华北电力大学（保定）

硕士学位论文

实时多任务集成调度算法的研究

姓名：冯艳红

申请学位级别：硕士

专业：计算机应用技术

指导教师：张玉明

20051220

摘　　要

本文通过研究在一个实时系统中会有多类型任务的出现，将实时任务按照到达模式和截止时间类型两个纬度进行分类。并且在分析了诸多的调度算法中，对截止时间单调调度算法进行了详细的研究，它已被证明是最优的静态调度算法，比起动态调度算法的复杂性更易于在实际的操作系统中实现，所以成为了实时系统的支柱性技术。此基础上，本文提出并实现了一个实时调度模型（文中称之为 SM 模型），在 SM 模型中将实时任务划分为可调度性能够保证的 ScheduableTasks 和可调度性不能够保证的 ManageableTasks 两个任务集合，对它们分别采用不同的调度策略。本文研究的重点为：将任务集划分为 ScheduableTasks 和 ManageableTasks 两个集合的思想，最大化 ScheduableTaks 的算法，硬实时任务的调度优化算法，ManageableTasks 的处理方法。最后本文对于 SM 模型提出了进一步改进的方向。

关键词：实时调度，截止时间单调调度，SM 模型，集成调度

ABSTRACT

By detailedly studying multitask types of real-time system, Real-time tasks are categorized 2-dimentionally by arrival pattern and deadline type among the various real-time scheduling agriothms,Deadline Monotonic Scheduling is researched and has become a backbone technique because of its static optimal quality and its advantages such as easy to implement in real applications compared to dynamic scheduling agriothms. a real-time scheduling model based on this technique (in this thesis refer to SM model).Among the SM model, the technique of dividing task set into Scheduletasks and Managebletasks. Uses the different dispatch strategy separately to them. This article studies is with emphasis: The duty collection division is ScheduableTaks and the ManageableTasks two sets thought, the maximized ScheduableTaks algorithm, sturdy and healthy time duty dispatch optimization algorithm, ManageableTaks processing method. Finally this article proposed regarding the SM model further improves direction.

Feng YanHong(Computer Applied Technology)

Directed by Prof. Zhang YuMing

KEY WORDS: real-time task scheduling, Deadline Monotonic Scheduling, SM model, Integrated scheduling

声　明

本人郑重声明：此处所提交的硕士学位论文《实时多任务集成调度算法的研究》，是本人在华北电力大学攻读硕士学位期间，在导师指导下进行的研究工作和取得的研究成果。据本人所知，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得华北电力大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：冯艳红　日　期：2005.12.30

关于学位论文使用授权的说明

本人完全了解华北电力大学有关保留、使用学位论文的规定，即：①学校有权保管、并向有关部门送交学位论文的原件与复印件；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同意学校可以用不同方式在不同媒体上发表、传播学位论文的全部或部分内容。

(涉密的学位论文在解密后遵守此规定)

作者签名：冯艳红　　导师签名：张东明

日　　期：2005.12.30　　日　　期：2005.12.30

第一章 绪论

1.1 论文的选题背景及意义

实时系统是指系统在接到数据的同时，能够在规定的时间内予以响应，以足够快的速度处理，及时将处理结果送往目的地的一种处理系统。

实时系统最早出现在二十世纪中叶，当时，IBM 为美国海军和空军开发了飞行模拟系统 Whirlwind 和防空系统 SAGE。这些系统对时限性的要求导致了实时系统的诞生。随后，在五十年代末期，实时系统开始进入工业过程控制领域。到了六十年代，美国开始了雄心勃勃的宇航计划，其中太空船的控制系统和遥感系统都要求具有相当好的实时性能。这为实时系统的发展提供了一个良机。在这个时期，实时系统应用的主体也开始从军用走向了民用。随着大量适于实时处理的电子器件的出现，实时系统的应用范围也在不断扩大[1]。目前，它已渗透到了社会生活的各个层面，无论是在国防应用还是在航空航天、核反应堆控制、电信行业、过程控制、敏捷制造、医药行业、信息家电等各类民用应用上都发挥了巨大的作用。Internet 的飞速发展以及各种基于 Web 的应用对于实时性能的大量需求，更是为实时系统的发展提供了广阔的前景。“实时”正成为一种无处不在的计算。

实时系统的正确性不仅依赖于计算的逻辑结果，而且还与计算结果产生的时间有关，如果任务没有在规定的时间内完成，计算结果不仅毫无意义，而且有时会引起严重的后果。实时系统最本质的特征是任务的实时性和可预测性。因此，实时系统的首要目标是满足系统内实时任务的时间约束，合理的调度并执行它们，使它们能在各自的任务时限（deadline）之前完成操作。这就使实时任务的调度成为实时系统的核心问题之一，它也是计算机科学领域研究的热点之一。

实时调度（Real_time Scheduling）是指在有限的系统资源（如，CPU 等）下，为一系列任务决定何时（以及在哪一个处理器上）运行，并分配任务运行所需的除 CPU 之外的资源，以及保证其时间约束、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两个必备特性——时限性和高可靠性的最重要手段之一，对单机系统而言，设计合适的任务调度算法是提高系统实时性能的关键所在。因此，任务调度算法一直都是实时系统研究的热点问题，在国内外学术界备受关注。

实时计算机系统常常区分为硬实时系统和软实时系统。硬实时系统是那些在规定的时限前做出响应是绝对强制性要求的系统。软实时系统是那些响应时间虽然重

要；但如果偶尔错过时限系统依然正常运行的系统。例如，战斗机的飞行控制系统是一个硬实时系统，因为错过时限可能导致一场大灾难，而过程控制应用的数据获取系统是软实时的，因为它可被定义为按固定的时间间隔对输入传感器采样，但容许间歇的延迟。一方面， 最初的实时系统通常都是硬实时系统，随着实时系统向各个领域的渗透，又出现了大量的软实时系统甚至是混合实时系统。所以，随着实时应用的日渐复杂，这种静态的硬实时系统将无法满足应用的需要。因此，随着混合实时系统的被广泛应用于航空航天，工业控制、电讯行业、图像处理以及 Internet 应用等诸多领域，使得适合于该系统的调度算法的研究引起了学者们的广泛关注。然而，目前的调度算法大都是针对单一的硬实时系统提出的，对于混合实时调度算法研究则比较薄弱。

另一方面，实时系统中的任务可以分为周期性的任务、间发任务（sroradic task）和非周期性的任务(aperiodic task)三类。这 3 类任务可能会同时存在于同一个实时系统中。其中，周期性任务是系统要处理的主要任务，由传感器数据或控制回路触发，相邻两次触发之间有固定的时间间隔，此类任务占用处理机的时间较多；间发任务是系统为了处理一些意外情况或紧急事件所需要执行的任务，如果考虑作业到达间隔的最坏情形，它可并入周期任务一类；而非周期性任务，与间发任务不同，在同一时间，不是只有一个任务要求获得处理机，而是可能会有多个非周期任务同时要求处理机去执行，此类任务占用处理机时间较少而且数目也较少。这给系统调度机制带来了新的需求和挑战，因此，对于这几类任务的集成调度是在保证周期任务的时间约束条件的基础上，再考虑如何调度非周期任务。在实时系统中多任务的调度已引起了学术界的广泛关注。

此外，对于实时任务的调度算法可以分为静态调度算法和动态调度算法。动态调度算法虽然在某些性能上要优于静态调度算法，但静态调度算法比动态调度算法的额外开销小，稳定性和可预测性好，分析任务的时间行为较为容易，便于实现，所以得到了广泛应用。在诸多的静态调度算法中，对截止时间单调调度算法进行了详细的研究，它已被证明是最优的静态调度算法，所以本论文采用截止时间单调调度算法。

由此可见，无论从理论研究还是实际应用的角度，实时多任务的集成调度算法研究都具有重大意义，一直是一个具有挑战性的课题，具有一定的学术意义和实用价值。

1.2 国内外发展动向

实时系统是一种带有时间约束特性的多任务的复杂的计算机控制系统，其应用范围

涉及到许多重要的应用领域，如：航空航天、先进制造业、铁路及机场管理调度、核电站及化工过程监控等。随着计算机应用向高度（高性能）、广度（网络化、嵌入式）和深度（智能化）方向的不断发展，有时间约束的应用系统需求越来越突出，如媒体播放器、视频会议系统、复杂工业控制、银行自动取款机、网上电子商务、QoS(Quality of Service)约束的网络服务、家庭信息网络的互连接等。其理论及应用研究涉及到计算机软硬件设计理论、操作系统理论及各个应用领域所包含的特殊原理、方法与技术，属典型的边缘交叉学科。

