开发效率。按照层次的不同可以将它分为三个部分：最小的基本部分包括一个用作引导的可用设施，一个具备内存管理、进程管理、定时器服务的微内核，一个初始进程；为了具备一定的实用性，还应加上硬件驱动程序，一个或几个应用进程完成必须的功能；随着对系统要求的增加，一般还包括一个文件系统（放在 ROM

或 RAM 中)、TCP/IP 协议栈等。

### 2.2.2 嵌入式操作系统的分类

嵌入式操作系统的任务是控制和管理计算机系统内各硬件和软件资源，合理有效地组织计算机系统的工作，为用户提供一个使用方便且易于扩展的工作环境。嵌入式操作系统的分类标准有很多种[15,16,17]，而其中最基本的嵌入式操作系统类型有三种，即多任务操作系统、分时系统和实时系统。但在一个实际的操作系统中可能同时兼有三者或者两者的功能。

（1） 多任务操作系统

多任务系统种，用户提交的作业都先放在外存上排成一个队列，再有操作系统中的调度程序按照一定的算法从外存中调入内存，使得调入内存的作业共享 CPU 和系统中的各种资源，以达到提高资源利用率和系统吞吐量的目的。该操作系统具有以下特征：

- 并行性

在内存中可同时驻留多个程序，这些程序可同时并发执行，从而有效地提高了资源利用率和吞吐量。

- 调度性

一个作业由开始提交给操作系统到完成，要经过作业调度和进程调度两个调度过程。前者将作业由外存后备队列中调度到内存，后者从内存中选出该作业，并将处理机分配给它。

- 无序性

作业完成的先后顺序和它们进入内存的先后顺序无关，先进入的可能最后完成，相反，后进入的可能首先完成。

多任务操作系统也存在缺点：由于作业首先要进行排队，然后进行处理，导致作业的平均周转时间长；另外就是无交互能力，对程序的修改和调试都极不方便。

（2） 分时系统

基于共享计算机和人机交互方便性两个因素的考虑，产生了分时系统。分时系统能令用户觉得自己好像是独占一台计算机，并可以对它进行直接控制，能够方便地修改错误。用户还可以通过自己的终端把作业提交到主机上运行，能对主机的程序进行控制。分时系统具有以下特征：

- 并行性

允许在一台主机上连接多台终端，系统按分时原则为每个用户提供服务。宏观上是多个用户同时工作，共享系统资源，实际上是每个用户作业轮流运行一个时间片。

■ 独立性

每个用户各自独占一个终端，相互间是独立的，因此从用户的角度看好像是一个人在使用主机。

■ 交互性

用户可以通过终端与系统进行广泛的人机对话，即用户可以请求分时系统提供多方面的服务，如文件编辑、数据处理等。

■ 及时性

分时系统要求用户的请求能在很短的时间内得到响应，而这个时间间隔应该是以人们能够忍受的范围内的等待时间来决定的。

（3） 实时系统

嵌入式实时系统[18]是一种能够进行实时计算的嵌入式计算机系统。“实时”并不等于“快”，实时系统的经典定义是：系统的正确性不仅依赖于计算结果逻辑上的正确，还依赖于此结果产生的时机是否正确。实时系统分硬实时和软实时两种。

在软实时系统中系统的宗旨是使各个任务运行的越快越好，并不要求限定某个任务必须在多长的时间内完成。

硬实时操作系统要求系统能够在确定的时间内执行相应的功能，并对外部的异步事件做出响应。其操作的正确性不仅依赖于逻辑设计的正确性，同时与这些操作进行的时间有密切的关系。“在确定的时间内”是硬实时操作系统的核心因素。也就是说，硬实时操作系统对响应时间有严格的要求。

实时系统具有及时性、交互性和独立性的特征。实时系统对响应时间的要求比分时系统高，分时系统的响应时间通常为秒级，而实时系统的响应时间是以控制对象能够接受的延迟时间来确定的，可能是秒级，也可能是毫秒级，甚至是微秒级。实时系统也保留了通用操作系统的交互性，但是它仅允许操作员访问其中有限的专用程序，一般不能写入程序和修改程序，其交互性比通用操作系统差。从独占性来看，每个用户通过自己的终端请求系统服务，互不干扰，好像独占一样。根据 IEEE 实时 UNIX 委会定义，实时操作系统应具备的特征有：

■ 异步事件响应

异步事件是指无一定时序关系随机发生的事件，如外部设备完成数据传输，实

时控制设备出现异常情况等。

- 切换时间和中断延迟时间确定
- 抢占式调度

为确保响应时间（任务进入就绪状态到被调度执行的时间），实时操作系统必须允许高优先级的任务一旦准备好运行，马上抢占低优先级任务的执行。

- 内存锁定

必须具有将程序或部分程序锁定在内存的能力，锁定在内存的程序减少了为获得该程序而访问磁盘的时间，从而保证了快速的响应时间。

- 优先级中断和调度

必须允许用户定义中断优先级和被调度任务的优先级，并制定如何中断服务。

- 连续文件

通常要求数据存储在连续文件上，以减少存取数据时的查找时间。

- 同步

提供同步协调共享数据使用和时间执行的手段。

实时系统的结构图如图 2-1

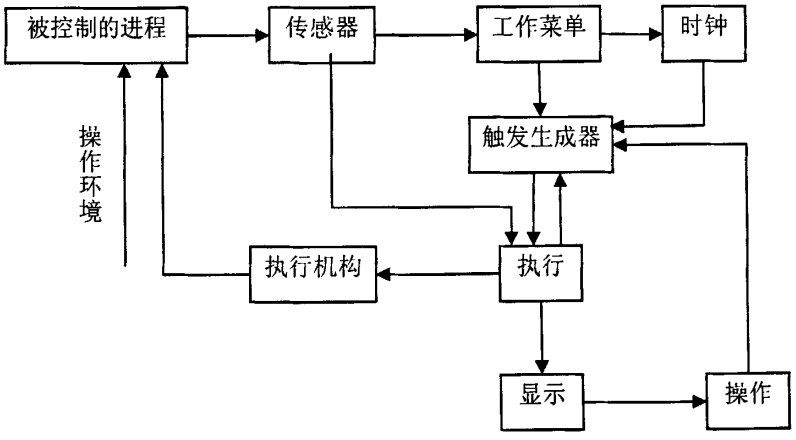



图 2-1 实时系统结构图

Figure 2-1Structure of real-time systems

## 2.3 嵌入式操作系统的特点

嵌入式操作系统具有区别于通用操作系统的一些重要特征，这些特征主要有以下几点。

（1） 体积小

设计者不断地在更小空间、更便宜的CPU中注入更强的计算能力，并希望将这些CPU集成于各种各样的项目中。一个小的嵌入式操作系统通常只使用几千字节的RAM和ROM内存。通用计算机系统一般要求功能强大，而在嵌入式系统中，在满足相应需求的情况下，则要求占用资源尽可能的少。

（2） 运行时间长

嵌入式操作系统在没有人工干预的情况下应该能运行几年，这意味着硬件和软件永远都不会出错。因此，系统最好没有机械部分，比如软驱和硬盘。因为机械部分更容易出问题，而且还会占用更多的空间，需要更多的能量，通信时间更长，而且驱动更复杂（比如机械部件的移动控制）。

（3） 故障重启

尽管已经将嵌入式系统设计的非常坚固可靠，但开发人员仍然需要预防系统出现故障的可能。因为此系统可能运行在一个无人的环境，无法手工进行复位操作，因此嵌入式系统通常应具有在出现故障时立即启动到一个安全状态的功能，特别是系统控制着一些关键设备时。

（4） 低功耗

嵌入式系统一般需要长时间工作，在电池功率一定的情况下，就需要系统尽可能小地消耗能量。大的耗电量还直接影响到硬件的费用，需要更大功率的电源。

（5） 价格便宜

嵌入式系统应该尽可能便宜。嵌入式系统通常是成千上万的大批量生产，哪怕是价格降低一点点，也会使库存产品销量大增。

（6） 动态加载

一些嵌入式系统在被启动以后在物理上是没法接触到的，但还是有许多嵌入式系统仍然可以进行远程访问。为了软件升级，应该可以支持动态链接，在开始启动时不存在的目标代码要能被上传到系统，在不需要停止系统运行的情况下，使得目标代码链接到运行的操作系统并运行。

## 2.4 嵌入式实时操作系统（RTOS）

### 2.4.1 RTOS的概念

关于嵌入式实时操作系统也没有一个精确的定义。简单地说，嵌入式实时操作

系统[19]是指能支持实时控制系统工作的一种嵌入式操作系统。它的首要任务就是要调度一切可以利用的资源完成实时控制任务，其次才着眼于提高计算机系统的使用效率。重要特点是要满足外部环境和和事件及对事件的限制和要求。

### 2.4.2 RTOS 的特征

实际的应用需求决定了 RTOS 的特性特征[24,25,26]。一般的 RTOS，其公共特征主要包括实时性、可靠性、可预测性、性能、紧凑性、可裁剪性。

（1） 实时性

任何一个电子系统都可以看成是一个激励－响应系统。每个特定的电子系统都有一个从激励输入到响应输出的时间，即激励－响应周期 T，它表现为系统的响应能力。如果系统的响应能力 T 能满足嵌入对象所规定的响应时间 Td 要求，即 $T \leqslant Td$，则这个系统便是实时的电子系统。

嵌入式实时操作系统也是一个激励－运行－响应的电子系统。但是，它与嵌入对象体系交互时，要满足事件交互过程的响应要求。一方面，嵌入式操作系统有较长的和不确定的激励－响应时间，导致系统实时能力的降低；另一方面，由于嵌入式实时操作系统是嵌入到对象体系中的一个电子系统，与对象系统密切相关，考虑到嵌入对象的多样性、复杂性，而且不同的对象系统会提出不同的响应时间要求，因此，在嵌入式实时操作系统的具体设计中，必须考虑系统中每一个任务的运行时，其激励－响应时间 $T_i$ 能否满足相应的嵌入对象规定的响应时间 $T_{id}$ 的要求，即要求每个任务都满足 $T_i \leqslant T_{id}$。这就是嵌入式系统的实时性问题。

只有对任意 i（i 是嵌入式实时操作系统的一个任务），有 $T_i \leqslant T_{id}$，那么整个嵌入式系统才是实时的。这种实时性体现为任务的快速调度、快速运行等实时性的设计内容上。实时能力强的嵌入式实时操作系统能更容易地实现整个应用系统的实时性。

（2） 可靠性

嵌入式实时操作系统必须是可靠的。因为在一般的应用过程中，嵌入式实时操作系统往往会长时间运行在无人干预的环境下，如果出现异常或错误的情况，就会造成灾难或严重的后果。当然，不是所有的嵌入式系统都要求很高的可靠性，针对应用需求会有不同的可靠性要求。虽然每个嵌入式实时操作系统的可靠性可以要求不同，但一个嵌入式实时操作系统需要达到一些最基本的可靠性要求，必须保证它是一个可使用的系统，能够连续提供服务并且不能失败。

（3） 可预测性

满足时间要求是一个实时系统保证合理运行的关键，因此 RTOS 的任务运行时间必须是可预测的，以保证任务处理能够在已知的时间范围内完成。

（4） 性能

“性能”指标往往是要求一个嵌入式系统能够把任务执行的足够快。一个嵌入式系统需要满足很多时限要求，首先想到的就是 CPU 一定要很快，必须充分满足系统的时限要求，其次是一些硬件也必须很快。剩下的影响系统处理能力和整体性能的就是软件了，即操作系统和应用软件。其中操作系统会产生关键影响。

（5） 紧凑性

应用设计约束和费用约束决定一个嵌入式系统的紧凑性要求。这些要求限制了系统内存，进而限制了应用和操作系统的尺寸。

（6） 可裁剪性

由于 RTOS 可以广泛用于各种嵌入式系统，所以必须能够进行裁剪以满足应用的特定要求。依据所要求的功能，RTOS 应当具有增加或减少模块化部件的能力，包括文件系统和协议栈。

## 2.5 RTOS 的组成

RTOS 一般由内核、嵌入式 TCP/IP 网络系统和嵌入式文件系统等部分组成。

### 2.5.1 RTOS 内核

内核是 RTOS 的基础，也是必要部分。它提供任务管理、内存管理、通信管理、同步与互斥机制、中断管理、时间管理及任务扩展等功能。内核还提供特定的应用编程接口，但是目前还没有统一的标准。

（1） 任务管理

任务管理是内核的核心部分，具有任务调度、创建任务、删除任务、挂起任务、解挂任务和设置任务优先级等功能。通用计算机的操作系统追求最大的吞吐率。为了达到最佳整体性能，其调度原则是公平的。而嵌入式实时操作系统多采用基于静态优先级的调度策略，任务的优先级是在运行前通过某种策略分配好的，一旦有优先级更高的任务就绪，马上进行调度。

（2） 内存管理

相对于通用操作系统来说，嵌入式实时操作系统的内存管理比较简单，通常不采用虚拟存储管理，而是采用静态内存分配和动态内存分配相结合的管理方式。有

些内核利用 MMU 机制提供内存保护功能。

（3） 通信、同步和互斥机制

这些机制提供任务间以及任务与中断处理程序间的通信、同步和互斥功能，一般包括信号量、消息、事件、管道、异步信号和共享内存等功能。

（4） 中断管理

中断管理一般具有以下一些功能：

- 安装中断服务程序
- 中断发生时，对中断现场进行保护，并转到相应的中断服务程序上执行
- 中断退出前，对中断现场进行恢复
- 中断栈切换
- 中断退出时的任务调度

（5） 时间管理

时间管理要提供高精度且应用可设置的系统时钟，该时钟是嵌入式系统的时基，实时系统的时钟精度一般要求在 10ms 以下，最好是微秒级的；时间管理还要提供日历时间，负责与时间相关的任务管理工作。

### 2.5.2 嵌入式 TCP/IP 网络系统

TCP/IP 协议已经广泛地应用在 RTOS 中。嵌入式 TCP/IP 网络系统需要提供符合 TCP/IP 协议标准的协议栈，提供 Socket 编程接口，如图 2－2 所示。