在实时系统的不断发展过程中，单机系统的调度理论逐渐成熟，调度技术也在不断丰富和发展，其中对于混合实时系统的任务调度问题的研究，许多学者为此做出了不懈的努力，并且也取得了不少的成果。传统上解决实时系统中的多任务混合调度问题有两大类方法：一种是采用一个高优先级的周期性的轮询任务来为非周期任务服务，它的缺点是不管有没有非周期任务的请求，系统总是要浪费一段时间来为非周期任务服务。另一种是在出现非周期任务请求时，以满足周期任务的最后期限为前提，尽可能地推迟周期任务的执行，从而挪出一段时间（或称可延缓时间）供非周期任务使用。这种方法使非周期任务获得较理想的响应时间，但由于要对任务的可延缓时间表进行维护，增加了许多不必要的开销。

对于在这些问题上所造成的资源浪费等问题，国内有很多技术文献提出了改进办法。

文献[1]采用的方法是对于周期任务采用 DMS 算法，当有非周期任务请求时，引入 DD 算法，改变任务原来的优先级，然后对所有周期任务和非周期任务进行调度。由于 DD 算法使得处理机潜在的利用率达到 1，因此非周期任务可以从周期任务那里获得比采用静态调度算法更多的可挪用时间，从而达到改善非周期任务的响应时间的目的，同时又不影响周期任务的执行。

文献[2]引入了一种称为加入 DD 策略的软非周期任务调度算法，简称为 ODD 算法。ODD 算法事先并不计算每个周期任务的每个作业可“被挪用”多少时间，当非周期任务请求执行时，它首先考虑，在使用 RM 策略的情况下，若将这个非周期任务执行完，是否会影响周期任务的执行，如果不会，则可将非周期任务执行完再按 RM 策略调度周期任务；如果会，则计算当在未来的某个时间段内使用 DD 策略调度周期任务时，最多可挪用出多少时间供非周期任务使用。

文献[3]对 RMS 算法进行了改进，使之能够应用于周期任务和非周期任务共存的实时系统。周期任务按照周期大小设置优先级，将非周期任务看成是以平均到达时间间隔 $1/\lambda$ 为周期的周期任务。

文献[4]对于实时多任务的调度方法采用可抢占的 RMS 算法，并且带有非周期任务的服务器。任务的调度方式为：(1)设置非周期服务器 C，服务器的周期为非周

期任务的最小到达时间间隔，容量为非周期任务实例的最长执行时间。(2)优先级低于非周期服务器的周期性任务实例到来后，若没有非周期任务实例到达、执行或在队列中，即非周期服务器空闲，则先执行周期任务实例，并把非周期服务器的容量寄存在该周期性任务中。(3)如果在一个非周期服务器的服务周期内没有非周期任务实例产生，而且已经将非周期服务器的容量寄存在周期性任务中，则清除已寄存在周期性任务中的容量，否则，非周期服务器重新计时。(4)当非周期性任务实例到达后，首先查看非周期服务器的容量是否已经被使用，如果没有则根据非周期服务器的优先级运行非周期任务实例；否则在寄存非周期服务器容量的周期任务中立即执行。

另外中国科学院软件研究所互联网软件技术实验室正在研究的内容包括《实时系统的开放性混成调度机制研究》，该项目基于越来越突出的多类实时任务并存，以及实时、非实时任务并发的混合复杂调度应用需求，研究基于 Linux 的、可支持多类调度策略同时并存的、具有开放结构的混合调度机制等实时多任务操作系统核心技术。主要研究解决多类型调度策略的有机集成、开放性的混合调度框架、自适应等实时操作系统调度策略，研究前沿关键技术及基于该理论架构的 Linux 内核实时改造技术。

在国外的文献中，也有很多对于此类相关问题的讨论研究。包括实时系统中非周期任务调度算法的改进，截止时间单调算法在实时多任务系统中的应用等。

综合上述国内外研究情况可以发现：由于实时系统中多任务的调度问题相对比较复杂，国内外对于此课题的研究还很不成熟，因此本课题根据当前实时系统的发展要求，针对实际应用中大量的硬实时、软实时多任务，对实时系统的调度理论与实践等问题进行了深入的研究。

1.3 研究的主题及达到的水平和目标

本论文是在基于实时系统中多类型任务调度的研究基础上撰写而成的。实时系统多任务调度主要包括以下几个方面，也就是本文所要完成的主要工作：

1.研究、分析了实时系统中所存在的各种实时任务，对于实时任务按照到达模式和截止时间类型两个维度进行了分类。

2.研究任务的驱动方式及任务就绪队列的建立方法。

3.实现将非周期任务向周期任务映像。

4.研究各种实时任务调度算法特别是截止时间单调算法。

5.提出了一个 SM 模型，将任务划分为可调度性能够保证的 scheduletasks 和可调度性不能够保证的 managabletasks 两个集合。

6.实现硬实时任务优先级优化算法，非周期任务向周期任务映射算法，最大化scheduletasks 算法等。

对于实时应用系统涉及混合任务集的调度处理，由于任务的时间性需求不同，导致系统在调度处理这些任务时的策略与目标也是不同的。具有混合任务集的实时调度算法的目标是：①保证硬截止期任务满足截止期；②最大化固截止期任务的完成数量；③最小化软截止期任务的平均响应时间。

1.4 实时程序设计语言应具有的特点及使用的工具

实时系统中软件最终是用程序设计语言编写出来的，为了更好的支持实时系统，实时程序设计语言就需要 支持相应的实时特性，并需要相应的程序设计工具对它进行支持。

同时对于实时任务基于优先级的调度的实现，很少有程序设计语言明确定义优先级作为其并发设施的一部分。它们常常只提供初步的模型。常用的实时设计语言有 Ada 语言，POSIX 设施以及实时 java.

本文选用类 C++语言来实现简单的算法流程。

1.5 论文的组织结构

论文共分六章，其组织结构为:

第一章是绪论。首先，提出了论文的选题背景及意义，包括实时系统的起源与发展，实时调度算法的研究历史以及本论文的动机。然后介绍了该研究领域的国内外发展动向。接着指出了论文研究的主题及达到的水平和目标。同时，还指出了论文所做的创新性工作以及实时程序设计语言应具有的特点及使用的工具。

第二章是对实时调度算法研究的综述。本章介绍了实时调度的基础知识，并阐述了实时多任务调度算法研究中的两大热点问题—实时多任务的静态调度算法和动态调度算法的研究现状。在此基础上，我们详细分析了当前实时多任务静态调度算法中的截止期单调算法的性能和特点，以及进行了充分且必要的可调度性测试方法分析，进而提出了本论文所要解决的问题。

第三章是对实时多任务集成调度框架的研究。首先，给出了我们所研究的实时多任务的系统模型。并且对实时任务按照到达模式和截止时间两个纬度进行了系统的分类。在此基础上，确定了一个新的任务模型——SM 模型，并且提出了对于实时任务调度器的设想。

第四章在第三章工作的基础上，提出了实时多任务系统的可适应性集成调度算

法，给出了集成调度器的体系结构，并且对于调度策略分派器进行了详细的分析和讨论。

第五章对论文工作进行了总结，并提出了进一步的工作。

第二章 实时调度理论研究

实时调度是实时系统研究中最为活跃的领域之一。本章是对实时调度算法研究的综述。首先，我们介绍了实时调度中的基本概念，包括任务的定义及其特性、任务间的相关性；其次，我们又介绍了实时调度算法的分类、常用的实时调度策略以及实时调度算法的性能评估标准。此外，我们对当前实时多任务调度算法的研究现状做了详细的分析和介绍，指出了实时多任务调度算法的一些不足，进而提出了本论文所要解决的问题。

2.1 实时调度的基础知识

2.1.1 任务调度

任务调度就是按某种算法动态地把处理机分配给就绪队列中一个任务并使之执行。任务调度问题的关键是对调度性能有决定性影响的调度算法，调度算法的好坏将直接影响系统的性能。

2.1.1.1 任务调度算法

从实时性考虑，为了保证紧急重要的任务及时运行，实时操作系统的任务调度一般采用可剥夺的优先级调度算法，即在调度时把处理机分配给优先级最高的就绪任务。而且，当被唤醒或被激活任务的优先级高于现行任务的优先级时，可立即剥夺现行任务的处理机，以保证紧急事件的及时处理。如果允许不同任务具有相同的优先级，就绪队列中优先级相同任务的调度可采用先来先服务或按时间片轮转的调度算法。由于任务调度的执行频率最高，因此，不宜采用复杂的调度算法，否则会增加任务的切换时间，影响实时性。

2.1.1.2 就绪队列的插入操作

当把一个任务插入就绪队列时，从就绪队列首指针开始，用被插入任务的优先级依次与就绪队列中就绪任务的优先级比较，当插入任务的优先级第一次出现大于就绪队列任务的优先级时，即将被插入的任务插入到该任务的前边（设优先数越大，优先级越高）。插入操作应设计成一个子程序，供操作系统的其他程序调用。