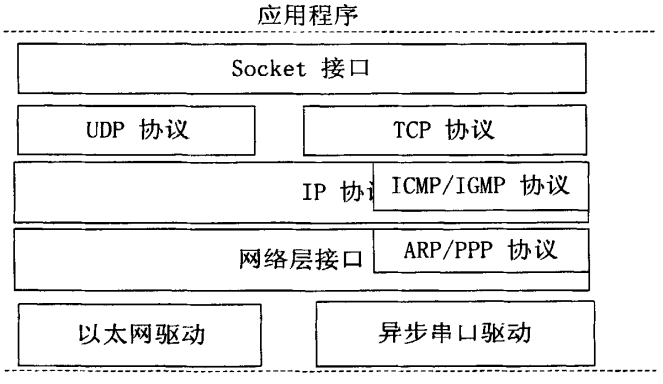



图2-2 嵌入式系统TCP/IP

Figure 2-2 TCP/IP of Embedded

嵌入式实时系统的 TCP/IP 网络系统具有以下主要特点：

- 可裁剪性

- 零复制
- 可扩展

## 2.6 嵌入式实时操作系统 uC/OS-II 简介

uC/OS-II 是一个免费的、源码公开的嵌入式实时操作系统，其内核提供了实时系统所需要的一些基本功能，其中包含全部功能的核心代码占用 8.3K，而且由于 uC/OS-II 是可裁剪的，所以在用户系统中实际的代码最少可达 2.7K，可谓短小精悍。uC/OS-II 不仅使得用户得到廉价的解决方案，而且由于 uC/OS-II 的源代码是开放的，使得用户可以针对自己的硬件优化代码，从而获得更好的性能[20,21]。

### 2.6.1 uC/OS-II 系统的特点

（1） 有源代码

uC/OS-II 源代码是开放的，用户可以登录 uC/OS-II 的网站下载对不同微处理器的移植代码。这极大地方便了嵌入式实时系统 uC/OS-II 的开发、降低了开发成本。

（2） 可移植

在 uC/OS-II 的源代码中，除了与微处理器硬件相关的部分代码是使用汇编语言编写的，其绝大部分是使用移植性很强的 ANSI C 编写的，并且把用汇编语言编写的部分已经压缩到最低限度，以使 uC/OS-II 更方便地移植到其他微处理器上使用。

（3） 可固化

uC/OS-II 是为嵌入式应用而设计的操作系统，只要具备符合的软/硬件工具，就可以将 uC/OS-II 嵌入到产品中去，从而成为产品的一部分。

（4） 可裁剪

uC/OS-II 可根据实际用户的应用需要，使用条件编译来完成对操作系统的裁剪，这样就可以减少 uC/OS-II 对代码空间和数据空间的占用。

（5） 可剥夺

uC/OS-II 是完全可剥夺型的实时内核，运行就绪条件下优先级最高的任务。

（6） 多任务

uC/OS-II 可管理 64 个任务。一般情况下，建议用户保留 8 个任务给 uC/OS-II。系统给每个任务的优先级必须不同，这意味着 uC/OS-II 不支持时间片轮转调度。

（7） 可确定性

绝大多数 uC/OS-II 的函数调用和服务的执行时间具有确定性，也就是说，

uC/OS-II 系统服务时间与用户应用程序任务数目多少无关，在任何时候用户都能知道 uC/OS-II 的函数调用与服务的执行时间。

（8） 任务栈

uC/OS-II 的每个任务都有自己单独的栈和栈空间。为了满足应用程序对 RAM 的需求，使用 uC/OS-II 的栈空间校验函数可以确定每个任务到底需要多少栈空间。

（9） 系统服务

uC/OS-II 提供了很多系统服务，如信号量、互斥信号量、消息邮箱、事件标志、数据队列、块大小固定的内存申请与释放及时间管理函数等。

（10） 中断管理

中断可使正在执行的任务暂时挂起，如果优先级更高的任务被中断唤醒，则更高优先级的任务在中断嵌套全部退出后立即执行。中断嵌套层数可达 255 层。

（11） 稳定性与可靠性

uC/OS-II 自 1992 年以来已经有数百个商业应用。uC/OS-II 与 uC/OS 内核是一样的，只是提供了更多的功能。另外，2000 年 7 月，uC/OS-II 在一个航空项目中得到了美国联邦航空管理局对商用飞机的符合 TRCA DO-178B 标准的认证。可以说，uC/OS-II 的每一种功能、每一个函数及每一行代码都经过了考验和测试。

### 2.6.2 uC/OS-II 系统的内核结构

与其他操作系统不同，uC/OS-II 其实只是一个内核，提供任务管理、任务间的通信与同步、中断管理、时间管理和内存管理等基本功能。

（1） 任务

在 uC/OS-II 中，一个任务通常是一个无限循环。一个任务看起来像其他 C 语言函数一样，但任务是绝不能返回的，故返回参数必须定义成 void，例如：

```
Void Your Task( )
    {
        for(; ;){
                /*用户代码*/
                /*调用 uC/OS-II 的某种服务：*/
                /*用户代码*/}
    }
```

（2） 任务调度

uC/OS-II 可以管理多达 64 个任务，其优先级可以从 0 到 OS_LOWEST_PRIO，优先级号越低，其任务的优先级越高。目前的 uC/OS-II 有两个任务已经被系统占用

了，而且保留了优先级 0、1、2、3 和 OS_LOWEST_PRIO-3、OS_LOWEST_PRIO-2、OS_LOWEST_PRIO-1 以及 OS_LOWEST_PRIO 这 8 个任务已备将来使用。

为了使 uC/OS-II 能管理用户任务，用户必须在建立一个任务的时候，将任务的起始地址与其他参数一起传给函数 OSTaskCreat( )或者 OSTaskCreatEXT( )。图 2-3 是 uC/OS-II 控制下的任务状态转换图，在任一时刻任务的状态一定是这 5 种状态之一。

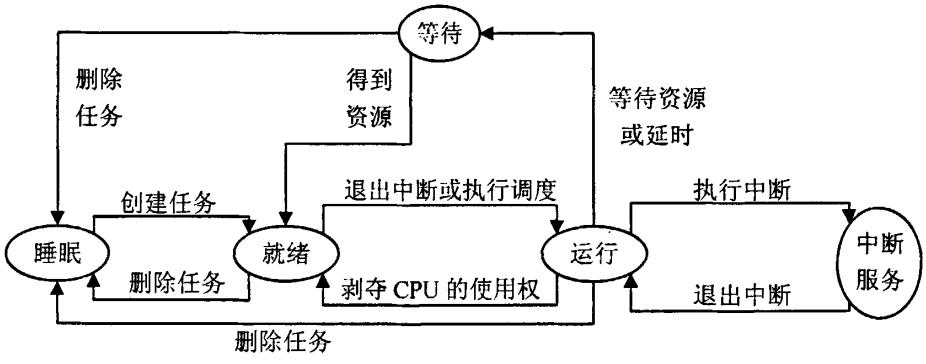



图 2-3 u C/OS-II 的任务状态

Figure2-3 Task states of uC/OS-II

挂起态又可称为休眠态，指任务驻留在程序空间中，还没有交给 uC/OS-II 管理。任务一旦建立就进入就绪状态准备运行。其中，任务的建立可以是在多任务开始运行之前，也可以是动态地被一个运行着的任务建立。正在运行的任务期待某一事件发生时则要进入等待态。在系统没有关中断的情况下，正在运行的任务是可以被中断的。当所有任务都处于等待事件发生或等待延迟事件结束的状态时，uC/OS-II 执行空闲任务。

uC/OS-II 总是运行进入就绪态的任务中优先级最高的那个任务，所以确定那个任务优先级最高，那个任务将要运行的工作是由调度器来完成的。uC/OS-II 任务调度所花的时间是常数，与应用程序中建立的任务数无关。

（3） 主要模块

■ 内存管理

uC/OS-II 把连续的大块内存按分区来进行管理。每个分区都包含若干个存储容量大小相同的内存块，但不同分区之间的内存块容量是可以不同的。在需要动态分配内存时，可以选择一个适当的分区，按块来分配内存。在释放时，将该块放回到它以前所属的分区，这样就有效地解决了内存碎片问题。

■ 时间管理

与大部分内核一样，uC/OS-II 要求提供定时中断，以实现延时与超时控制等功能。这个定时中断也可以被叫做时钟节拍。

（4） 文件系统

uC/OS-II 操作系统的文件系统如图 2-4 所示，其核心可以分为以下三层：

- 应用软件层，指的是基于 uC/OS-II 的应用程序代码。
- 内核的核心代码层，主要包括 8 个源文件代码。其主要实现的功能是核心管理、事件管理、存储管理、消息队列管理、定时管理、信号量处理、消息管理和任务调度等，这部分代码与处理器无关。
- 系统设置与移植层，其主要功能是用来配置事件控制块的数量以及是否包含消息管理的代码。与处理器相关的代码与具体应用和处理器相关，在移植过程中这是需要关注的文件。

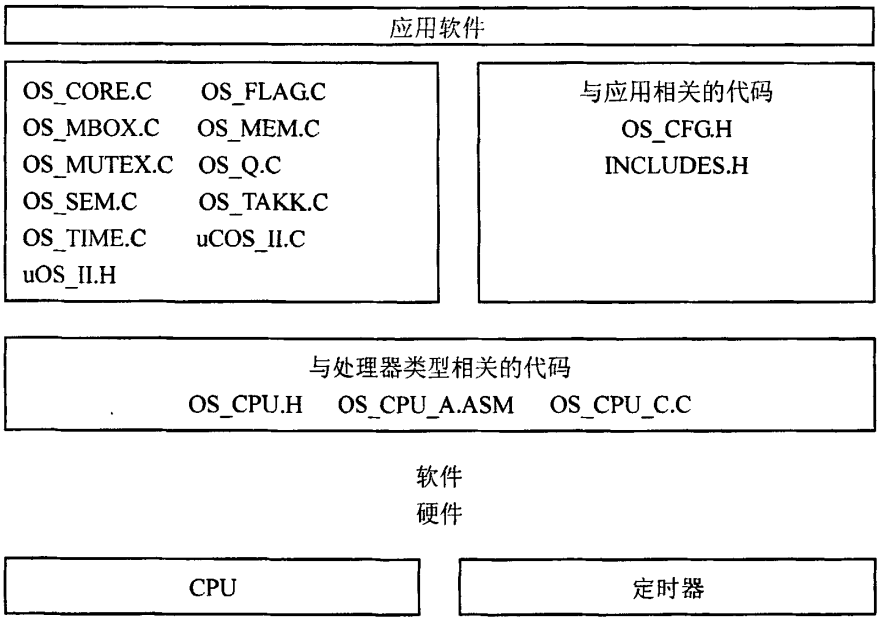



图 2-4 uC/OS-II 的结构图

Figure 2-4 Structure of uC/OS-II

# 第三章 OSEK/VDX 标准

随着嵌入式应用复杂度的提高，嵌入式软件的规模也发生了指数型增长。软件的时限从某种意义上说决定了产品的功能，已成为新产品成功与否的关键因素。在汽车电子业，各厂商生产的电子产品其软件特点是五花八门，使得成本和产品性能受到极大的影响。

针对这种情况一些地区和国家的若干行业协会纷纷制定了嵌入式系统标准，在汽车领域，欧共体汽车产业联盟在 1993 年规定以 OSKE（OSEK 的名称来自于德文“车内电子设备的开放系统接口”）标准作为开放汽车嵌入式系统的公用平台和应用编程接口。并得到世界各大生产厂商的支持，在 1994 年法国的标致和雷诺汽车制造商也加入了这一体系，并将法国汽车工业使用的汽车分布式系统 VDX 业纳入到这一体系中来。世界各大汽车生产厂商于 1995 年对 OSEK/VDX 达成了共识，正式将其规定为汽车嵌入式系统开发的统一标准。该规范包括四个部分：操作系统规范、通信规范、网络管理规范和实现语言（OIL）。

OSEK/VDX 规范从 1995 年提出至今，国际上已经有很多研究机构深入研究了该标准，如美国的密西根大学[22,23,24,25,26]、美国的卡内基梅隆大学[27,28,29]、德国的卡尔斯鲁厄大学[30,31,32]等。我国已经将符合 OSEK/VDX 规范的汽车电子嵌入式软件平台的开发及其相关技术的研究作为“十五”期间 863 重大研究项目。目前，清华大学[33,34]、浙江大学[35,36]、上海交通大学[37]、同济大学[38]、北京航空航天大学[39,40,41,42]、北京理工大学等高校均展开了对 OSEK/VDX 规范的学习与研究。

## 3.1 操作系统规范

OSEK 操作系统规范[43,44]描述了一个统一的环境，这个环境为建立在汽车控制单元基础上的应用软件提供有效的支持。OSEK 操作系统是单处理器操作系统，这意味着它是一个分布式的嵌入控制单元。

汽车应用软件的特点是它的时间要求特别苛刻。因此 OSEK 操作系统提供了支持事件驱动控制的必要函数。

规范中规定的操作系统服务组成了一个基础的、可以与各个制造厂家制定的软件集成在一起的服务的集合。为了能和一个由资源消耗最小性决定的独立控制单元的特性相适应，它的焦点不是实现 100%应用模块的兼容性，而是实现可移植性。

为了使操作系统适合低端的微控制器和复杂的控制单元，高度模块化和灵活的

可配性，规范提出了“符合类“的概念。对于一个时间要求苛刻的应用程序，生成动态操作系统是不可行的，取而代之的是生成一个静态操作系统，从而在操作系统中可以避免很大范围的错误请求，并且不影响操作系统的整体速度。另一方面，系统版本也定义了扩展的错误查询。即它定义了统一的系统错误体系。

应用程序和操作系统之间的接口由操作系统的服务定义，这些接口对于各种各样的处理器的实现是统一的。系统服务被规范为和 ISO/ANSI-C 一样的语法，但系统的实现语言未被规范。