2.1.1.3 任务调度程序

任务调度程序执行具体的处理机分配功能，那么，在什么情况下需要启动任务

调度程序进行处理机分配以及如何进行分配，这是在代码设计之前就应考虑和确定的问题。

(1)任务调度的时机

概括的说，当现行任务因某种原因放弃处理机的控制权而离开执行状态时，都会引起调度程序的执行。实现时可设一调度标志（设为 DPH），任务放弃处理机控制权时将其置位，当决定是否要重新调度时，可据此标志的状态确定是转调度程序还是让当前任务继续。调度时机可有下述几种情况：

1)现行任务调用等待、延迟系统调用命令或调用挂起系统调用命令将自己挂起，主动放弃处理机的控制权而暂停。

2）现行任务调用接收数据系统调用命令接收数据时，没有数据可接收，需要等数据到来；或申请缓冲区而无可用空缓冲区。

3）现行任务调用输入输出系统调用命令需要等 I/O 的完成。

4）当暂停的任务被唤醒或激活，因优先级高于现行任务而需要剥夺其处理机。

(2)任务调度程序

鉴于任务调度程序的运行频率最高，因此，任务调度程序的代码必须进行精心设计。由于任务已经按优先级在就绪队列中排队，任务调度过程就比较简单。调度过程为：

首先判断就绪队列是否空。若就绪队列空，则将处理机状态置为空闲状态后而动态等待任务的就绪；若不空，则根据就绪队列指针从队首取任务的标志符 ID 送现行指针，修改就绪队列指针使其指向下一个就绪任务。据任务标识符 ID 找到其现场保护区并恢复现场后转去执行被调度的任务。任务调度程序处理流程如图 2-1 所示。

需要说明的是：

1）任务调度一般可在关中断情况下执行。在转任务执行前应开中断。

2）任务状态的变化通常情况下都与中断相关联，因此，当就绪队列空时，必须开中断，否则，等待的任务就永远不能变为就绪，就绪队列永远是空的。

2.1.2 实时相关概念

2.1.2.1 任务及其特性

任务是指完成某一特定功能的软件实体，它是实时调度中的基本单位。从调度的角度来看，一个实时任务具有如下基本特性：

• 任务到达时间：即任务被启动并准备执行的时间；

• 任务最大执行时间：即任务在最坏情况下无中断执行所需的处理器时间；

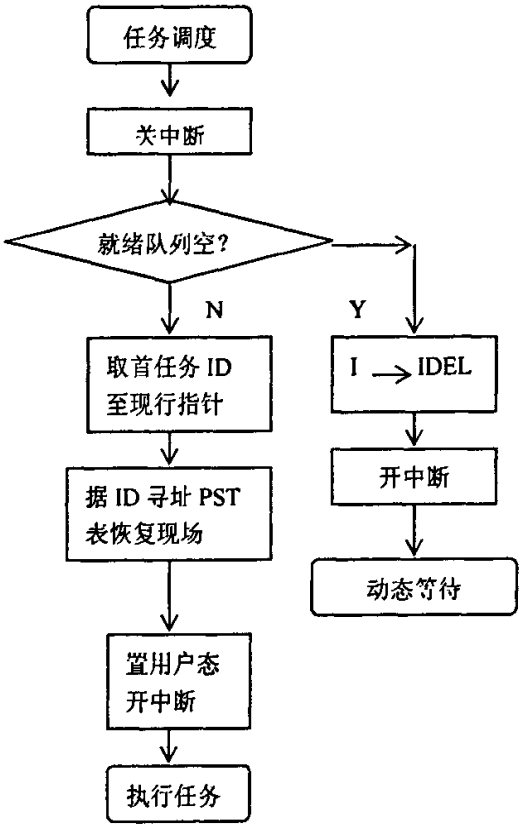


图 2-1 任务调度程序处理流程

• 任务的最后期限：即任务在这个时间应该完成执行并且产生一个有价值的结果；

• 任务的周期：即周期任务两次执行的时间间隔；

• 任务的截止期：即任务在这个时间应该完成执行并产生一个有价值的结果；

• 任务的最长响应时间：对于任意一个任务 T_i，从任务 T_i 实例的到达时刻，到该任务被执行完毕所需要的最长时间间隔。

根据任务到达频率的不同，任务可以被分为周期性任务和非周期性任务。对于周期性任务，其相邻两次到达时间的间隔是一个固定的常数（即周期）。这样，一旦获得了任务的首次到达时间，它今后的每次到达时间用首次到达时间加上周期的若干倍数即可得到。非周期任务的相邻两次到达时间的间隔是任意的，没有任何限制。同时，这类任务的到达时间是不可预测的，且其特性在任务到达前是不知道的。

此外，任务还具有其它一些与调度相关的特性：

• 等待时间：这是任务为了取得运行权而需要等待的时间。它是由于其它更高级别任务的抢战或由于该任务要访问一些共享资源而被阻塞所造成的。

• 转向时间：这是指任务从就绪到完成的时间。如果忽略开销，这一时间为任务运行时间与等待时间之和。

同时，根据任务重要性的不同，还可以把实时任务分为硬实时任务和软实时任务两种。硬实时任务具有硬时间约束，其截止期的错过将会导致严重的后果。软实时任务则具有软时间约束，其截止期的错过在某种程度上是可以被接受的，不会引起非常严重的后果。

2.1.2.2 任务间的相关性

任务间的相关性有以下几种：

- 顺序相关性：这是指某些任务之间必须按照一定的先后顺序开始运行。它是由与应用相关的处理顺序、任务间交换数据的需要或任务间通讯而导致的 。如果任务间不存在顺序相关性，则称这些任务是独立的。
- 资源相关性：它包括任务互斥地使用临界资源以及任务共享使用某些资源。
- 优先级顺序：高优先级的任务可以抢占低优先级任务的运行。
- 与策略相关的约束：主要是指任务的可抢占性，而且是由应用的性质所决定的。对于可抢占任务，如果必要，它在任何时刻都可被其它任务中断；对于非抢占任务，一旦他开始运行，则在它完成之前都不能被中断。

2.1.3 任务分配法则

默认的任务分配法则如下。每个任务都有一个基本优先级和一个活动优先级。基本优先级由程序员指定或者系统默认，任务的活动优先级是基本优先级和被继承的任何活动优先级中的最大值。

每个处理器都有一系列与其相关的处于等待状态的队列。每个优先类别有一个队列。当一个任务为执行做好准备时，它被放置在其活动优先级相关的队列的尾部。如果一个任务被放置在一个准备好执行的队列中时，其活动优先级改变，那么该任务则根据其新的活动优先级被移至新队列尾部。如果任务的基本优先级改变，那么任务也会根据活动优先级被转移至队列的尾部，无论其活动优先级发不发生改变，该任务是不是正处于执行中这种情况都会发生。当运行中的程序被延迟（例如等待一个集合）或者已经完成，那么具有最高优先级的非空队列中的排在最前的任务被执行。

2.1.4 实时调度算法的分类

调度的实质是对资源的分配，而实时任务的调度强调的是任务的时间约束。实时调度的分类方式可以有多种。常见的分类方法如图 2-2 所示：

分类标准	分类结果
根据所解决问题的性质	静态调度、动态调度
根据任务是否可以被抢占	抢占式调度、非抢占式调度
根据任务的实时性要求	硬实时调度、软实时调度
根据实时系统运行环境	单处理机调度、多处理机调度
根据调度器的体系结构	集中式调度、分布式调度

图 2-2 实时调度的分类方式

• 静态调度和动态调度

若调度算法是在编译的时候就做出决定从就绪任务队列中选择哪个任务来运行，则这样的调度是静态的。这类调度算法假设系统中实时任务的特性是事先知道的，它脱机地进行可调度性分析，并产生一个调度表，调度器在运行时将依照这张表所提供的信息选择任务来运行，静态调度适合于问题需求比较确定的情况，如工业过程控制，静态调度算法的优点是运行开销小，可预测性强，一旦找到了一个调度解决方案，便能够保证所有任务的截止期都可以被满足。但是，由于静态调度算法一旦做出调度决定后在运行期间就不能再改变，所以它的灵活性较差，不适合不可预测环境下的调度。

如果调度器是在运行期间才决定就绪队列中的哪个任务来运行，则这样的调度是动态的。这种调度只是考虑当前就绪任务集中的任务特性，以此来确定当前的任务调度序列，对于未来将要到达的任务的特性一无所知。动态调度算法需要对变化的环境能够及时做出反应。因此，这类调度算法的优点是比较灵活，适合于任务的不断生成，且在任务生成前其特性并不清楚的动态实时系统。但是，动态调度算法的可预测性差且运行开销较大。

• 抢占式调度和非抢占式调度

在抢占式调度中，目前正在运行的任务可以被别的更紧迫和更重要的任务中断。同时，被抢占的任务在未来可以恢复运行，且不会影响到任务的整体时限约束。这种抢占式的调度常见于工业制造中，它的优点是比较灵活，对资源的利用率较高；但是由于经常出现的上下文切换使得系统开销较大。