不同的符合类、各种调度机制和配置特性使得 OSEK 操作系统能兼容很大范围的应用程序和硬件。OSEK 操作系统被设计为仅需要很少量的硬件资源(RAM、ROM、CPU 时间)，甚至能运行在 8 比特的微控制器上。

OSEK 操作系统提供了两级错误检测：开发阶段的扩展状态和产品阶段的标准状态。

扩展状态允许在调用操作系统服务时使用增强的真实性检测，和标准状态相比，额外的错误检测需要更多的执行时间和内存空间，但在测试阶段很多错误可以被发现。测试阶段的错误被消除后，系统可以在标准状态下重新编译。

OSEK 的一个目标就是支持应用软件的可移植性和可重用性。因此，应用软件和操作系统之间的接口由标准的系统服务函数来定义，使用标准的系统服务可以减少应用软件的维护和开发费用。可移植性意味着把一个应用软件模块从一个电控单元上转到另一个电控单元时，应用程序不需要很大的改动。标准接口（服务调用，类型定义和常量）给操作系统提供源码级的支持，但交换的目标代码没有被 OSEK 规范所定义。应用软件依赖于操作系统，在 OSEK 规范中没有规定输入/输出接口，这部分应用软件模块可以有很多接口。如果应用工作在特定的微控制器模块上，也有一些软件模块在一个系统中描述了一个完善的函数，至少对这个特定的硬件是如此。

为了应用软件有更好的可移植性，OSEK 规范为标准化配置信息定义了一种语言，这种语言“OIL“（OSEK 实现语言）支持简单的对 OSEK 的所有具体目标的描述，例如：任务、报警等。

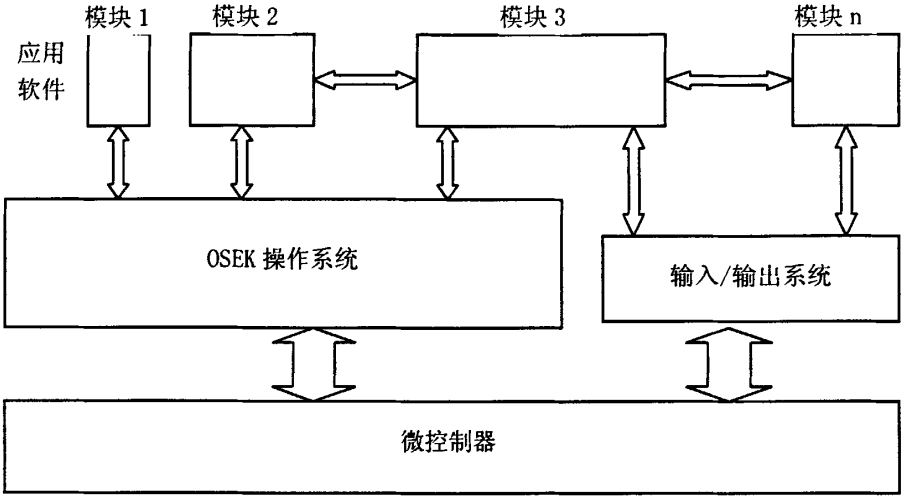



图 3-1 应用软件与 ECU 的接口

Figure 3-1 Software interfaces inside ECU

应用软件从一个电控单元向另一个电控单元移植的时必须考虑软件的开发过程、开发环境、电控单元的硬件体系。例如：

- 软件开发指导方针
- 文件管理系统
- 编译器的数据分配和堆栈使用
- 电控单元的内存体系
- 电控单元的定时器行为
- 不同的微处理器的特殊接口，例如：端口、A/D 转换、串行通信和看门狗定时器。
- API 调用的放置

这意味着 OSEK 规范不能完全描述 OSEK 的实现，具体的实现将提供更详细的文档。

一个 OSEK 操作系统的特殊需求起因于汽车控制单元的软件开发中的应用程序上下文。下面的特性描述了这种请求，诸如可靠性，实时性，价格等。

- OSEK 操作系统是可配置的、静态的。用户静态的指定任务数，资源和服务请求。
- OSEK 操作系统规范支持程序在 ROM 中的执行，即代码可以从只读存储器中执行。
- OSEK 操作系统支持应用程序任务的可移植性。

- OSEK操作系统规范提供了使操作系统执行的可预测的、文档化的行为，它是满足汽车实时性的必要条件。
- OSEK操作系统规范允许性能参数的可预测性执行。

为了满足不同应用软件的需要，操作系统规范定义了一个独特的新概念——符合类（Conformance class：CC）。根据任务的类型、多重激活性和每个优先级所允许的任务数，OSEK操作系统规范一共定义了四种符合类：

- 基本符合类1：只包含基本任务，每个任务只能激活一次，每个任务都有一个优先级且不同任务的优先级不同。
- 基本符合类2：包括基本符合类1的要求，且允许同一个任务的多次激活且允许多个任务具有相同的优先级。
- 扩展符合类3：包括基本符合类1的要求，且允许有扩展任务。
- 扩展符合类4：包括扩展符合类2的要求，且允许同一任务的多次激活及允许多个任务具有相同的优先级。

### 3.1.1 任务管理

任务管理主要的功能是激活和中止任务；任务状态管理和任务的切换管理。操作系统规范中的任务主要分为两类：基本任务和扩展任务。扩展任务因为可以拥有事件而增加了等待状态外，其他状态模型和基本任务的状态相同。

基本任务没有等待状态，基本任务的同步点仅由任务开始和结束两处同步点组成。基本任务的优点就是它有合适的运行上下文（RAM）请求。扩展任务的一个优点就是无论哪个同步请求是活跃的，它都可以在单个任务中处理连贯的工作。当要处理的消息缺少相应的资源时，扩展任务将进入等待状态。在没有接收到与之对应的事件信号或需要的数据或事件没有更新之前，它将一直处于等待状态。扩展任务比基本任务还有更多的同步点。

任务有以下几种状态：

- 挂起：在挂起状态，任务是不起作用的，并且可以被激活。。
- 运行：在运行态，CPU被分配给任务，因此它的指令被执行。在一个时间点仅有一个任务处于此状态。而其他的状态可以同时有几个任务。
- 就绪：所有任务进入运行态的首要条件，任务仅等待处理器的分配。调度程序决定下一个要运行的就绪任务。
- 等待：一个任务不能一直执行，因为它有可能至少要等待一个事件的发生。

任务状态的转换如图 3-2 所示

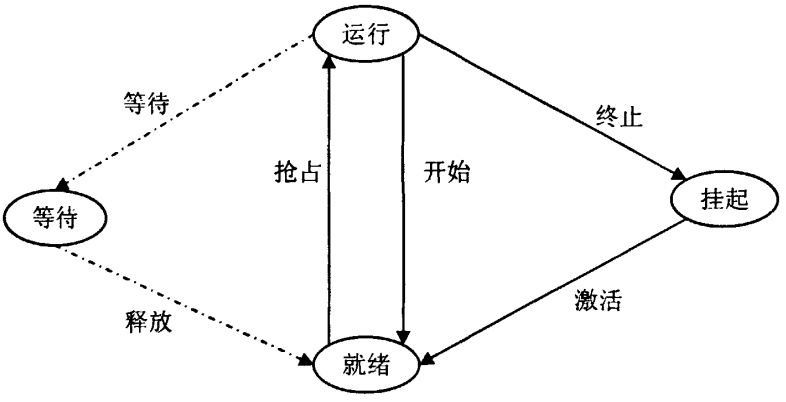



图 3-2 任务状态转换
Figure 3-2 Extended task state model

OSEK 操作系统内部通过调度程序激活相应任务的过程称为调度。按照系统实现时的调度策略，只要存在任务切换的可能，那么调度程序就会被激活。调度器可以被当作一种资源，它可以被任务占用或释放。调度器以任务的优先级为基础来决定哪一个任务接下来将从就绪态进入运行态，一个任务的最低优先级被定义为 0，因此越大的数字代表的优先级越高，为了提高效率，OSEK 操作系统不支持动态优先级管理；任务的优先级被静态的定义，因此，用户在任务的执行过程中不能改变优先级。然而，在特殊情况下，操作系统可以提高一个任务的优先级，相同优先级上的任务启动顺序由它们的激活顺序决定，通常是按照先进先出队列来进行调度，处于等待状态的扩展任务不会阻塞其随后相同优先级任务的启动。图 3-3 给出了为每一个优先级使用调度器的实现示例。不同优先级的几个任务处于就绪态，比如：3 个优先级为 3 的任务，一个优先级为 2 的任务，一个优先级为 1 的任务和两个优先级为 0 的任务。那些已经等待了很长时间的任务，依赖于它的请求顺序。处理器仅任务进行调度处理，调度器选择下一个将要执行的任务（第一个队列，优先级 3），优先级为 2 的任务只有在所有高优先级的任务执行完之后且它处于就绪态后才开始执行。

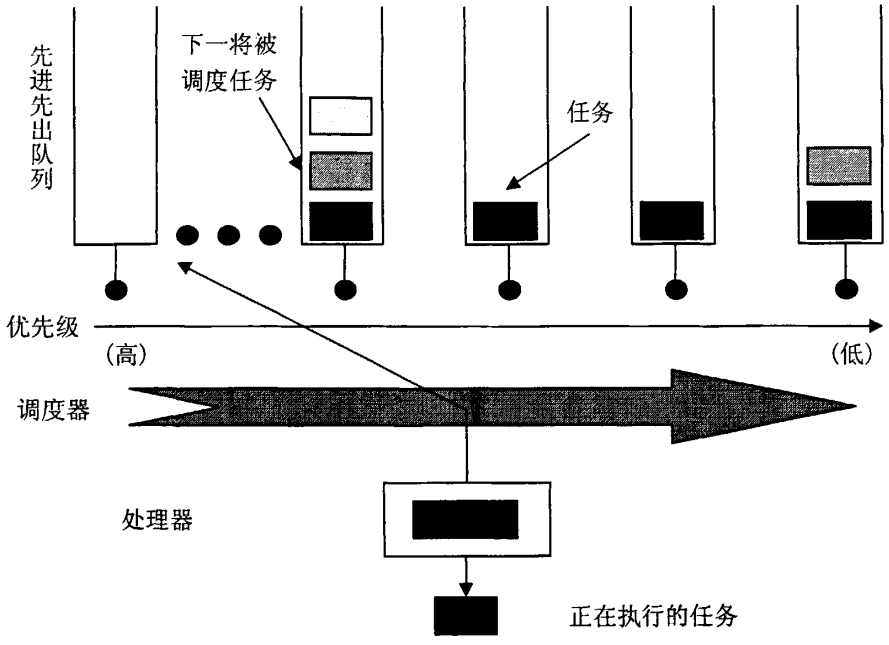



图 3-3 调度器运行过程实例
Figure 3-3 Scheduler: order of events

下一个被处理的任务必须满足下面几个基本的条件：

- 调度器搜索所有处于就绪/运行态的任务。
- 从就绪/运行任务队列中，调度器决定最高优先级的任务队列。在最高优先级的就绪/运行任务队列中，调度器选择最老的任务。

根据任务的可抢断性可以将调度策略分为三类：

（1） 全抢占调度

全抢占调度意味着目前正在运行的任务在任何指令时间都有可能被其他优先级更高的任务抢占。全抢占调度能使正在运行的任务进入就绪态，一个比当前任务的优先级更高的任务得以运行，被打断任务的上下文被保存，这保证它在随后的调度中可以从被打断处继续运行。

全抢占调度的等待时间独立于最低优先级的运行时间，这个时间与系统确定的约束和保存上下文请求内存(RAM)空间有关。两个任务之间的同步增加了这些特性的复杂性，就像每个任务理论上可以在任何地方被重新调度一样，但实际情况往往不是如此。如图 3-4，较低优先级的任务 T2 不会使较高优先级的任务 T1 延迟。

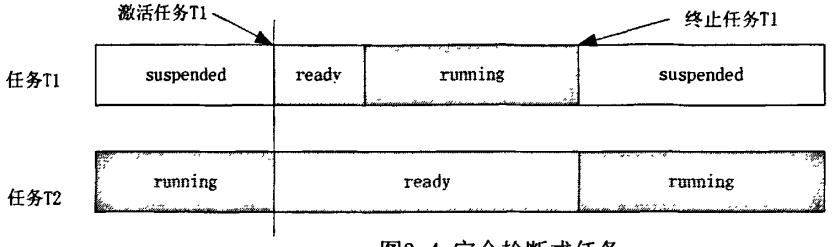



图3-4 完全抢断式任务

Figure 3-4 Full preemptive scheduling

总之，重新调度发生在下面的几种情况：

- 成功的结束一个任务。
- 成功的结束了一个明确规定了后续将要执行哪个任务的任务。
- 在任务级激活了一个任务（例如：系统服务 ActiveTask，消息通知机制，报警期满）。
- 明确的等待调用使任务进入等待状态时。
- 给一个任务级的等待任务设置一个事件。
- 在任务级释放资源。
- 从中断级返回到任务级。

中断服务程序运行期间不能有任何任务调度发生。应用程序使用“全抢占”的调度策略不需要系统服务 Schedule，但是其他的调度策略可以利用这个系统服务。为了增强应用程序的可移植性，可以使用不同的调度策略，用户可以通过系统服务 Schedule，在他认为可以正确分配 CPU 的地方强行执行一个调度。

（2） 非抢占调度

非抢占的任务调度策略即一个任务一旦获得 CPU 资源而得以运行就不能被别的任务抢占，直到运行结束或等待资源而进入等待状态，而自动放弃 CPU。

非抢占调度策略对任务可能的请求时机施加了特殊的约束，特别是在低优先级任务的临界区，它推迟了高优先级任务执行。在图 3-5，低优先级的任务 T2 延迟高优先级的任务 T1 直到下一个重新调度点。