如果在调度过程中不允许正在运行的任务被别的任务中断，任务一旦占有了处理器便会一直运行下去直至任务完成，这样的调度则是非抢占式的，它比较适合于任务运行时间都比较短的系统。其优点是省去了进行上下文切换的开销，同时具有更好的可预测性，更易于测试，在保证对共享资源的互斥访问上也有较强的集成能

在抢占和非抢占两个极端之间，有一个可供选择的策略，那就是允许较低优先级的进程继续运行一段限定的时间（但不一定能够完成）。这种方案被称为延期抢占或合作分派。

• 硬实时调度和软实时调度

在硬实时调度中任务必须在其截止期限(deadline)内执行完毕,否则将产生严重后果。而对于软实时任务,当系统负载过重的时候,允许发生错过截止期限的情况。

• 单处理器实时调度和多处理器实时调度

如果实时系统运行在一个单独的处理器上，则为实时单处理器系统。在这种系统中对实时任务进行调度时，仅需要考虑任务在这个处理器上的运行情况。如果实时系统运行在一个多处理器的环境下，则为实时多处理器系统，此时在进行实时调度时，不仅要考虑任务在单个处理器上的运行情况，还要考虑任务被分配到哪个处理器上运行，即任务的分配问题。

• 集中式调度和分布式调度

这种分类是针对实时多处理器系统的调度而进行的。在集中式调度中，系统中有一个专门的处理器作为调度器，所有的任务都到达这个中心调度器，由它来做统一的调度后，这些任务再从这里分配到其它处理器上去运行；在分布式调度中，每个节点上都是一个调度器，这些调度器在系统中的地位相同，它们可以各自做出调度决策，若任务在某个节点上无法被调度，则可以将该任务迁移到其它可以保证其截止期的节点上去运行。

2.1.5 实时调度策略

不同的操作系统和硬件实现方式造成任务调度所需要的时间开销不同。因此,操作系统的实现方式对任务可调度性影响很大。根据驱动任务调度发生方式,实时系统调度可分为中断事件驱动调度，时钟驱动调度，加权循环策略，优先级驱动策略，数据驱动调度，事件驱动调度[1]。

◆ 中断事件驱动调度策略

中断事件驱动的实现方式是依赖一个外部的硬件设备,通过产生中断的方式来为任务周期提供信号。这种调度方式又可细分为两种：集成中断事件驱动调度和非集成中断事件驱动调度。在集成中断事件驱动的调度系统中,每一个任务周期都会有一个中断被发布给处理器,中断的优先级与任务的优先级相对应,并且该中断只有在其优先级高于正在执行的任务时才会被处理器响应。所谓集成的含义就是,中断的优先级是取决于其对应任务的优先级的。在非集成中断事件驱动的调度系统中,任务都是通过外部中断启动的。每一个新任务到来时,当前任务都会被中断,即中断的优先级与相关的到达任务的优先级是没有关系的。如果到达的新任务优先级高于正在执

行的任务,那么新任务抢占正在执行的任务,否则不发生抢占,但是即使没有发生抢占,中断仍然会发生,并且调度器也会运行。

◆ 时钟驱动调度策略

时钟驱动调度策略的实现方式是利用一个可编程的时钟发出的周期性时钟中断来控制调度器的运行,并且调度器维护一个内部的时间值来计算任务周期和决定何时调用这些任务,这种实现方式可以继续被细分为时钟驱动调度和有计数器的时钟驱动调度。时钟驱动调度使用一个周期时钟发出的中断来中断系统运行并调用调度器来更新系统时间,并在需要时重新调度。有计数器的时钟驱动调度维护一个计数器用来限制调度点的数量。在每一个调度点,计数器的值被初始化为到达下一个任务截止期限的时钟周期数。当每一次时钟中断发生时,计数器的值都会递减;当计数器的值减为0时,调度器被触发。

此种驱动又可以分为以下三种类型:

➢ 延迟驱动。延迟驱动是使实时任务暂停一指定时间间隔（即延迟时间）后再驱动执行的一种驱动方式。本方式适用于对周期时间要求不太严格的周期任务，执行周期等于实时任务的执行时间与延迟时间之和，由于实时任务执行时间的长短受系统当时状态的制约，因此，执行周期亦不固定。延迟驱动方式要求实时任务在本次执行结束时设置好下次执行的间隔时间，然后该任务便去睡眠（等待），等间隔时间到，即再次被驱动执行。

➢ 绝对时间驱动。这种驱动方式是指只要实时任务在本次执行时将下次驱动执行的时刻设置好，一旦定时钟计时到指定的时刻，该任务即被驱动。

➢ 定时驱动。类似于上述的延迟驱动，其区别在于延迟驱动设置的时间间隔不包括任务的执行时间，而定时驱动设置的时间间隔包括任务的执行时间。该驱动方式适合于对周期时间要求严格的周期任务。定时驱动的特点是：①驱动的时间间隔一次设定始终有效；②两次驱动之间的时间间隔保持不变；③周期任务的执行时间应远远小于该任务的周期；④为了使这类任务驱动后能立即获得处理机的控制权即尽快运行一遍，可赋予较高的调度优先级或被驱动后不经任何调度而直接进入执行状态。

◆ 加权循环策略

循环方法是针对分时应用的一个传统调度策略。在这种方法中，任务在就绪后便进入一个先进先出(First-in-first-out, FIFO)队列。从队列头开始，轮流运行队列中的任务，且每个任务都至多只占有一个时间片。若一个任务在其所占有的时间片结束时还没有完成，则该任务将被放置到队列的末尾，以等待下一次的运行。

加权循环策略是建立在循环方法基础上的。但是，该策略并不是让所有任务来

平等的分享处理器的时间，而是给不同的任务以不同的权值(权值指的是分配给任务的处理器时间的比例)，这样，任务将按照权值来占有时间片。

◆ 优先级驱动策略

优先级驱动策略的特点是，从不故意让任何的资源空闲，也就是说，只有需要某资源的任务全部未就绪时，该资源才会是空闲的。在这种调度策略下，调度决策通常是在任务到达或完成时进行，因此，优先级驱动的调度策略是事件驱动的。

优先级驱动策略总是试图达到局部最优。当某个处理器或资源为空闲的，而此时需要这些资源的任务又是就绪的，则应让这个任务运行，而不应让它等待。同时，优先级策略通常要为任务分配一个优先级。那些就绪任务被放置在一个或多个任务队列中，而这些队列中的任务则是按照优先级的顺序来排列的。在做调度决策时，总是选择一个队列中具有最高优先级的任务来运行。

◆ 数据驱动调度策略

数据驱动是指实时任务是否获得所需之数据作为运行条件。例如数据处理任务可采取此种驱动方式。假设采样任务与数据处理任务之间以数据通信方式进行数据交换，显然，当数据处理任务接收不到采样任务的数据时只好等待，一旦采样任务把实时数据采集进来，便可将数据发送给数据处理任务，并立即驱动数据处理任务运行来处理这些数据。

◆ 事件驱动调度策略

“事件”是指一种情况或一种活动的发生或一种指定状态的变化。在操作系统中，一般将事件抽象成一个数据或一种数据结构。如果一个任务以事件出现作为运行条件，事件一旦产生，事件的发出者就要将等待该事件的任务唤醒并投入运行，处理完事件所要求的操作后又继续等待下一个事件的出现。报警管理任务、数据库管理任务、操作控制台命令处理任务等可采用此种驱动。

2.1.6 实时调度算法的性能评估标准

实时调度算法有多种性能评估标准，其中最常见的有：

• 调度成功率：指被某算法调度成功的任务集数目与参加调度的可调度任务集数目的比值；或指被算法调度成功的任务个数与参加调度的任务数之比。

• 调度长度：即将参加调度的所有任务都执行完所花费的时间。

• 处理器利用率：是指处理器在运行时，处理任务所用的时间占总时间的比例。

• 吞吐量：即单位时间内处理任务的个数。

• 可调度性：是实时系统调度理论研究中的核心问题，这是因为，实时任务具有时限要求，在一个或多个处理器之间调度实时任务，需要判断是否每个任务的执行都能够在

其截止期限内完成。如果每个任务的执行都能够在其截止期限内完成,则称该调度是可行的。可调度性判定(或称调度可行性判定)就是判定给定的n个实时任务在应用某种调度算法的前提下能否产生一个可行的调度。调度算法的设计要尽可能满足任务可调度性的要求。

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

2.2 实时多任务调度算法研究

调度的实质是资源分配,而实时任务的调度强调的是任务的时间约束。任务调度是实时多任务程序的核心,它的主要功能是从当前众多可运行的任务中提取最高优先级的可运行任务,找到该任务的TCB地址,从该TCB中取出上次被中断运行的现场,从而将该任务投入运行。

通常调度方案应提供两个功能:

- 一个安排系统资源(特别是CPU)使用顺序的算法
- 一个在应用调度算法时预测最坏情况系统行为的手段

实时任务调度算法中的静态调度算法(如果预测是在执行前进行的)和动态调度算法(使用运行时决策)是实时任务调度研究中的热点问题。本文采用静态方案,主要关注基于优先级的抢占方案。这里,任务被分配了优先级,任何时刻都是执行最高优先级的任务(如果它没有被延时或挂起)。所以一个调度方案应该包括优先级分配算法和可调度性测试两部分。