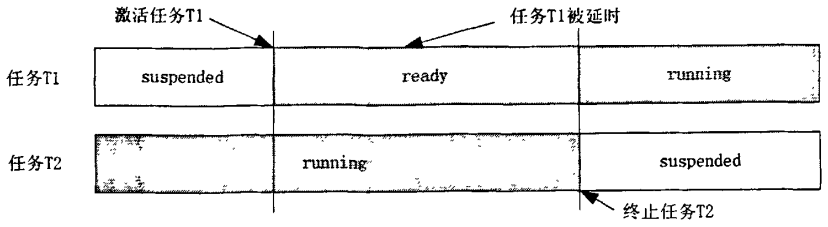



图3-5 非抢断式任务

Figure 3-5 Non preemptive scheduling

在一个非抢占任务调度策略的情况下，重新调度将会在下面的情况下发生：

- 成功的终止了一个任务。
- 成功的结束了一个明确规定了后续要执行任务的任务。
- 明确的调用调度器。
- 产生了一个进入等待状态的变迁。

（3） 混合抢占调度

如果可抢占的和不可抢占的任务调度策略混合在一个系统中，形成的这种调度策略称为“混合抢占”调度。在这种调度策略下，任务的运行依赖于任务的抢占属性，如果运行任务是非抢占的，非抢占调度被执行，如果允许的任务是可抢占的，则抢占调度被执行。

不可抢占的任务调度策略在一个全抢占的操作系统中有很重要的意义：

- 如果任务的执行时间和任务的切换时间有相同的数量级。
- 如果RAM被用来为上下文切换提供空间。
- 如果任务不会被抢占。

很多应用程序仅由很少的几个任务，但这些任务有着较长执行时间，这时全抢占的操作系统把这些执行时间较长的任务设置成不可抢占，对系统来说是非常有利的。对于那些执行时间很短的任务，采用非抢占调度会更有效。

### 3.1.2 事件机制

事件是操作系统管理的对象，它们不是独立的对象，它们被分配给扩展任务。每一个扩展任务有一定数量的事件。这个任务称为事件的所有者。一个事件被它的所有者和它的名字（或者掩码）所标识。当激活一个扩展任务时，这些事件被操作系统识别，事件可以给拥有它的任务发送二进制信息。事件的含义通过应用来定义，例如：一个时间到期的定时器发信号、一个资源可用、接收到一个消息等等。

任务对于事件的操作，任何没有挂起的任务可以设置任何事件；只有事件的拥有者可以清除它所拥有的事件，并且可以等待接收它所拥有的事件。扩展任务从等待状态进入就绪态，事件是必须的，操作系统为那些等待事件发生的任务提供设置、清除和查询事件的服务。任何任务或者 2 类中断可以为一个没有被挂起的扩展任务设置事件，可以通过这个事件通知扩展任务改变其状态。在任何情况下，一个事件的接收者都是一个扩展任务，因此，中断服务程序或者基本任务不可能等待一个事件。一个事件仅能被拥有它的任务清除；基本任务不能使用操作系统服务来清除事件。如果一个任务等待的事件发生，则扩展任务由等待状态进入就绪态。如果一个运行的试图等待一个已经发生的事件，则这个扩展任务继续保持运行态。

图 3-6 是在全抢占调度的情况下，被设置一个事件的扩展任务的同步解释。此处扩展任务 T1 有较高的优先级。

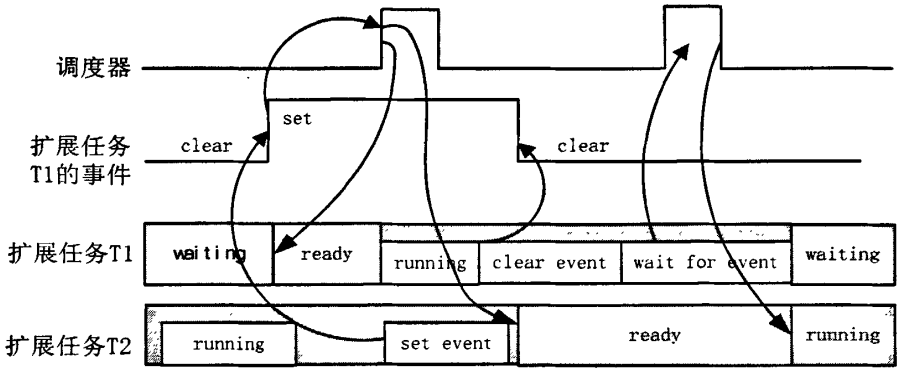



图3-6 可抢占扩展任务的同步

Figure 3-6 Synchronisation of preemptable

### 3.1.3 资源管理

资源管理被用来协调具有不同优先级的任务对共享资源的并发访问，例如：实体（调度器）、程序队列、内存和硬件的管理。对于所有的符合类，资源管理是必须的，资源管理可以被随意的扩展到任务和中断服务程序，资源管理确保：

- 两个任务不能在同一时间占据相同的资源。
- 优先级反转不能发生。
- 使用资源时可以防止死锁的发生。
- 对资源的访问不会导致任务进入等待状态。

同步机制中的典型问题是，优先级反转的问题。这意味着一个低优先级的任务

延迟了一个高优先级任务的执行。OSEK 规范中通过优先级天花板协议来避免优先级反转问题。举例说明：如果两个任务访问一个信号量（在一个全抢占的系统中，任务 T1 有最高的优先级）的通常顺序问题，任务 T4 为低优先级，占用信号量 S1，T1 抢占 T4 并请求相同的信号量，由于信号量已经被占用，T1 进入等待状态；现在低优先级的任务 T4 被优先级比 T4 高但比 T1 低的任务所中断并抢占，T1 仅能在所有低优先级的任务被终止才能获得 CPU 而运行，尽管 T2 和 T3 没有使用信号量 S1，但它们延迟了 T1 的执行。

同步机制中的另一个典型问题是：使用信号量时的死锁问题。死锁是任务由于无限期的等待被任务互锁的资源而不能执行的情况。下面是死锁的实例：任务 T1 占用信号量 S1 并且后来不能继续运行，例如，由于它等待一个事件；因此，低优先级的任务 T2 进入运行状态，它占用信号量 S2。如果 T1 再次就绪并且试图占用信号量 S2，它再次进入等待状态。如果现在 T2 试图占用信号量 S1，这样就导致两个任务都不能得到自己需要的信号量而互相无限期的等待下去，从而导致了致死锁的产生。

为了避免优先级反转和死锁的问题，OSEK 操作系统规范中通过以下方式来解决这个问题：

- 在系统生成时，为每个资源定义一个优先级，此优先级是静态分配的。天花板优先级至少被设为与访问资源或访问和该资源相连的所有任务里面优先级最高的那个任务的优先级。天花板优先级将比所有的不访问该资源的最低优先级的任务的优先级还低。比所有访问该资源的任务里面最高优先级还高的任务可以访问该资源。
- 如果任务获得了资源，并且它当前的优先级比资源的天花板优先级还低，则这个任务的优先级会被提升到资源的天花板优先级。
- 如果任务释放了资源，这个任务的优先级还原到为它静态定义的优先级。

优先级天花板导致优先级低于或等于资源优先级的任务在一段时间内被延迟，这个延迟由被所有较低优先级任务中，需要使用该资源的，运行时间最长的任务所限制。那些可能占用相同资源的任务不能进入运行态，由于它们低于或等于运行任务的优先级。如果资源被占用它的任务释放，那么其它的可能需要获取该资源的任务可以进入运行态。对于允许抢占的任务来说，这是一个重新调度点。图 3-7 中的举例说明了优先级天花板机制，其中任务 T0 有最高的优先级，任务 T3 的优先级最低，任务 T1 和 T3 想访问相同的资源；系统明确的表明了优先级的反转，高优先级的任务 T1 等待的时间比被 T3 占据的资源的最大持续时间短。

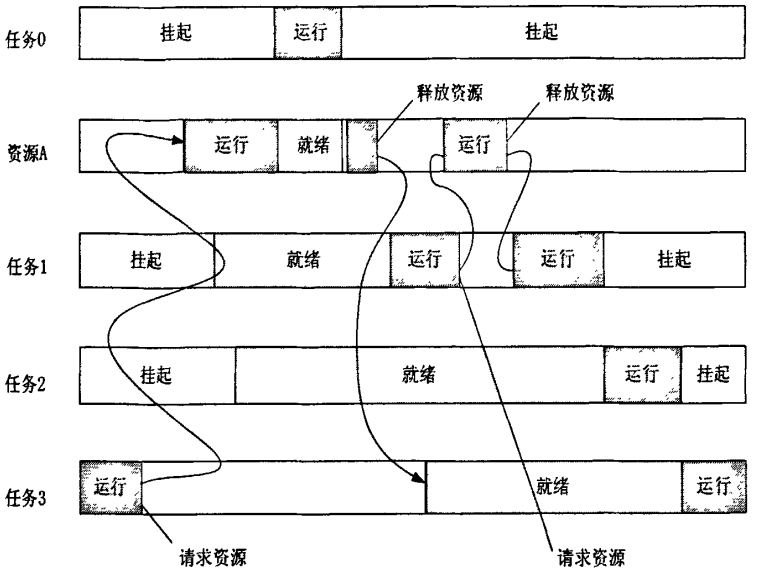



图3-7 优先级天花板协议下的资源分配

Figure 3-7 Resource assignment with priority ceiling between preemptable tasks

preemptable extended tasks

### 3.1.4 报警

OSEK 操作系统可以为处理周期事件提供服务，这些事件可能是定期提供定时器中断，或者来自外围设备产生的中断，或者定期申请规定的触发器等等。OSEK 操作系统为处理这种事件提供了一个两用的概念，报警。重发的事件（资源）被系统中计数器所登记，基于计数器，OSEK 操作系统提供报警机制给应用软件。

报警可以被设为单个报警或周期报警。另外，操作系统提供撤消报警或得到当前报警状态的服务，多个报警可以被赋予一个计数器。在系统生成时一个报警被静态的分配给一个计数器，一个任务或一个报警回调程序，这取决于系统的配置。当报警到期时，报警回调程序被调用、或者某个任务被激活、或者为某个任务设置一个事件等等。报警回调程序执行时，2 类中断禁用。在一个报警回调程序中允许的系统服务，就像普通任务的激活和事件设置一样；当一个报警到期时，任务激活和事件设置有着相同的优先级。计数器和报警被静态定义，分配给计数器的报警也被静态定义。

### 3.1.5 中断

在 OSEK 规范中，中断处理函数（中断处理程序：ISR）被细分成两类 ISR：

1 类 ISR：在这类 ISR 中不能调用要操作系统服务，当 ISR 结束后，被中断的程

序从中断处继续执行。中断对任务管理没有影响，且开销最小。

2 类 ISR：OSEK 操作系统提供了一个 ISR 框架，给一个特定的用户程序提供运行时的环境。在系统生成期间用户程序就被分配给中断。

## 3.2 通信规范

OSEK COM 是一个汽车控制单元应用软件中一个统一通信环境[45]。它通过定义通用软件的通信接口，内部通信（在一个 ecu 中通信）和外部通信（网络节点间通信）增强了应用软件便捷性，内外通信独立于使用的通信协议。这个标准描述了在一个 ECU 中的活动，它假设 OSEK COM 与符合 OSEK OS 标准的操作系统一起使用。

### 3.2.1 概念

通信的概念模型，如图 3-8 表明了通信在 OSEK 结构中的位置及其与他单元间的交互情况。此外，它还部分或全部地涉及到以下这些层：

■ 交互层：它提供了为消息提供传输服务的 API。它使用低层提供的服务来处理外部消息，因此内部消息都被它处理。

■ 网络层：依据系统所使用的通信协议，网络层处理消息分割、重组和通知。它提供了使能使系统能和性能及功能不是同级的其他系统进行通信。OSEK COM 没有规定网络层，它只是定义了网络层支持交互层所有特性的最小需求。

■ 数据链路层：它为上层提供了网络中单个数据包（帧）不通知的传送服务，而且，它还提供了 NM 服务。OSEK COM 没有规定数据链路层，它只是指定了数据链路层支持交互层的最小需求。

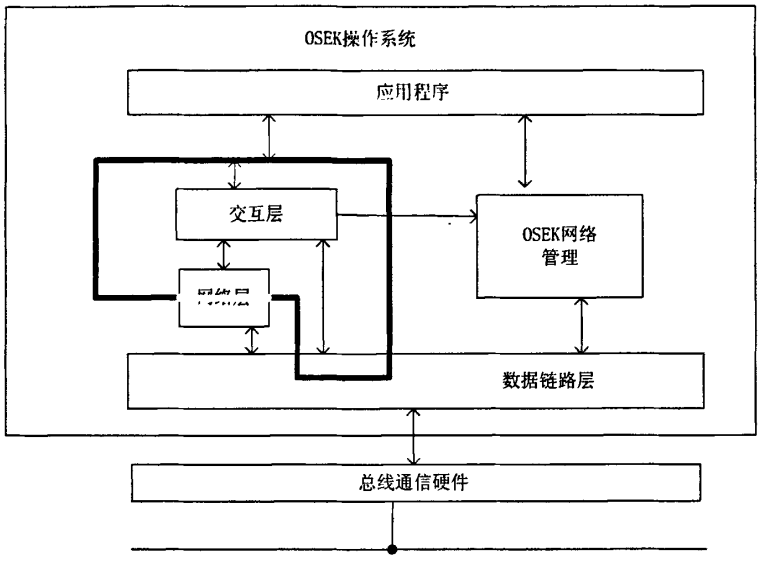



图3-8 通信概念模型

Figure 3-8 OSEK COM's layer model

### 3.2.2 消息

消息是内存空间中一段长度可变的缓冲区。其内容可以是实际的数据、数据块的指针或空。消息机制在任务和任务之间、任务和中断服务程序之间提供消息传送机制，实现带数据的同步。消息可分为队列消息和非队列消息。

队列消息是一个 FIFO（先进先出）队列。当队列为空，则不提供任何消息数据给应用程序，当队列不为空，应用程序可以接收到消息。如果新消息数据到来，队列没有满，这个新消息存储在队列中。如果新消息数据到来，队列已满，这条消息会丢失。

非队列消息不是使用 FIFO 机制。在消息数据接收过程中，应用程序不能使用该消息，但是，消息一旦被消息控制块接收，便可以被应用程序读取多次。如果在消息控制块启动后没有接收到消息，应用程序接收初始化时设置的消息值。