2.2.1 实时多任务静态调度算法综述

静态调度算法是指事先为每个任务赋予一个优先级,该任务作业流中的每一项作业都具有相同的优先级,然后根据优先级大小调度任务。主要算法有先进先出(First in First out, FIFO)、速率单调调度算法(Rate-monotonic Scheduling, RMS)、截止时间单调调度算法(Deadline monotonic Scheduling, DMS)等。

2.2.1.1 速率单调调度

RMS是C.L. Liu 和J.W. Layland 在1973年提出来的静态调度算法,它是针对那些响应和处理周期性事件的实时任务。该算法按照任务的周期长短分配优先级,

周期越短，优先级越高。它可使周期任务负载的最小上确界（Least Upper Bound）达到 $n(2^{1/2}-1)$(n 是周期任务数)；它具有算法简单、可预测性强、开销小、灵活性好的特点，是目前实时调度的基础理论。即使系统出现瞬时过载，也完全可以预测哪些任务丢失时限。该算法的缺点是处理机的利用率很低，当 n 很大时，处理机利用率界限为 ln2=0.69。此外，它以任务周期来确定任务的优先级，但在实时系统中，有些事件是随机的，没有一个量化的方法来确定随机激发任务的优先级。这种，只有参照任务的紧急度，确定它的优先级。

2.2.1.2 截止时间单调调度

截止时间单调调度(Deadline Monotonic Scheduling)是Leng于1982年提出来的与速率单调调度相类似的调度策略。这种策略以截止时间的反序来分配任务的优先级，任务的截止时间越短，分配的优先级就越高。

截止时间调度具有与速率单调算法相同的优点。运行开销小，可调度性测试简单等。更重要的是，截止时间算法中放松了对任务的周期必须等于其截止时间的限制。在任务的截止时间小于或等于其周期的情况下，截止时间单调调度已被证明是静态最优的调度算法。速率单调调度和截止时间调度都已发展成为了较为成熟的理论体系。由于它们的调度性能和易实现性都较好，被广泛的应用于实时系统中，是实时调度的支柱。本文将在SM模型调度器的设计中应用截止时间单调调度算法。

2.2.2 实时任务动态调度算法综述

动态调度算法不仅可对任务作业流中的不同作业赋予不同的优先级，而且可在作业调度执行过程中改变作业的优先级，从而使任务系统的总体调度趋于优化，这种调度主要有：最早任务期限最优先（Earliest-deadline First, EDF）、最小紧急度优先(Least-laxity First, LLF)、最短处理时间优先(Shortest Processing Time First, SPTF)等。

2.2.2.1 最早截止时间优先

最早截止时间优先 EDF 算法思想：在调度时刻，任务的优先级根据任务的截止期进行动态的分配，截止期越早，其优先级也就越高。(对于周期任务，其截止期即为下一周期开始的时间)。EDF 调度算法处理机的利用率最大可达 100%，但是若出现瞬时过载时（实时任务要求的计算量超过可用的 CPU 时间），系统的行为将不可预测，可能会发生多米诺骨牌现象，即一个任务丢失时可能会引起一连串的任务接连丢失，所以，它的在线调度开销也要比 RMS 大。同时，如果某任务在当前时刻开始使用资源，已经无法满足其实现的要求，EDF 算法仍会做出一个无益的的决定，将资源使用权赋给该任务。

2.2.2.2 最小紧急度优先

最小紧急度优先算法(LLF: Least Leisure Fisrt)算法LLF是指在调度时刻，任务的优先级根据任务的空闲时间动态分配。空闲时间越短，优先级越高。其中空闲时间=截至时间-任务剩余执行时间。

理论上，EDF和LLF都是动态最优的调度算法。它们对于CPU使用率小于或等于1的任务即都是可调度的。但由于EDF和LLF在每个调度时刻都要计算任务的截至时间或空闲时间，并根据结果改变任务的优先级，因此开销大，不易实现。另外，EDF和LLF在过载的情况下都是很危险的，实际应用时必须有相应的过载保护机制与之相配合。

虽然动态调度算法具有更大的适应性(静态调度算法可视为动态调度算法的一个特例)，在理论上也能达到更好的性能，但由于算法本身的复杂性导致的运行开销和实现难度，其应用受到的极大的限制。与之相反，静态调度理论则因为相对简单，应用前景广泛而得到了深入的研究。

2.3 问题的提出

随着实时应用的日趋复杂，以及人们对于各种实时调度算法的深入研究，在此基础上不断有新的算法提出，用于解决各种实时调度问题，如实时任务同步、瞬间超载（实时任务要求的计算量超过可用的 CPU 时间）以及周期任务和非周期任务的混合调度问题等。周期、非周期混合调度问题之所以重要，是因为许多实时系统中都既包含硬周期任务又包含软非周期任务。解决这个问题的出发点是在保证不影响周期任务调度的同时，尽早完成非周期任务。因此，实时多任务调度算法一直以来都是实时调度研究中的一个热点问题。

传统上解决混合任务调度问题的方法有三种：

①后台执行法，即在一个基于优先级的方案中，调度非周期任务的一个简单方法是使这些任务运行的优先级低于硬实时任务的优先级。这样，非周期任务作为后台活动运行，因此，在抢占式系统中，它不能窃用硬进程的资源。虽然这是一个安全的方案，但是它没有对软实时任务提供足够的支持—如果软实时任务只作为后台进程运行，它们常常错过它们的时限。为了改善软实时任务的这种状况，可以利用②。方法。

②轮询法，即用一个高优先级的轮询任务（或称为服务器）周期性地为非周期任务服务。轮询法大大优于后台执行法，因为服务器每过一个固定时间间隔，就为非周期任务提供了一段高优先级的可用执行时间（我们称之为带宽）。但是，如果服务器周期开始时没有非周期任务到达，带宽将被丢弃，因此，若一个非周期任务恰在

带宽丢弃后到达，则它将或者等到下一个轮询周期开始时才能运行，或者就在后台执行 。

③空闲时间法，基于空闲时间的算法主要包括空闲时间偷取算法(slack stealing algorithm)、时间片移位算法(slot shifting algorithm)与双重优先级算法(dual priority algorithm)，这些算法都是通过离线或者在线分析从周期任务调度的空隙获得尽可能多的处理时间来处理非定期任务.

为了克服轮询法的缺陷，J.P.Lehoczky 等人提出了优先级交换 PE（priority exchange）算法和可延期服务器 DS（deferrable server）算法[23] ,Sprunt 稍后又提出了间发服务器 SS（sporadic server） 算法[24]。

在 DS 中，采用一种分析方法（例如，使用响应时间方法），引进一个新的、在最高优先级的进程。这个进程，也就是服务器，有一个周期 T_s和一个容量 C_s。选择这些值使系统中的所有硬实时任务仍然是可调度的，即使服务器以周期 T_s和执行时间 C_s周期性的执行也是如此。在运行时，当一个非周期进程到达时，如果服务器有可用容量，他就立即开始执行，直到它完成或服务器容量耗尽。在后一种情况，非周期任务被挂起（或转换为后台优先级）。在 DS 中，服务器容量每 T_s个时间单位被补充一次。

SS 的操作不同于 DS 之处在于它的容量补充策略。在 SS 中，如果一个任务在时间 t 到达，并要求使用容量 c,那么服务器在时间 t 后的 T_s个时间单位补充容量 c。通常，SS 能提供比 DS 高的容量，但是它增加了实现开销。

因为所有服务器限制了非周期软实时任务可用的容量，它们也能保证偶发任务不会执行的比预期的更频繁。一个时间间隔为 T_i和最坏情况执行时间为 C_i的偶发任务,如果它不直接实现为一个任务,而是经由服务器实现,该服务器的 $T_s=T_i$、$C_s=C_i$，那么这个偶发任务对较低优先级任务的影响就被限制了，即使偶发任务到达得过快。

所有服务器(DS、SS 和其他)都可以被描述为带宽预留(bandwidth preserving)，因为它们试图

- 使 CPU 资源对非周期任务立即可用（如果有容量）;
- 如果当前没有非周期任务，尽可能长地保留容量（通过允许硬实时任务执行）。

另一个比服务器技术性能更好的带宽预留方案是双重优先级调度（dual-priority scheduling）。优先级的范围被分为三档:高、中、低。所有非周期进程运行在中优先级。硬进程，当他们启动时，运行在低优先级，但是当他们要满足时限时，它们的优先级被及时提升到顶层。因此在执行的第一阶段，它们将为非周期活动让路（但如果没有这样的活动，它们就执行）。在第二阶段它们将转到