接收消息的对象可以被定义为队列或非队列的。同时，一条被定义为队列的消息对象接收的消息只能被读取一次（读操作从队列中删除时间最长的消息），一条被定义为非队列的消息对象接收的消息可以被读取多次，它返回每次读取最后接收到的值。

每个被定义为队列的消息对象规定了消息对象的队列长度，并且该长度不能为0。如果接收消息对象的队列已经满，同时又来了一条新消息，那么这条消息就会被

丢失。

OSEK COM 不负责分配应用程序消息的内存，但是，它允许每个发送者和接收者独立访问消息对象。对于非队列消息，可以有任意数量的接收者接收消息。消息控制块可以保证应用程序消息副本中的数据含义一致性，消息控制块会自动处理消息。应用程序所需的消息数据只能在一个发送或接收服务调用期间被读取或写入。

通信规范中定义了三种不同的消息传输模式：

■ 直接传输模式：死限监控机制监控每次调用 SendMessage，SendDynamicMessge 或 SendZeroMessage 并检查底层在给定时间间隔内发送的确认。在调用 SendMessage，SendDynamicMessge 或 SendZeroMessageAPI 服务完成时，开启监控定时器功能。根据底层传输的确认和通知机制通知应用层来确定定时器是否被取消。

■ 周期传输模式：死限监控机制监控在给定时间间隔内 I-PDU 的传输。根据系统设计的约定，超时间隔可以大于传输周期。如果监控定时器不是正在运行，则在每个周期传输请求后开启监控定时器。根据底层任何监控 I-PDU 的传输确认，取消相应监控时间间隔的定时器。

■ 混合传输模式：限监控机制是指在给定监控时间间隔内传输一个 I-PDU。如果定时器不是正在运行，则它在每个传输请求后启动。定时器可以被底层监控 I-PDU 的任何传输确认取消。

### 3.2.3 通知机制

通知机制可以决定应用层此前调用的发送或接收操作的最终状态。一旦相应事件发生，就通知应用层。比如，用户不需要为了预先确保通知方案被激活而去调用专门 osekcomAPI。通知总是条件性通知，这意味着在过滤情况下只有数据在没有被过滤机制丢弃才会运行通知。同样的，如果队列消息由于缓冲区溢出而被丢弃，接收方是不会发送通知的。收发双方的每个消息对象都配置通知，通过监控包含传输消息的 I-PDU 和监控用于接收的消息对象来执行。

OSEK COM 支持消息发送和接收的四种通知类。类 1 和类 3 通知消息的接收者，而类 2 和类 4 通知消息的发送者，所有类都支持外部通信。

■ 通知类 1：消息接收。在消息被接收消息对象存储后，这个配置消息机制被立即调用。

■ 通知类 2：消息传输。在包含消息的 I-PDU 成功传输后，这个配置消息机制

被立即调用。

■ 通知类 3：消息接收错误。在消息接收错误通过死限监控机制或通过底层提供错误代码检测到后，这个配置消息机制被立即调用。

■ 通知类 4：消息传输错误。在消息传输错误通过死限监控机制或通过底层提供错误代码检测到后，这个配置消息机制被立即调用。

## 3.3 实现语言（OIL）

OSEK 实现语言（OSKE implementation language: OIL）的主要目的是以实现 OSEK 便携软件为目标，定义了一种 OSEK 应用的配置方法[46]。这个规范仅仅对一个电子控制单元的中央控制单元作了说明，不是网络电子控制单元。根据应用软件的实际需要配置操作系统及通信，以缩减最终生成可执行文件的体积。图 3-9 表明了基于 OSEK 规范的应用软件的开发过程。

OIL 被看成是由一系列 OIL 对象组成的， CPU 是这些对象的容器。OIL 为每一个对象都定义了它的标准类型，每一个对象都由一组属性和参数来描述。OIL 对每一个对象的标注属性都有明确的定义。对于一个 OSEK 的实现可以定义附加的实现属性和参数。对已存在的 OIL 对象增加新的属性是可以的，但是创建新的对象或改变语法结构是不允许的。没有标准属性（可选属性）的对象的属性，要看具体的实现，它们没有标准的解释。对于每一个实现都被限制必须给出一系列属性的取值范围。

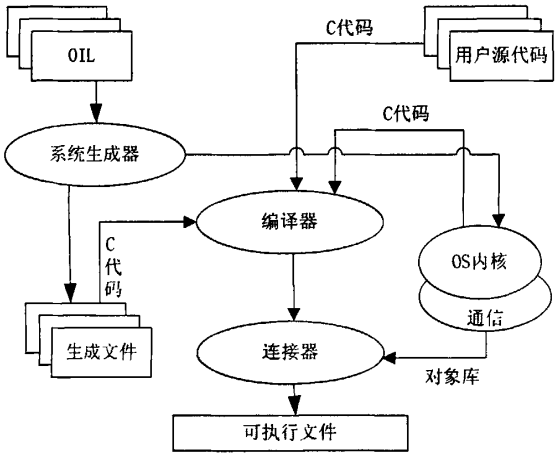



图3-9 应用软件开发过程示例

Figure 3-9 OSEK COM's layer model

### 3.3.1 与操作系统相关的OIL对象

OIL对象的描述：

- CPU：在OSEK子系统的控制下，应用程序在上边运行。
- OS： 运行在CPU上的OSEK操作系统。从OSEK操作系统到其他的OIL对象都没有标准的参数被定义。
- APPMODE：对一个应用中的操作定义了不同的模式。对于模式对象没有标准的属性定义。
- ISR：操作系统支持的中断服务程序。
- RESOURCE：能够被任务抢占的资源。
- TASK：能够被操作系统处理的任务。
- COUNTER：一个代表硬件或然见计时资源的计数器，它用于报警。
- EVENT：任务能够响应的事件。
- ALARM：报警是以计数器为基础的，它可以激活一个任务，设置一个事件或者激活一个报警回调程序。
- COM：通讯子系统。在OSEK通讯标准总对通讯对象的属性都做了定义。
- MESSAGE：消息在OSEK的通讯标准中给出了定义，对于一个具体的实现给出了一种通讯的机制，关于不同的实体（任务或中断）和CPU之间的数据交换。
- IPDU：OSEK的通信标准对其做了定义，它是在外部通讯时消息数据的载体。
- NM：网络管理子系统。

### 3.3.2 OIL的文件结构和语法

（1） 文件结构

OIL 包括两个部分，一部分是标准的定义和特定实现的定义，另一部分为在一个特定的CUP上的其应用结构的定义。OIL文件包括一个主要文件，它可以分成很多子文件。

（2） 语法

OIL文件的语法规则类似于巴科斯－诺尔范式（BNF）中用的符号文档。所有的关键字，属性，对象名和其他标识符区分大小写。注释符和c++中的注释符一样。

# 第四章 嵌入式实时操作系统的设计

## 4.1 系统概述

本文所设计的嵌入式实时操作系统只是一个操作系统的微内核，它提供了任务调度、任务间的通信与同步、任务管理、时间管理和内存管理等的基本功能。这种微内核的实现思路有很明显的优势，比如体系结构更加清晰、扩展性强等，而且由于内核保持很小，可移植性（不同的 CPU 间移植）也很强。但也存在弊端，就是效率相对低下。本系统主要由内核和通信系统组成，如图 4-1 是系统的整体结构图。其中操作系统内核的主要部分有：

- 任务管理：任务管理主要的功能是激活和中止任务；任务状态管理和任务的切换管理。
- 同步机制：主要功能是实现任务的同步，即事件管理和资源管理。
- 中断管理：处理内部中断和外围设备产生的中断。
- ALARM：管理相对报警和绝对报警。
- 消息处理机制：管理系统内部的通信。

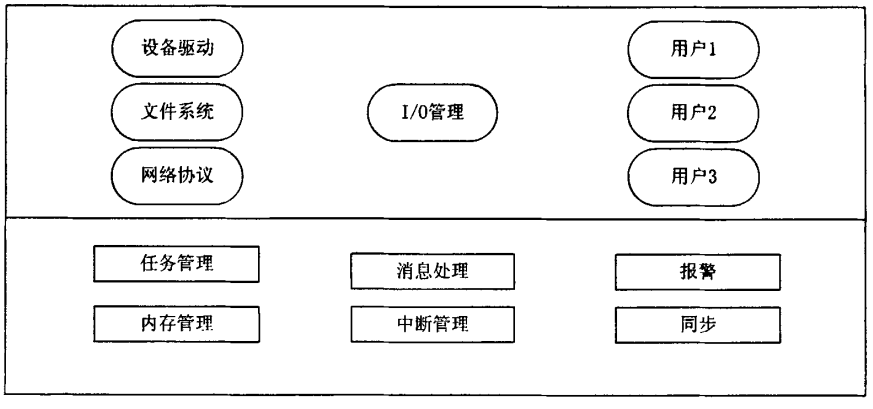



图4-1 系统的整体结构

Figure 4-1The overall structure of the systempreemptable extended tasks

该系统的功能特点：

- 多任务
- 可抢占调度
- 任务同步
- 共享资源互斥访问

- 内存管理
- 时钟管理
- 消息机制
- 中断调度机制

## 4.2 操作系统的任务管理模块

### 4.2.1 任务的分类及特征

任务是操作系统演进中的具有革命意义的概念。所谓任务，比较通俗的说法就是“一个运行起来的程序”，这就是说，一个任务，首先是一个程序，即一段代码，而且是一段已经运行起来的代码。在嵌入式操作系统中，任务的本质就是一个线程，但特别的是，任务是一个一直运行的循环。

本系统中实现了两类任务，基本任务和扩展任务，基本任务和扩展任务的区别就是扩展任务能够等待事件，能够进入等待状态，因此，在讨论事件时，它所涉及的任务只能是扩展任务。

对于操作系统来说任务的管理包括任务的组织、创建、撤销、修改任务的优先级等功能。基本任务有三种任务状态：就绪、运行、挂起；扩展任务比基本任务多一种状态即等待态，扩展任务可以调用操作系统的系统调用而改变自己的状态到等待状态。在不需要结束正在运行的扩展任务的情况下，等待状态允许处理器释放并且可以被其他任务抢占。基本任务和扩展任务的状态模型如图 3-2。

- 挂起：在挂起状态，任务是不起作用的，并且可以被激活。。
- 运行：在运行态，CPU 被分配给任务，因此它的指令被执行。在一个时间点仅有一个任务处于此状态。而其他的状态可以同时有几个任务。
- 就绪：所有任务进入运行态的首要条件，任务仅等待处理器的分配。调度程序决定下一个要运行的就绪任务。
- 等待：一个任务不能一直执行，因为它有可能至少要等待一个事件的发生。

每种状态的任务对象被组织在一个队列中（在单 CPU 环境中，状态是 Running 的任务不进入任何队列），每个队列都是一个优先级队列对象，因此，位于其中的任务对象可以按照先进先出的顺序进行调度。调度程序只选择就绪队列中的任务投入运行。

基本任务的一个优点就是它有合适的运行时上下文（RAM）请求。扩展任务的

一个优点就是它可以在单个任务中处理连贯的工作，无论哪个同步请求是活跃的。

系统的符合类的确定可以通过以下条件来确定：

- 任务能被请求启动的次数
- 任务类型
- 每个优先级的任务数
- 本文设计的系统实现了 BCC1 和 ECC1：
- BCC1：只有基本任务，每个任务只能有一个启动请求，每个优先级只能有一个任务，所欲任务都有不同的优先级。
- ECC1：与 BCC1 类似，加上扩展任务。

### 4.2.2 任务的优先级与调度策略

本系统实现了一种任务调度算法，即基于优先级的差分时限调度算法，该算法的该算法执行流程如下：

（1） 任务创建时初始化任务的优先级，确定任务优先级的值。

（2） 任务每运行一个时间片把所有有时限的任务的时限减去一个时间片的值。

（3） 确定时限是否到达，如到达则提高到达时限的任务的优先级进行调度，如没有任务到达，则按照原来优先级进行任务调度。

（4） 如运行的是提高优先级的任务，则在任务结束的时候恢复任务的优先级。

根据实现的调度算法，任务的调度可以分为三种调度策略，即完全抢断式调度、非抢断式调度和混合调度，如图 4-2，的优先级的任务 T2 并不耽搁高优先级任务 T1 的调度。

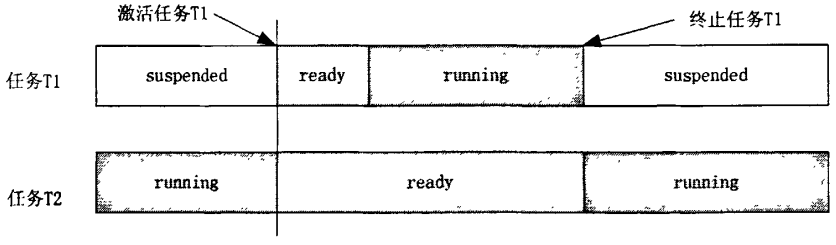



图4-2 完全抢断式任务

Figure 4-2 Full preemptive scheduling

图 4-3，低优先级的任务 T2 使得高优先级的任务 T1 直到 T2 结束时才得以启动。

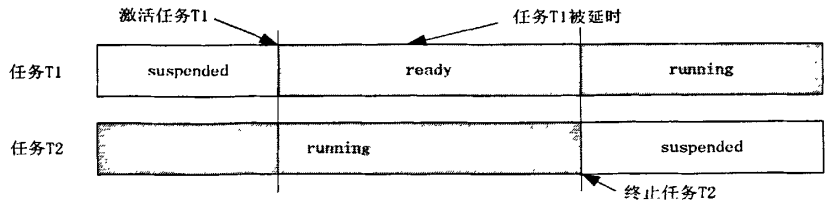



图4-3 非抢断式任务

Figure 4-3 Non preemptive scheduling preemptable extended tasks

如果抢断式和非抢断式任务混合在一个系统中，产生了“混合抢断调度”。这种调度策略依赖于当前运行任务的抢断特性。如果运行任务是非抢断的，则采用非抢断式调度；如果运行的任务是抢断的，则采用抢断式任务调度。