较高优先级，优先于非周期进程运行。在高优先级级别中，优先级根据时限单调方法来分配。实现双重优先调度方案需要使用动态优先级方法。

当服务器周期开始而没有非周期任务时，它们采用了各自不同的方法来保留非周期服务器的带宽。在某些情况下，这些算法可以达到很理想的效果，但在周期任务负载较大，非周期任务所需执行时间较多等情况下，它们的性能会急剧下降。

近年来，Lehoczky 等人提出了另一种解决混合任务调度问题的方案[25]，本文简称它为 OFP 算法，O 表示最优（Optimal），FP 表示固定优先级（Fixed-Priority）。与轮询法 PE, DS 和 SS 等方法不同，OFP 算法不使用服务器为非周期任务服务，而是在出现非周期任务时，以满足周期任务的最后期限为前提，尽可能的推迟周期任务的执行，从而“挪出”一段时间供非周期任务使用，我们将最多可挪用的时间称为可延缓时间。

OFP 算法事先算出在 0 到超周期（Hyperperiod，是所有周期任务周期的最小公倍数）时间内，每个周期任务的每个作业有多少可延缓（或可挪用）时间，并将这些值保存在一张表（可延缓时间表）中，在运行过程中，每个周期任务需要一个计数器，用于跟踪记录已处理的优先级高于或等于这个任务的周期作业量，另外还需一个计数器用于记录从 0 时刻到当前时刻非周期任务使用的总时间量。当非周期任务需要执行时间时，根据延缓时间表和非周期任务已占用的时间量等数值来计算当前最多可将所有周期任务延迟执行多少时间，并将这段时间分配给非周期任务。

OFP 是非周期任务的响应时间达到非常好的效果，它还可扩展，用于解决硬非周期任务调度问题。但它也存在一些缺点：(1)当超周期很大时，周期任务的可延缓时间表开销也会很大，而且即使在无非周期任务时，还需跟踪记录周期和非周期任务已完成的作业量，因此造成了不必要的运行开销；(2)OFP 是基于固定优先级的，文献[22]中指出，固定优先级方案下处理器的利用率要受到一定限制，因此，周期任务的可延缓时间也会受到限制。

2.4 充分且必要的可调度性测试方法

如前所述，由于实现动态调度策略的复杂性和缺乏操作系统支持，使之很难在实际的实时系统中得到应用。而静态调度策略中，速率单调算法和截止时间单调算法由于在一定条件下具有最优性，而且拥有确定的可调度性测试方法，在实际系统中得到了广泛的应用。

实时任务调度器也使用了截止时间单调算法作为周期性任务的优先级分配策略。下面给出截止时间单调算法的可调度性测试方法的证明。

为了能应用截止时间单调算法，我们必须找出一个准确的可调度性测试方法。

也就是说我们必须找出使用截止时间单调算法的任务集的可调性的充分必要条件。

我们采取的测试任务集可调度性的方案分为两个阶段。首先，使用一种分析方法去预测每个任务最坏情况下的响应时间。然后将这些值同任务的时限（截止时间）进行比较。

在讨论之前我们引入如下一些记号：

C_i：任务i的最坏情况下的执行时间。

D_i：任务i的截止时间，即从任务i被触发到完成的最大可接受的时间间隔。

其中$C_i \le D_i \le T_i$。

T_i：任务的周期。其中$D_i \le T_i$，对于非周期任务来说，T_i也可以理解为两个连续的任务请求之间的最小时间间隔。

B_i：任务i在最坏情况下由于临界资源的竞争而导致的阻塞。B_i通常等于低优先级任务中相关临界区的最大长度。

I_i：任务i在时间间隔$[T_i, T_i+R_i]$期间所经受的最大干扰，也就是在最坏情况下被抢占的时间。即任务i在就绪队列中等待高优先级任务完成的时间。最坏情况出现在所有更高优先级的任务同任务i同时启动的时候。

R_i：任务i在最坏情况下的响应时间，即从任务被触发到任务完成的时间。任务的可调度性可以由D_i和R_i来判断。如果$R_i \le D_i$，则该任务是可调度的。

hp(i)：任务集中优先级比任务i高的任务的集合，也就是可能对任务i造成抢占的任务的集合。

分析任务i的执行过程不难看出，任务i的最坏响应时间包括最坏CPU执行时间C_i，最坏的由于临界资源竞争导致的阻塞时间B_i，和高优先级任务对i的抢占导致的延迟I_i。因此有公式(2-1)成立：

$$R_i = C_i + B_i + I_i \qquad \text{公式(2-1)}$$

公式(2-1)中，任务的执行时间是预先给定的，或者可以根据具体情况计算得到。而阻塞时间B_i等于低优先级任务中相关临界区的最大长度。因此计算R_i的关键是计算I_i。

表2-1两个简单任务的响应时间计算

任务	执行时间C_i	周期T_i	阻塞B_i	优先级	响应时间R_i
τ_1	15ms	30ms	0ms	高	15ms
τ_2	15ms	75ms	0ms	低	30ms

考虑如表2-1所示的两个任务，为计算任务的最坏强占情况，不失一般性，我们假设任务τ_1和τ_2同时被触发，那么任务的执行过程如图2-3所示：

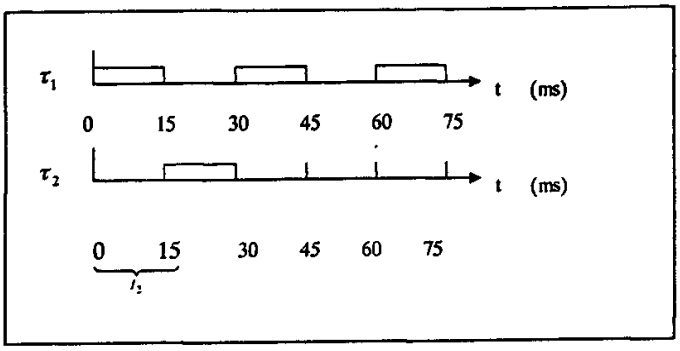


图 2-3 τ_1对τ_2的抢占情况

由图2-3可以看出，由于τ_1的优先级高于τ_2，没有任务会抢占它的执行，因此I_1=0，R_1=C_1+B_1+I_1=15ms。对于τ_2，最坏情况下会被τ_1抢占一次，I_2=C_1=15ms，因此R_2=C_2+B_2+I_2=30ms.现在我们将τ_2的执行时间增加1ms，那么任务的描述如表2-2所示，任务的执行情况如图2-4所示：

表2-2 τ_2执行时间增加1ms后的结果

任务	执行时间C_i	周期T_i	阻塞B_i	优先级	响应时间R_i
τ_1	15ms	30ms	0ms	高	15ms
τ_2	16ms	75ms	0ms	低	46ms

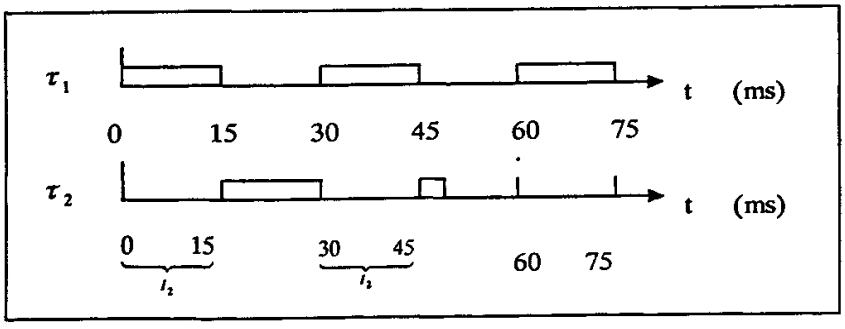


图 2-4 τ_2执行时间增加 1ms 后τ_1对τ_2的抢占情况

由图2-4所示，由于τ_2不能够在τ_1再次被触发之前完成，而被τ_1抢占了两次。结果是$I_2=2*C_1=2*15=30ms$；$R_2=C_2+B_2+I_2=46ms$.