### 4.2.3 任务切换的上下文

当进行任务切换时，系统会检查任务的就绪队列，选择一个优先级最高的任务投入运行，从而确保最高优先级的任务得以运行。这就涉及到一个问题，如何保存当前任务的上下文状态，并选择另外一个任务，恢复其上下文，并投入运行。

假设任务调度前运行的任务是T1，这时候发生任务调度，CPU将在T1的堆栈中保存T1相关的寄存器的值，然后进行调度。然后调度器会从就绪队列中选择一个优先级最高的任务，假设任务为T2，然后恢复其寄存器信息，并建立其运行的堆栈架构。

## 4.3 内存管理机制

本系统采用的是uC/OS-II的内存管理机制，即，uC/OS-II把连续的大块内存按分区来进行管理。每个分区都包含若干个存储容量大小相同的内存块，但不同分区之间的内存块容量是可以不同的。在需要动态分配内存时，可以选择一个适当的分区，按块来分配内存。在释放时，将该块放回到它以前所属的分区，这样就有效地解决了内存碎片问题。

## 4.4 互斥和同步机制

在操作系统的设计中，尤其是引入多任务的情况下，需要考虑下列几种可能的“竞争”状态：

（1） 中断处理程序和应用程序之间的竞争，比如一个共享的数据对象既可以被应用程序修改，也可以被中断处理程序修改，这时候就需要充分考虑

应用程序和中断处理程序之间的竞争关系，因为中断随时可能发生。

（2） 应用程序与应用程序之间的竞争，比如两个应用任务共享同一个数据结构，且可以任意修改，这样就需要充分考虑如何避免修改过程中出现的冲突（一个任务修改了一半被替换出去，另一个任务继续修改同样的数据结构）。

（3） 在多 CPU 的情况下，除了考虑上述竞争关系之外，还需要考虑多个 CPU 之间的冲突关系。

本系统对这几种可能的冲突情况都做了充分的考虑，并实现了相应的机制来处理这几种竞争关系。在本系统中临界区用于完成中断处理程序和应用程序之间的同步，而应用程序之间的互斥与同步，则用来实现事件对象、互斥体对象和信号量对象，这些对象可支持标准的同步或互斥语义，也支持超时等待机制。

### 4.4.1 临界区概述

临界区是一段可执行的代码。这段代码在执行过程中，不能被打断，否则可能会产生严重的问题。一般情况下，临界区段内的代码往往对全局的数据进行修改或读取，这些数据为系统中所有的任务共享，如果在修改的过程中被打断，可能会导致这些全局数据处于一种不一致的状态。

在临界区段对关键代码进行保护的时候，需要遵循下列注意事项：

（1） 关键代码段不能太长，如果太长，可能会影响系统的整体效率。

（2） 在临界区段保护的代码中，不能调用可能堵塞任务继续执行的系统调用。比如不能在代码段中等待一个内核对象，这样可能会导致严重的资源浪费，甚至可能导致死锁的发生。

（3） 对于临界区段，只建议在操作系统核心的实现代码中使用，对于普通应用程序，不建议直接使用临界区段，而建议使用内核的对象如事件、信号量、互斥信号量等，来实现任务之间的同步和关键资源的保护。

### 4.4.2 临界区产生的原因

临界区的最根本目的是确保系统数据结构的一致性。而数据的不一致性一般是由竞争修改引起的。下面讲述的是操作系统中常见的竞争修改发生的原因。

（1） 多个任务之间的竟真

（2） 中断服务程序与任务之间的竟真

（3） 多个 CPU 之间的竞争

### 4.4.3 信号量

信号是一个内核同步对象，用来同步或保护共享资源的访问。通常情况下，对于信号量设定一个初始值，比如为 N。对于每个请求等待信号量的核心任务，操作如下：

（1） 判断 N 是否小于或等于 0，如果不是，则 N=N-1，并从等待操作中直接返回。这时候，等待信号量的任务得以继续执行。

（2） 判断 N 是否小于或等于 0，如果是，则说明当前没有可利用的资源，于是等待过程把请求的任务插入等待队列，然后执行一个重调度过程。需要注意的是，这种情况下也对 N 进行递减操作，不过这时候 N 已经称为负值，因此，如果 N 大于 0，N 的数值代表了当前可用资源的数量，如果 N 小于 0，则 N 的绝对值代表了当前等待信号量资源的任务数量（等待队列的长度）。

已经获得信号量的任务在执行完关键代码后，必须释放申请的信号量资源。释放资源的操作过程如下：

（1） 对 N 进行递增操作（加 1），如果 N 大于 0，则说明当前没有等待该信号量资源的任务存在，直接从释放函数中返回。

（2） 如果 N 小于或等于 0（递增之后），说明仍然有任务在等待该信号量资源，于是从等待队列中提取一个任务，把该任务修改为就绪状态，并插入到就绪队列中，然后返回。

可以看出，一次释放信号量的操作最多会唤醒一个等待任务，这与事件对象不同，事件对象的一次设置操作可以唤醒多个等待该事件的任务。而如果信号量的 N 的值为 1，则信号量就演变为互斥对象，可以用来保护临界区的并发访问。

信号量的等待操作支持超时等待的方式，即等待的任务可以设定一个时间长度，如果能立即获取资源（等待成功），则直接返回，如果不能立即获取资源被阻塞，则在设定的时间超时后，如果资源仍然不能获得，也会返回继续执行。这个功能在网络协议的实现中十分有用。

## 4.5 中断和报警处理机制

中断和异常处理以及报警机制，是操作系统必须实现的最重要的核心功能之一。

### 4.5.1 中断和异常

中断和异常是系统在运行过程中可能发生的外部或内部事件。一般情况下，中断是由外部设备引起的；而异常则一般是由软件造成的，泛指软件运行过程中产生的不正常的事件。比如最容易理解的是除法运算，如果出现了除数为 0 的情况，就会引发一个不正常的事件。

一旦检测到中断或异常发生，CPU 就会中断当前的处理顺序，并根据中断或异常的类型转移到相应的处理程序。需要注意的是，CPU 并不是在中断或异常发生后马上跳转到相应的处理程序，而是在指令的执行边界检查是否有异常或中断发生，如果没有，CPU 继续执行下一条指令。一旦有中断或异常发生，会引起 CPU 设置内部相应的寄存器位，然后继续执行当前的指令（中断发生时，正在执行的指令或者引发异常的指令），只有在当前指令执行完毕之后，相应的中断或异常才有机会得到处理。

本系统的设计中充分考虑了不同 CPU 对异常或中断处理机制的不同，采用了中断向量表跟中断处理链接的方式进行结合的实现方式，即首先有一个中断向量表，这样就可以根据不通的 CPU 选择不通的处理机制。

一旦中断或异常发生，CPU 首先根据中断或异常向量号找到对应的描述符，从描述符中提取出中断或异常的处理程序，然后把控制转移到中断或异常处理程序。

### 4.5.2 报警

报警是操作系统必备的一项重要服务。一般情况下，报警被用户任务使用来定时完成一项工作。在操作系统核心中，一些同步对象，比如事件对象、信号量对象、互斥对象等，支持超时等待操作，为实现超时等待操作，也需要使用操作系统提供的报警功能。在系统中提供了一些函数来实现对报警的操作如：

（1） SetTimer，设置一个报警。

（2） CancelTimer，取消设置的报警对象。

（3） ResetTimer，复位报警对象。

系统还提供了其他一些相关的编程接口函数。

考虑到报警的实现机制以及目标系统的性能要求，在实际应用中使用报警时要遵循下列规则。

（1） 使用报警时，如果不是十分必要，建议不要使用回调通知机制。因为报警的回调函数是在报警中断处理程序中被调用的，如果有大量的回调函

数存在，会大大降低系统的性能。

（2） 如果一定要用回调函数处理机制来处理超时，建议回调函数的处理时间不要太长，一般情况下不能大于 0.01ms。

（3） 采用消息通知机制作为超时处理机制的报警会引入误差。比如，假设设定的报警在 T1 时刻超时，则在 T1 时刻，设定报警的任务会收到一个报警超时的消息，但真正处理该报警超时的消息可能会延迟到 T2 时刻。但是对于采用回调函数作为超时处理机制的报警，在 T1 时刻，回调函数就可以被调用，即超时消息马上被处理。

# 第五章 任务调度算法的优化及实现

在非实时存在系统中，调度的主要目的是减小系统的平均响应时间，提高系统资源利用率，或优化某一项指标；而实时调度的目的则是要尽可能地保证每个任务满足它们的时间约束，及时对外部请求做出响应。实时操作系统不仅要求系统的逻辑正确性，还必须要求系统能在规定的时间内计算出正确的结果并对实时事务进行响应，因此实时系统中的调度算法是决定系统实时性的重要指标。任务调度算法是实时操作系统中的关键技术之一，也是实时操作系统中最重要、最活跃的研究领域之一。

为了达到实时操作系统的实时性和可预测性要求，并能够满足实时系统的新要求，须用实时调度理论对任务调度策略进行分析和优化。任务调度技术的研究包括任务使用系统资源的策略和机制，以及提供判断系统性能是否可预测的方法和手段。例如：什么时候调度任务运行、运行多长时间等等，以及判断分析用一定参数描述实时任务能否被系统正确调度。

实时操作系统调度算法的主要职责就是确保所有任务都能满足任务的时间约束性要求。现在已经提出了很多种实时调度算法，但还没有能够同时适应所有情况的调度算法。为了更好地满足任务调度多样化的需求，仍需继续研究适应不同应用场合的调度算法和普适性较强的最优调度算法。

针对上述分析，本文提出了一种新的调度算法，基于优先级的差分时限调度算法。该算法在保证高优先级任务得以运行的同时保证达到时限的低优先级的任务得以运行，从而保证了系统的实时性要求。

## 5.1 实时调度算法的分类

实时调度算法通常有多种划分方法，主要存在以下几种划分方法：

- 离线和在线调度
- 静态和动态调度
- 抢占和非抢占调度

根据获得调度信息的时机，调度算法分为离线和在线调度两类。对于离线调度算法，运行过程种使用的调度信息在系统运行前就确定了，如时间驱动的调度。离线调度算法具有确定性，但缺乏灵活性，适用那些特性能够确定，且不容易发生变化的应用。在线调度算法的调度信息则在系统运行过程中动态获得，如优先级驱动

的调度。在线调度算法在形成最佳调度策略上具有较大的灵活性。

优先级驱动策略指按照优先级的高低确定任务的执行顺序。根据任务的优先级的确定时机，调度算法可分为静态优先级调度和动态优先级调度两类。在静态优先级调度中，任务的优先级在设计时就确定下来了，且在运行过程中不发生任何变化。静态优先级调度算法适用于能够完全把握系统中所有任务及时间约束特性的情况。静态优先级调度比较简单，但缺乏灵活性，不利于系统的扩展；动态优先级调度有足够的灵活性来处理变化的系统情况，但需要消耗更多的系统资源。

根据任务在运行过程中能否被打断的处理情况，调度算法分为抢占调度和非抢占调度两类。抢占式调度通常是基于优先级驱动的调度。在抢占调度算法中，正在运行的任务可能被其他高优先级任务所打断，除了共享资源的临界段之外，高优先级任务一旦准备就绪，可在任何时候抢占低优先级任务的执行。在非抢占调度策略中，一旦任务开始运行，该任务只有在运行完成而主动放弃 CPU 资源，或是因为等待其他资源被阻塞才会停止运行。实时内核大都采用抢占式调度算法，使关键任务能够打断非关键任务的执行，确保关键任务的截止时间能够得到满足。抢占式调的优点是实时性好，反应快，调度算法简单，可优先保证高优先级任务的时间约束，但抢占式调度算法要更复杂些，且需要更多资源，并可能在使用不当的情况下造成优先级翻转的问题。非抢占式的调度常用于那些任务需要按照预选确定的顺序执行，且只有当任务主动放弃 CPU 资源后，其他任务才能得到执行的情况。

目前大多数实时操作系统采用两个使用普遍的实时调度算法：

- 优先级抢占调度
- 时间片轮换调度

（1） 优先级抢占调度算法

大多数实时内核缺省使用优先级抢占调度算法。每个任务都有一个优先级，在任何时候运行的任务是所有任务中具有最高优先级的任务。优先级抢占调度算法又分为动态优先级抢占调度算法和静态优先级抢占调度算法。

➢ RM(Rate Monotonic)调度算法是一种典型的静态优先级抢占调度算法，也是实时调度研究中最经典的算法。RM 算法共有 3 个条件：①由独立的周期性任务组成，且各任务都具有静态优先级；②分配优先级的策略是一个任务的周期越短，其优先级越高；③采用可抢占的优先级调度方式。

➢ EDF(Earliest Deadline First)算法是一种典型的动态优先级抢占调度算法，该算法根据就绪队列中各个任务的截止期限来分配优先级，具有最近截止期限的任务

的优先级最高。

（2） 时间轮换调度

时间轮换调度为每个任务提供相等份额的 CPU 执行时间，但纯粹的时间轮换不能满足实时系统的要求，因为在实时系统中，任务的执行随其重要性变化。

（3） 其他实时调度算法

➢ FCFS(First Com First Served)算法是最简单的先到先服务算法。调度的依据是每个任务到达的时间 s。每次调度进程选择，最大的任务 $T_s$ 占 CPU 运行，$T_s$ 在运行过程中不能被其他任务中断。显然，FCFS 算法是不能保证任务的时限。系统中的任务个数极少，并且在任务的时限不急迫的情况下，FCFS 才使用。

➢ LLF(Least Laxity First)算法是具有最短松驰时间的任务优先调度。松驰时间是任务 $T_i$ 的当前时间 $t_i$ 和任务时限 $d_i$ 之间的时间差减去执行时间 $e_i$。由于任务在执行中其松驰时间是动态的，所以可能出现这种情况：一个任务 Ti 在执行过程中。另一个任务 $T_j$ 的松驰时间变得更短了，那么 $T_j$ 就抢占 $T_i$ 运行，紧接着 $T_i$ 的松驰时间又变短，$T_i$ 又抢占 $T_j$。这样两个任务彼此抢占，造成不能调度新任务，任务切换的开销巨大。