以上的结果显示任务τ_i被任务τ_j所抢占而造成的延迟取决于在任务τ_i的响应时间内τ_j被触发的次数和τ_j的执行时间。设任务τ_i的响应时间R_i已知，则最坏情况下任务τ_j的触发次数可以使用一个高限函数来简单表达：

$$启动次数=\left\lceil \frac{R_i}{T_j} \right\rceil$$

高限函数（⌈ ⌉）给出大于分式的最小整数。比如，1/3的高限是1，6/5的高限是2。

最坏情况下由于τ_j强占而造成的延迟为$\lfloor R_i/T_j \rfloor * C_j$

将这个结论推广到一般情况有，所有高优先级任务造成的延迟为：

$$I_i=\sum_{\forall j\in hp(i)}\left\lceil \frac{R_i}{T_j} \right\rceil * C_j \qquad 公式（2-2）$$

其中hp(i)为优先级高于τ_i的集合。结合(2-1)和(2-2)有：

$$R_i=B_i+C_i+\sum_{\forall j\in hp(i)}\left\lceil \frac{R_i}{T_j} \right\rceil * C_j \qquad 公式（2-3）$$

(2-3)中R_i出现在了方程的两端。这个问题可以用迭代的方法来解决。设R_i^n为R_i的第n次近似计算结果，则第n+1次计算结果为：

$$R_i^{n+1}=B_i+C_i+\sum_{\forall j\in hp(i)}\left\lceil \frac{R_i^n}{T_j} \right\rceil * C_j \qquad 公式(2-4)$$

迭代以$R_i^n=0$开始，以计算收敛结束，即当$R_i^n=R_i^{n+1}$。容易证明，当$\sum \frac{C_i}{T_i} \le 1$ 时，即CPU的使用率小于100%的情况下 ，迭代是确定收敛的。由于我们研究的是任务的可调度性，因此当$R_i^{n+1} > D_i$时，算法可以提前结束，此时任务τ_i不可调度。(2-4)中的迭代算法很容易用软件来实现，作为任务可调度性判断的依据。

2.5 本章小结

作为保障实时系统时限性和可靠性的最重要手段之一，一直以来，实时调度算法都是实时系统研究中的重要课题。实时多任务的调度算法则是该领域的热点问题。

本章对实时调度算法的研究进行了详细的介绍。首先，从实时调度的基本知识出发，介绍了包括任务及任务相关性在内的一些与实时调度相关的基本概念，并介绍了实时调度算法的分类、常用的实时调度策略以及实时调度算法的性能评估标准。随后，本章以实时多任务的调度算法研究为背景，就其两大热点问题—实时多任务系统的静态和动态调度算法展开了讨论。以算法所采用的方法为依据，对目前所提出的实时多任务系统的调度算法进行了分类地介绍及分析，从而对这一领域的研究现状做了比较全面的阐述。在此基础上，我们详细分析了传统上实时多任务系统调度算法研究中所存在的问题，进而提出了本论文所要解决的问题。

第三章 SM模型的确定和集成调度器的设计思想

从本章开始，我们开始介绍SM模型的确定思想和集成调度器的设计。集成调度器以截止时间单调调度理论为基础，其设计目标是要建立一套较为通用的实时调度策略，使之能适用于大部分的实时应用系统，从而能为真实实时应用系统的设计提供一个调度方法的模板。本章将概要介绍对SM模型确定思想以及集成调度器的设计中的一些需要考虑的因素，也就是如何将调度算法应用到调度器的设计上。本章前两节对比了现实实时系统中常见的实时任务的类型和集成调度器中处理的任务类型，提出了集成调度器在性能、可靠性等方面的要求。

3.1 实时多任务模型

多个任务在单CPU计算机系统上运行，为任务设置几种状态，与任务的运行规律、控制方式及设计者的考虑有关，因此，不同的实时多任务操作系统，所定义的任务状态有一些差别。一般而言，为任务设置的状态多，对任务的控制就更灵活，系统及使用的复杂性也必然增加。本模型定义任务为五种状态。

3.1.1 任务的五种状态

(1)执行状态：又叫运行状态。任务获得CPU的控制权，其代码正在CPU上执行时的状态，由于CPU只有一个，因此，在任何时刻，处于此状态的任务只能有一个。

(2)就绪状态：任务具备运行条件，但由于调度优先级低，未获得CPU的控制权而所处的一种等CPU的状态。

(3)等待状态：是任务的暂停状态，或说是不可执行状态。使任务等待的原因可能是等I/O的完成，等中断的发生，等数据，等时间间隔到或等事件。使任务等待的原因有多少，取决于任务的控制方式及管理策略。

(4)挂起状态：该状态是通过调用挂起命令（suspend）将指定任务挂起时所处的状态。被挂起的任务可能是调用者自己，也可能是处于其他状态的别的任务。挂起状态也是一种不可执行的等待状态。被挂起的任务只能用激活命令激活后方可能运行。

(5)等待挂起状态：等待挂起状态是处于等待状态的任务又被其他任务调用挂起命令挂起所处的状态。

除执行状态外，同时处于某状态的任务不止一个，故应对同时处于某种状态的

多个任务加以管理。

3.1.2 任务状态的变化规律

任务被创建（静态或动态创建）后，其状态总是在上述5种状态之间变化，变化规律如图3-1所示：

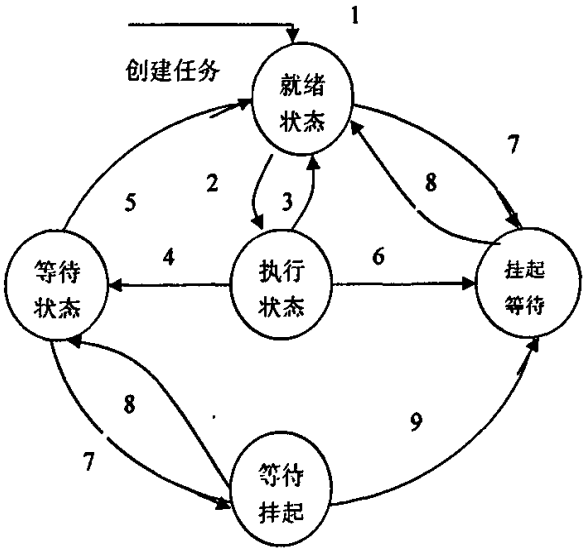


图 3-1 任务的状态及其转换

其中：

(1)任务创建后处于就绪状态。

(2)处于就绪状态的优先权最高的任务被任务调度程序调度而进入执行状态。

(3)正在执行的任务的处理机被优先级更高的任务剥夺，被剥夺处理机的正在执行的任务便退回到就绪状态。

(4)任务在执行中提出I/O请求后需要等I/O的完成，或者请求数据时而数据不存在，或者等中断的发生，都会导致执行的任务变为等待状态。

(5)处于等待状态的任务由于所等事件出现，如I/O完成、中断发生等，被唤醒后处于就绪状态。

(6)执行中的任务调用挂起命令将自己挂起。

(7)执行中的任务调用挂起命令将其他任务挂起。若被挂起的任务原来处于就绪状态，则执行挂起命令后处于挂起状态；若原来处于等待状态，则挂起后处于等待挂起状态。

(8)执行中的任务调用激活命令将由挂起命令挂起的指定任务激活。若被激活的任务原来处于挂起状态，则激活后处于就绪状态；若原来处于等待挂起状态，则激活后回到等待状态（因为此时所等事件尚未出现）

(9)处于等待挂起状态的任务，当其所等事件出现后进入挂起状态。

3.2 实时任务类型分析

3.2.1 按照任务的时间要求分类

按照对处理器的时间要求，任务通常可以被分为四类:无时间要求的任务、硬截止时间任务、固定截止时间任务、软截止时间任务。

- 无时间要求的任务

这类任务其实不是实时任务。但在一个实时系统中通常是实时任务和非实时任务是共存的。因此，无时间要求的任务也是实时系统设计中需要考虑的一类任务。

- 具有硬截止时间的任务

通常具有硬截止时间的任务也被称为硬实时任务或关键任务。其特点是任务具有时间限制。任务被触发后必须在其截止时间内完成，否则会造成灾难性的后果。因此，这类任务也是实时系统中最重要的一类任务。例如生命攸关的系统和飞机的稳定性控制。尽管它们的特性是关键的，并不意味着这些关键性任务的每一次重复都要在一个给定的时间限内成功地被执行。关键任务经常要以很高的频率执行，远远超过实际所需的情况。这造成了时间冗余，从而保证关键性的周期任务i的每n_i次重复出现能成功地运行一次，这就足以保证系统的安全。

- 具有固定截止时间的任务

任务具有固定的截止时间。完成时间超过截止时间的任务示例将被视为执行失败或者是结果无效。但是任务的失败不会造成灾难性的后果。调度这些任务的目的就是最大化在其时间限内被成功执行的任务的百分比。这类任务可以是周期时间较长的周期性任务，也可以是非周期性任务，它们都是按设定的北京时间的某个时刻，到时必定运行一遍的任务。实时系统中的制表打印任务、历史数据库的管理任务等，都属于此类任务。

- 具有软截止时间的任务

软截止时间的任务是指执行结果在超过了截止时间的情况下还有一定参考价值的那一类任务。因此通常对这类任务的响应是以平均响应时间来衡量。而不是考察具体的任务实例。

不难看出，任务的时间要求也在一定程度上反映了任务的重要性。通常任务的

重要性是:硬实时任务>固定实时任务>软实时任务>非实时任务。这也为我们的优先级分配策略提供了一定的参考性。

3.2.2 按照任务到达模式分类

任务的到达模式(Arrival Pattern)是指任务实例被触发的时间特性。任务按照其到达模式可以分为:

- 周期性(Periodic)任务

如果一个任务在t时刻被触发，而且可以预见其下一次触发是在t+T±Δt，那么我们称这个任务是周期性任务.其中T是一个常数代表任务的周期，Δt代表每次任务到达之间的可以接受或忽略的时间误差(通常是一个较小的值，如0.01T).如采样任务、绝对时间显示任务等都是周期任务，对周期时间要求比较严格，而像实时数据显示刷新任务，虽然也是周期任务，但对周期时间要求没那么严格。