➢ DM(Dedline Monotone)是单调时限调度算法。如果任务 $T_i$ 的时限小于它的周期，那么它就违反了 DM 算法的前提。即使任务 Ti 的速率高于任务 $T_j$，也可能出现在任务 $T_i$ 执行中由于任务 $T_j$ 的时限提前而被抢占。使用 DM 算法必须动态检测每个任务的时限，进行排队，这将使调度十分复杂。

➢ SJF(Shortest Job First)是最小执行时间优先算法。调度器在调度准备时从就绪队列中选最小执行时间的任务。SJF 算法可以保证较小执行时间任务的时限，但是对于执行时间长、时限要求紧迫的任务，它往往不能保证任务的时限。

## 5.2 任务调度算法的优化

### 5.2.1 不同优先级任务调度的优化

（1）EDF 算法

EDF 算法是 Liu 和 Layland 提出来的，它采用动态调度，具有较高的 CPU 使用率。在 EDF 算法中任务的优先级根据任务的截止时间来确定。任务的绝对截止时间越近，任务的优先级越高；反之越低。当有新的任务处于就绪状态时，任务的优先级就有可能需要进行调整。对于 EDF 算法的使用是有条件的，其成立的条件是：

① 任务的相对截止时间等于任务的周期；

② 任务在每个周期内的计算时间都相等，保持为一个常量；

③ 任务之间不进行通信，也不需要同步；

④ 任务可以在计算机的任何位置被抢占，不存在临界区。

EDF 算法是最优的单处理器动态调度算法，其可调度上限为 100%。在 EDF 调度算法下，对于给定的一组任务，任务可调度的充分必要条件为式 5.1：

$$\sum_{i=1}^{n}\frac{C_i}{T_i}\le 1 \quad (5.1)$$

其中，Ti 为任务的周期，Ci 为任务的运行时间，n 为系统中的任务总数，i 表示低 i 个任务。

对于非周期任务任务的运行时间就应该是 $C_i+\Delta t$，其中 $\Delta t$ 为任务切换所消耗的 CPU 的时间，实际情况下，任务的调度对 CPU 所带来的负担不可忽略。那么式 5.1 就变成如下形势如式 5.2。

$$\sum_{i=1}^{n}\frac{C_i+\Delta_i}{T_i}\le 1 \quad (5.2)$$

在实际系统中，EDF 算法中是根据任务的时限来确定任务的优先级的，在任务的时限发生变化时重新确定任务的优先级这是一个非常复杂的工作，严重影响了系统的实时性，浪费了系统资源，而且不能保证的优先级任务的实践性约束，对此，本文在 EDF 的基础上提出了一种新的任务管理机制和任务调度算法，即基于优先级的差分时限调度算法。

（2）不同优先级任务调度算法的改进

在此算法中，系统首先根据抢占式优先级调度算法就系统中就绪队列中的任务进行调度，运行最高优先级的任务，在每次调度开始前要检查任务的时限，看是否有任务到达时限，如果有任务到达时限则提高该任务的优先级，然后再根据抢占式优先级调度算法进行任务调度，系统检测时限和修改任务的优先级的程序代码是不能被打断的，该段代码被处理成临界资源，系统在运行这段代码时是关中断的。然而，在检查时限，修改系统中所有任务的优先级时，逐一的为每个任务进行时限更新，检测将耗费很长的时间，造成了严重的资源浪费，降低了系统的实时性。系统中的任务数越多，这种浪费和影响就会越严重。

对此本文通过链表结构——差分时限链，来解决此问题。通过差分时限链来更

新、检测每个任务的时限。差分时限链中的每一个元素均为结构体变量。结构体成员由前向指针、后向指针和时限 3 个变量组成。

差分时限链中任务时限值和插入位置的确定。当第 $i(i\geq 0)$个任务建立时给定任务的时限为 $D_i$，差分时限链中该任务的时限为 $d_i$，其大小由式(5.3)确定，其中 $dj'(0\leqslant j<n)$

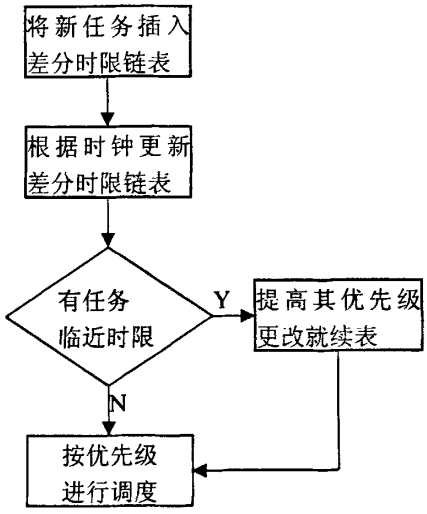



图 5-1 任务调度算法流程图

Figure 5-1Algorithm flow char of Task Scheduling

为差分时限链中第 $j$ 个元素所指任务在时限链中的时限值，n 为差分时限链的长度。插入的位置，应插入在差分时限链中第 $j$ 个元素所指任务的后边。

$$d_i = D_j - d_0 - d_1 - \cdots d_j > 0 \cdots\cdots (5.3)$$

如果插入的元素不是在差分时限链的最后，则新插入元素后边各元素的值应为式(5.4)的值$(0\leq k<n-j-1)$。

$$d_{j+2+k} = d_{j+2+k} - d_{j+1+k} > 0 \cdots\cdots (5.4)$$

在任务调度时，首先根据系统时钟的计时，更新差分时限链。在差分时限链中，只需将差分时限链首元素的时限减去计时时钟即可得到要插入任务的时限，如此就省去了为每个任务逐一进行时间更新的巨大时间开销。然后检测差分时限链首元素对应的任务是否临近其时限。若是，则将该任务优先级相应提高，而后进行基于优先级的抢占调度；否则，直接进行优先级调度。

差分时限链结构使得系统在进行任务调度时不用去检查所有给定了时限的任务是否达到其时限，只需要对差分时限链链表表头的任务进行检查即可。因在差分时

限链结构中，链表表头的任务一定是所有给定了时限的任务中最早到达其时限的任务。这样就大大降低了系统的开销，提高了系统的实时性。

在一次任务调度的开始，需通过系统时钟对差分时限链更新。当满足式(5.5)时，表明差分时限链首任务接近时限，应将其优先级相应提高。

$$0 \le t_i' - r_1 - \Delta \le t_d \quad \cdots\cdots (5.5)$$

其中，$t_i'$为更新后差分时限链链首元素的时限 $r_1$ 为任务执行所需的时间；△为每次优先级更新检测及任务上下文切换所需时间；$t_d$ 为定义的判断系数，表示任务到达时限的接近程度。

差分时限优先任务调度算法的流程如图 5-1 所示。

在发生任务抢占时，任务控制块中的时间变量(*ntime*)来记录当前运行任务的运行时间，并依据记录的时间来更显差分时限链中的时限。在一次检测完成时，要对 任务控制块中的时间变量清零，以便完成下一次的时间记录，如果不清零可能会给系统带来严重的后果。如果没有达到时限的任务在其到达时限前已经被系统处理完，则应该在差分时限链中将该任务的时限块与差分时限链断开，并更新时限链中各任务的时限。

### 5.2.2 同优先级任务调度的优化

对 OSEK 操作系统中同一优先级的多个任务之间的协调处理可借鉴时间片轮转调度法。所谓时间片轮转调度法是指当有两个或多个就绪任务具有相同的优先级，且它们是就绪任务中优先级最高的任务时，任务调度器按照这组任务就绪的先后顺序调度第一个任务，让第一个任务运行一段时间；然后又调度第二个任务，让第二个任务又运行一段时间；以此类推，到改组最后一个任务也得以运行一段时间后，接下来又让第一个任务运行。这里任务运行的时间段就是时间片。

为了实现同一优先级下的多个任务按时间片轮转调度算法进行任务调度，在系统中为每个优先级下的任务增加一个先进先出队列。在任务的运行过程中，同优先级下的任务每运行完一个时间片后都要进行优先级检测，即任务调度，检查是否有比当前运行着的任务的优先级更高的任务就绪，如果有，则按照基于优先级的差分时限调度算法进行调度，运行优先级更高的任务，把当前正在运行的任务挂在该优先级先进先出队列的队尾。如果没有更高优先级任务就绪，就按照时间片轮转方式进行任务调度，继续运行该优先级下的下一个任务。

## 5.3 任务调度算法在 uC/OS-II 的实现

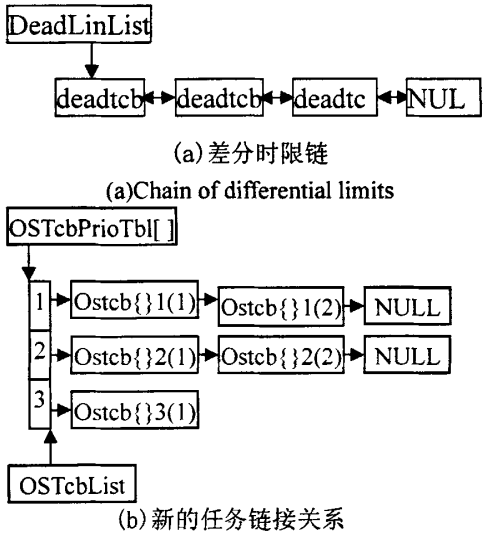
### 5.3.1 任务管理机制的实现

在 uc/OS－II 操作系统中通过一个任务控制块的结构体－－OS_TCB{}来表示一个任务，为了实现在一个优先级下有多个任务，在 OS_TCB{}中要添加一个指向同优先级任务的指针(*prioequ)，从而实现同优先级任务的 FIFO 调度策略。但要注意如果同优先级下有多个任务，任务就续表位图 OSRdyTbl[]相应的位必须在此优先级下的任务全部执行完后才能清零。

在不同优先级下加入了差分时限优先任务调度算法的机制，为了实现差分时限链，需为每个任务添加一个结构体(deadtcb)，其有任务 ID，前向指针(*Dpre)，后向指针(*Dnext)，任务控制块指针(*OSTCBDead)和时间限(deadline)四个元素构成。在 OS_TCB{}中要添加一个指向差分时限链结构体的指针(*DeadTcb)，任务号(ID)两个元素。改进后的任务结构如图 5-2，其中(a)差分时限链，(b)为新的任务连接关系。

### 5.3.2 就绪算法的实现

对于新任务的初始化以及根据优先级(prio)把 OSRdyGrp[4]和 OSRdyTbl[]置位的




(a)差分时限链

(a)Chain of differential limits

(b)新的任务链接关系

(b) New relationship of the task link

图 5-2 任务管理结构

Figure5-2 Structure of task management

操作已经省略。

根据进入就绪态任务的 prio，查找 OSTCBPrioTbl[]，如果该优先级下还没有任务，则通过 OSTCBList(指向任务链表的表头)把此任务加入到链表的表头，如果此优先级下已经有任务存在，则直接添加到 FIFO[47]列的队尾；并根据任务的实现将任务添加到差分时限链中。

就绪算法的伪代码：

```
void OS_ReadyTask( )
   {   if(此优先级下OSTCBPrioTbl[]没有置位)    //此任务是该优先级下的第一个任务
          {   通过OSTCBList将任务添加到任务链表表头；
              初始化任务控制块TCB中的相应项；
              将就绪表中的OSRdyTbl[ ]和OSTCBPrioTbl[ ]相应位置位；
              根据任务时限将任务添加到任务的差分时限链表中；  }
       else
          {   找到同优先级任务链表的末尾，将任务添加到链表末尾；
              置新增任务的next指针为NULL；
              根据任务时限将任务添加到任务的差分时限链中；}}
```

### 5.3.3 任务脱离就绪队列算法的实现

由于引入了同优先级的 FIFO 队列和任务差分时限链，当任务脱离链表时，不能简单地对就绪表清零，对于不同优先级的任务，如果任务执行完成或挂起，则需要检查任务的差分时限链是否存在该任务，如果存在，需要在差分时限链中删除该任务；另外，需对同优先级下是否存在多个任务加以判断；当满足条件：(1)在 prio 优先级下只有一个任务；(2)对于 OSRdyTbl[prio]所表示的 8 个优先级下均为空；才会将 OSRdyGrp 的某位清零，即表示在该位所表示的一组优先级下没有任务，而在同一优先级下有多个任务时，需要把指向同优先级链表头的指针后移一个节点，指向同优先级的下一个任务，以使当下一次调度产生的时候，调度器能以 O(1)的时间复杂度选取下一个任务，伪代码如下：

```
void OSClearReady( )
  { 根据OSRdyGrp[ ]和OSRdyTbl[ ]计算出最高优先级任务；
    if(此优先级下没有多个任务)
       {if(差分时限链表中的表头是此任务)
              {在差分时限链表中删除此任务；
               把OSRdyTbl[ ]相应位清零；
               if（OSRdyTbl[ ]所表示的优先级下任务均没就绪）
                     {OSRdyTbl[ ]的相应位清零}}
         else {把OSRdyTbl[ ]相应位清零；
                  if（OSRdyTbl[ ]所表示的优先级下任务均没就绪）
                    {OSRdyTbl[ ]的相应位清零}}
    else{同优先级链表的表头指向下一个任务；
```

```
OSSched( ); }}
```

### 5.3.4 任务调度算法的实现

在多任务操作系统中，任务调度是一个全局性的、关键性的问题；它对系统的总体设计，系统的实现，功能设置以及各方面的性能都有决定性影响。按照上述差分优先任务调度算法，在 uC/OS-II 中要做相应的改进。

为了判定任务的时限，要通过任务控制块中的执行时间变量(ntime)来计时，根据系统时钟来改变它；如果有任务达到时限，用系统的 OSTaskChangePrio()来改变其优先级，并放入就绪队列，就续表的相应位置位。下边是任务调度函数(void OSSched( ))，它先检查是否有任务到达时限，有则改变其优先级，后按优先级进行调度，无则直接按优先级进行调度。对于同一优先级下有多个任务的，调度时检查是否为同优先级任务。部分伪代码如下：

```
ntime++;
void OSSched( ){根据ntime更新差分时限链表；
                if(有任务到达时限)
                 {用OSTaskChangePrio()来改变其prio；
                 根据新prio将任务放入就绪队列；
                 就续表相应位置位；
                 根据任务prio进行任务调度；}
                else{直接根据任务的prio进行任务调度；}}
```

## 5.4 测试

通过实例对差分时限优先任务调度算法与 EDF 算法、基于指定优先级的占先式任务调度算法作对比分析。在系统中构造三个任务，并根据其释放时刻建立任务，同时确定其优先级，执行时间和时限；如表 5-1 任务集 S 所示。用 Di 来标识第 i 个任务的时限，执行时间和时限的单位为微秒(μs)。

表 5-1 任务集 S

Table5-1 task set S

| 任务 | 指定优先级 | 释放建立时刻 | 执行时间 (μs) | 时限 (μs) |
|---|---|---|---|---|
| T1 | 5 | 0 | 10 | 30 |
| T2 | 8 | 4 | 10 | 25 |
| T3 | 7 | 5 | 3 | 10 |

根据任务集 S 设计测试程序，分析 EDF 算法、基于优先级的抢占式任务调度算法和差分时限优先任务调度算法。

针对基于指定优先级的占先式任务调度算法，测试程序在任务切换时刻打印出原来运行的任务和执行时间；当总时间到达 20μs 时，停止系统的运行。可以得出如表 5-2 中基于指定优先级占先式任务调度算法栏的结果。从结果中可以看出 T1 和 T2 都在时限内完成，而 T3 超时，没有得到执行。

在 EDF 算法中，对任务集 S 进行调度，任务 T1 运行到时刻 4，T2 释放，根据时限的接近程度分配优先级，因 D2<D1，所以 T2 优先级高于 T1，T2 开始运行。在时刻 5，因 D3<D2，所以 T3 优先级高于 T2，T3 开始运行，直至结束；T2 开始运行，最后 T1 运行。其中有 4 次任务切换，每次任务切换的时间包括查询就绪表和任务优先级更新时间、任务上下文切换时间，其中任务上下文切换时间对所有调度法是共有的，可省去不计。设查询就绪表和任务优先级更新时间为 Δt，可得出如表 5-2 测试结果中 EDF 栏的测试的结果。

表5-2 测试结果

Table5-2 Results of test

| 基于指定优先级占先调度算法 | | | EDF | | | 差分时限优先任务调度算法 | | |
|---|---|---|---|---|---|---|---|---|
| 时刻 | 已运行的任务 | 运行时间（μs） | 时刻 | 已运行的任务 | 运行时间（μs） | 时刻 | 已运行的任务 | 运行时间（μs） |
| 0 | T1 | 0 | 0 | T1 | 0 | 0 | T1 | 0 |
| 4 | T1 | 4 | 4 | T1 | 4 | 4 | T1 | 4 |
| 5 | T2 | 1 | 5 | T2 | 1-$\triangle t$ | 10 | T2 | 6 |
| 14 | T2 | 9 | 8 | T3 | 3-2$\triangle t$ | 13 | T3 | 3-$\triangle t$ |
| 20 | T1 | 6 | 17 | T2 | 9-3$\triangle t$ | 17 | T2 | 4-$\triangle t$ |
| -- | -- | -- | 23 | T1 | 6-4$\triangle t$ | 23 | T1 | 6-$\triangle t$ |

在差分时限优先任务调度算法中，对任务集 S 进行调度，任务 T1 运行到时刻 4，T2 释放，由于 T2 指定的优先级高于 T1，且此时没有任务临近时限，则 T2 抢占 T1 开始执行。在时刻 5，T3 被释放，由于 T3 的指定优先级低于 T2 且没有任务临近时限，T2 继续执行。当到达时刻 10，任务 T3 临近时限，将 T3 的优先级相应提高(设提高到 9)，高于 T2 的优先级，因此 T3 开始执行。在时刻 14，T3 结束运行，T2 高于 T1 的优先级，开始运行。到时刻 20，T2 运行结束，T1 继续运行，直至结束。整个过程只进行了一次查询就绪表和任务优先级更新如表 5-2 测试结果中差分时限优先任务调度算法栏的结果。

从测试结果中可以看出基于指定优先级的任务调度算法不能保证低优先级的任务在时限内得以执行；而 EDF 算法中的任务切换过于频繁，增加了系统的开销；差分时限优先任务调度算法在这两方面都有了明显改进，在保证低优先级任务得以运行的情况下降低了系统的额外开销，从而大大提高了系统的性能和实时性。

# 第六章 总结和展望

## 6.1 总结

本文首先介绍了嵌入式系统，嵌入式操作系统及嵌入式实时操作系统相关的基本概念并对嵌入式实时操作系统的主要特性做了分析，说明了嵌入式实时操作系统与嵌入式系统之间的关系及嵌入式实时操作系统的主要功能；然后分析OSEK/VDX操作系统规范、通信规范和实现语言。根据对OSEK/VDX规范分析，讨论了OSEK/VDX操作系统内核的设计。重点讨论了系统的优化，包括任务调度算法的优化。其中任务调度算法的优化首先分析了现在流行的几种嵌入式实时操作系统中的任务调度算法，并根据这些算法提出了一种新的任务管理机制和任务调度算法，从而在保证了高优先级任务得以运行的同时保证到达时限的低优先级的任务也能运行，从而提高了嵌入式实时操作系统的实时性。

## 6.2 展望

随着嵌入式系统的发展嵌入式实时操作系统将迎来一个新的发展阶段。通用性将是下一步需要研究的重点。从某种程度上讲，不会出现一个标准的实时操作系统，因为嵌入式应用本身就极具多样性。因此下一步嵌入式操作系统研究的重点使得嵌入式操作系统具有更加经济适用的、开放的、容易移植的并具有通用性的实时操作系统环境。

# 参考文献

[1]2008 中国汽车电子产业现状和展望(组图)_业界_科技时代_新浪网. http://www.sina.com.cn. 2008 10 17.

[2]国外汽车电子技术的发展 国内汽车电子产品市场现状及预测. http://www.onlylt.cn. 2008 12 9.

[3]马洪连. 嵌入式系统设计教程. 北京:电子工业出版社, 2006.

[4]张晓林,崔迎炜等. 嵌入式系统设计与实现, 北京:北京航空航天大学出版社,2006.

[5]罗蕾. 嵌入式实时操作系统及应用开发. 北京航空航天大学出版社,2007.

[6]张茹,孙松林等. 嵌入式系统技术基础. 北京:北京邮电大学出版社,2006.

[7]朱珍民. 嵌入式实时操作系统及其应用开发. 北京邮电大学出版社,2006.

[8]陈翌等. 嵌入式软件开发技术. 北京:国防工业出版社,2005.

[9]Jean J.Labrosse. 邵贝贝等译. 嵌入式实时操作系统 uC/OS-II. 北京:北京航空航天大学出版社,2003.

[10]田泽等. 嵌入式系统开发与应用教程. 北京:北京航空航天大学出版社,2005.

[11]张大波等. 嵌入式系统原理、器件与应用. 北京:机械工业出版社,2005.

[12]Caroline Yao,Qing Li,王安生译. 嵌入式实时系统的概念. 北京:北京航空航天大学出版社,2004.

[13]慕春櫟. 嵌入式系统的构建. 北京清华大学出版社,2004.

[14]探矽工作室. 嵌入式系统开发圣经. 北京:中国铁道出版社,2003.

[15]吴明晖. 基于 ARM 的嵌入式系统开发与应用, 北京:人们邮电出版社,2004.

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

[17] 谢敏,李桥梁. 嵌入式实时操作系统任务调度算法优化. 电子科技,2005,12:24-26.

[18]孔祥营, 柏桂枝. 嵌入式实时操作系统 VxWorks 及其开发环境 Tornado. 北京:中国电力出版社,2001.

[19]钟汉如,王创生. 嵌入式 Linux 的中断处理与实时调度的实现机制. 计算机工程,2002,28(10).

[20]苏娟,吴旭光,张朝. 嵌入式操作系统μC/OS-II内核改进及其应用扩展. 计算机工程,2007,33(15):61-63.

[21]Caroline Jean. 嵌入式实时操作系统提 uC/OS-II(第二版). 邵贝贝等译,. 北京:北

京航空航天大学出版社,2003.

[22]陈予吒. 基于 uC/OS-II 的嵌入式系统设计与实现. [硕士学位论文],杭州:浙江大学电气工程学院,2004.

[23]Shoji Suzuki, Wataru Nagaura, Takaaki Imai, et al. A Distributec Control System Framework for Automobive Powertain Control with OSEK Standard and CAN Network. Society of Automotive Engineers Congress and Exposition, March 1999.

[24]K. M.Zuberi, P.Pillai, K.G. Shin, et al. EMERALDS-OSKE: A Small Real-Time Operating System for Automotive Control and Monitoring. Society of Automotive Engineers Congress and Exposition March 1999.

[25]S. Suzuki, W. Nagaura, T. Imai, et al. A distributed control system framework for automobive powertrain control with OSEK standard and CAN network. Society of Automotive Engineers Congress and Exposition, March 1999.

[26]Zonghua Gu, Shige Wang and Kang G.Shin. Issues in Mapping from UML Real-Time Profile to OSEK API. Workshop on Specification and Validation of UML models for Real-Time and Embedded Systems, 2003.

[27]Peter H.Feiler. Real-Time Application Development with OSEK A Review of the OSEK Standards[R]. SEI Technical Reports, CMU/SEI-2003-TN-004. November 2003.

[28]Uwe Kiencke, Ansgar Mainsch, Karl Joachim Neumannn. Stadardization of an open platform for distributed automotive Real-time Systems OSEK/VDX. Real-time Magazine, 97(3):64-73.

[29]Motorola OSEK Operating System USER'S MANUAL.

[30]Advanced Compiling Techniques to reduce RAM Usage of Static Operating Systems olker Barthelmann Erlangen–2004.

[31]Khawar M. Zuberi, Padmanabhan Pillai, Kang G. Shin. EMERALDS-OSEK: A Small Real-Time Operating System for Automotive Control and Monitoring. Society of Automotive Engineers.

[32]GUERNIC P LE, BORGNE M LE, GAUTHIER T. Programming Real Time Applications with Signal. Another Look at Real Time Programming. Proceeding of the IEEE Special Issue, Sept. 1991.

[33]王苗苗.嵌入式系统设计与实例开发. 北京:清华大学出版社,2003.

[34]陈文智.嵌入式系统开发原理与实践. 北京:清华大学出版社,2005.

[35]刘峥嵘. 嵌入式 Linux 应用开发详解. 机械工业出版社,2005.

[36]符意德. 嵌入式系统设计原理及应用. 北京:清华大学出版社,2004.

[37]资小龙,李莹. OSEK 车用嵌入式 RTOS 的设计[J]. 工业控制计算机,2005 18(3): 53-54.

[38] 陈卓,熊忠阳. 基于 OSEK/VDX 操作系统的任务管理机制设计[J]. 计算机工程,2006,32(12):82-84.

[39]蓝枫叶. 自己动手写嵌入式操作系统. 电子工业出版社,2007.

[40]袁铭蔚,孙泽昌, 陈觉晓. 一种嵌入式实时操作系统——OSKE/VDX OS. 测控技术,2003 22(12).

[41] ]刘波澜,孙业保. OSEK/VDX——汽车电子系统的开放式平台. 车辆与动力技术,2002 86(2): 61-64.

[42]李善平. Linux 与嵌入式系统. 浙江大学出版社,2003.

[43] The OSEK/VDX group. OSEK/VDX Operating System, Version 2.2.3[EB/OL]. February 17th,2005. http://rottal.osek-vdx.org/files/os223.pdf.

[44] The OSEK/VDX group. OSEK/VDX Time-Triggerec Operating System, Version1.0[EB/OL]. July 24th, 2001. http://portal.osek-vdx.org/files/pdf/specs/ttos19.pdf.

[45] The OSEK/VDX group. OSEK/VDX Cmmunication, Version3.0[EB/OL]. July 20th, 2004. http://portal.osek-vdx.org/files/pdf/specs/osekcom303.pdf.

The OSEK/VDX group. OSEK/VDX Network Management Concept and Application Programming Iterface, Version2.5.3[EB/OL]. July 26th 2004.

[46] The OSEK/VDX group, OSEK/VDX system generation. OIL: OSEK implementation language,Version.5[EB/OL].July1st,2004. http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf.

# 致 谢

时光飞逝，三年的研究生生活就要结束了，回想起三年的日日夜夜，回想起身边的老师、同学和朋友们，感慨万千。

本学位论文是在导师高慧敏教授的精心指导下完成的。从论文的选题直到最终结果，高老师以渊博的学识、严谨的学风和丰富的科研经验给予本人很多的启示和鞭策，而且也将激励我在专业领域的工作中继续努力奋进，使本人能够更全面、严谨、深入地分析和研究问题。

在此，我谨向我的导师表示深深的敬意和最诚挚的感谢。

感谢复杂系统与计算智能实验室的同学们，他们牺牲自己的休息时间，不断督促着我，指导着我，让我养成了定期总结工作，制定计划的好习惯，这些是我顺利完成论文所必不可少的，也为以后能够胜任工作岗位奠定了基础。谢谢你们！

最后，特别感谢我的父母，感谢父母的养育之恩，父母的理解、支持和鼓励伴随着我成长，我每前进一步都包含了他们无尽的关爱和辛勤的汗水。

感谢评阅本文的所有老师，感谢本文引用参考文献的作者，感谢所有帮助过我的人们!谢谢!

# 研究生期间发表的论文

毛成勇，高慧敏。《基于 OSEK 标准的任务调度算法的改进及实现》，计算机工程，2010 年第 4 期。

# 个人情况及联系方式

毛成勇，男，山东德州人，太原科技大学计算机科学与技术学院，计算机应用技术专业硕士研究生（2006.9－2009.7）。

Email: maochengyong0624@163.com