- 非周期性(Aperiodic)任务

除去周期性任务的其他任务都可以被看作非周期性任务，它们只是偶尔发生。例如，当飞行员希望转弯时，很多与这个操作相关的子任务将被设置。由于非周期性任务的突发性，它们不能被预测，所以必须预留足够的计算机资源来实时地执行它们。对于非周期性任务，我们仍然可以对其下次触发的时间作出一定的分析，从而将非周期性任务细分为以下几类:

- 有限制的(Bounded)非周期性任务

如果任务的触发必须在其前一次触发后一定的时间间隔之后才会发生，也就是说任务的两次触发之间有一个最小的时间间隔，这种到达任务被称为有限制的非周期性任务。

有限制的非周期性任务在实际的实时系统中是很常见的，如电子器件中的物理特性，硬件限制(如中断等)，软件设计的限制等都可能造成这种限制的任务到达模式。

- 突发性的(Burst)非周期性任务

如果任务的两次触发之间没有最小的时间限制，但在一段时间内任务的最大的被触发的次数是已知的，这种任务的被称为突发性的周期性任务。

突发性的非周期性任务可以用事件密度(Event Density)来描述。事件密度包括两个要素:

突发间隔(Burst Interval):一个固定长度的时间段。突发量(Burst Size):在突发间隔内任务最大的可能被触发的次数。

例如，每秒钟最多有2辆车通过收费站。

- 无限制的(Unbounded)非周期性任务

当一个非周期性任务既不是突发性的，也不是有限制的，即任务的两次触发之间没有最小的时间间隔，而且在任意时间段内任务被触发的次数都是无限制的，这种任务被称为无限制的非周期性任务。

3.3 SM模型的确定以及实时任务集成调度器的设想

3.3.1 SM模型的确定

前面我们对截止时间单调调度算法的可调度性测试方法进行了详细的分析，在综合考虑了实时任务的五种状态以及状态的变化规律，并且对于实时任务的类型按照时间要求和任务的到达模式两个纬度进行了系统的划分，在此基础上，我们提出了新的任务调度模型——SM模型。

3.3.2 实时任务集成调度的设想

实时任务集成调度将为预定义的任务分配优先级，分析任务集的可调度性，并模拟任务的执行。实时任务集成调度的设计目的是充分利用已知的实时调度理论，综合处理各类实时任务，从而为真实的实时系统提供调度算法的模板。因此，实时任务集成调度必须具有以下的特点:

- 能处理多种类型的实时任务

由3.2节中对实时任务的类型的分析我们不难看出，实时系统中难于实现的是硬实时任务和固定实时任务的调度。而软实时任务的调度比较容易，因此我们对实时任务集成调度的设计中排除了对软实时任务的考虑。这里要说明的是无限制的非周期任务中没有包括硬实时类型的任务，这是因为无限制的非周期任务的触发可以无限的密度进行，在理论上是无法保证单个任务的截止时间的实时任务

- 性能的调度策略

基于第2章第4节中对调度策略的分析，我们将在实时任务集成调度中使用基于截止时间单调的算法。在保持较小复杂性的前提下(相对于动态调度算法而言)得到较好的调度性能。

- 可预测性--可调度性测试

在确定了优先级分配策略的基础上，我们将分析任务集的可调度性。对于周期性任务，可以使用2.4所示的可调度性测试方法。对于有限制的非周期性任务，可以将其映射为周期等于其最小触发间隔的周期性任务;对于突发性非周期任务，可以将多个任务实例映射为一个周期性任务(见第4章实现部分)，仍然使用2.4所示的

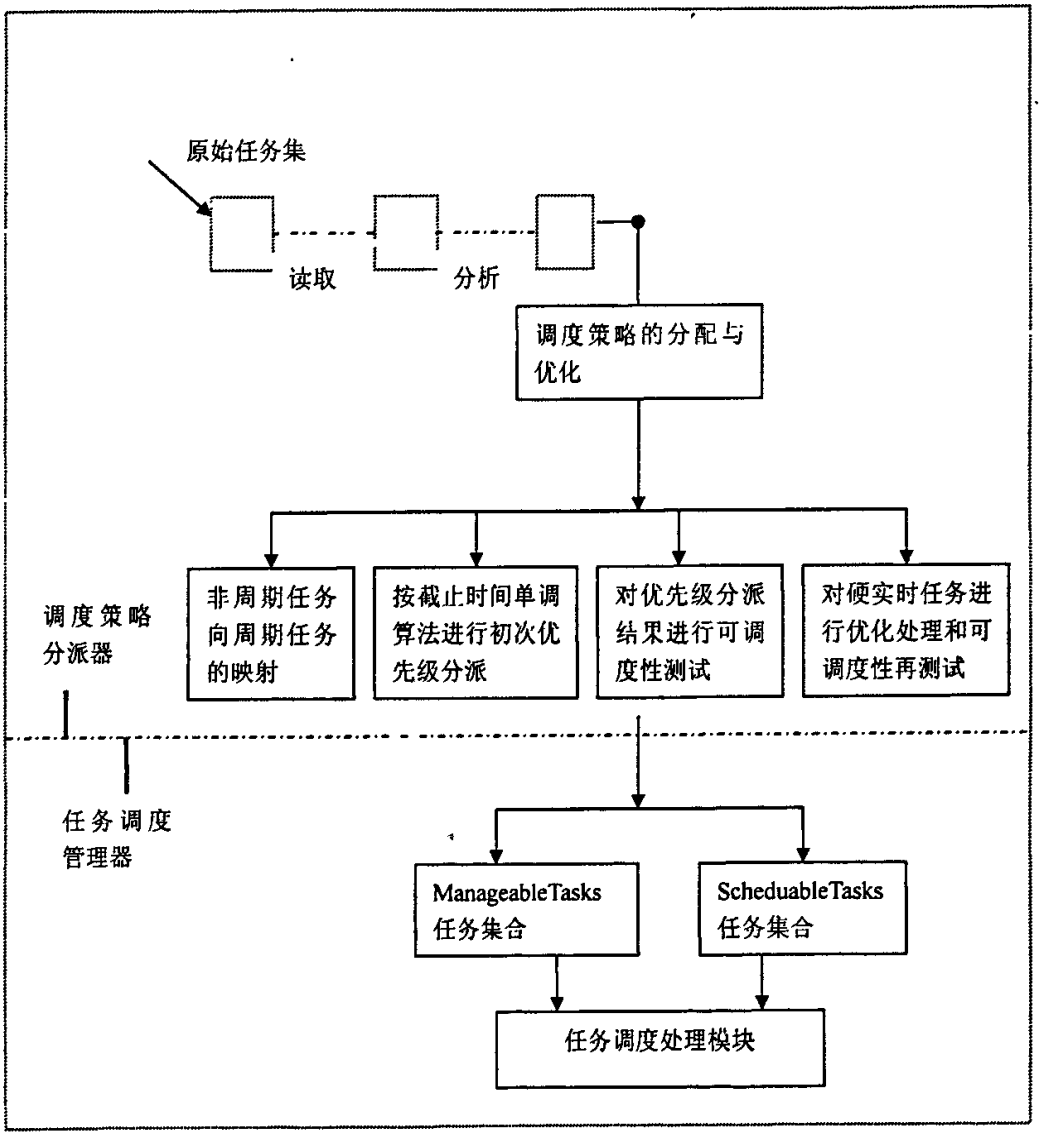
可调度性测试方法来保证任务集的可调度性。对于无限制的非周期任务，单个任务实例的可调度性是无法保证的。任务仍然采用基于优先级的抢占式调度。但为了不影响周期性任务和其它非周期性任务的处理，无限制的非周期任务将以低优先级执行，以不破坏对其它任务的已经过验证的可调度性。

- 可靠性--对硬实时任务的考虑和对截止时间单调算法的改进

由于硬实时任务的重要性，我们将尽量保证这类任务的执行不受其他任务的影响。因此，在不破坏整个任务集的可调度性的前提下，我们将提高硬实时任务的优先级。因此我们对截止时间单调算法做出了改进，详见第4章。

第四章 集成调度器的设计与实现

4.1 集成调度器的体系结构



4.1 实时任务集成调度器的结构

集成调度器在结构上由如图4-1所示的两个部分组成，包括调度策略分派器和任务调度管理器。调度策略分配器主要负责任务集的读取和分析，并决定使用的调度策略;而任务调度管理器主要负责按照调度策略分配器决定的方案把处理机分配

给就绪队列中的一个任务并使之执行。

下面是实时任务集成调度器两个基本组成部分完成的功能:

- 调度策略分派器
  - ✧ 负责原始任务集的读取和分析
  - ✧ 将有限制的非周期任务和突发性非周期任务向周期性任务映射
  - ✧ 按截止时间单调算法进行初次优先级分配
  - ✧ 对优先级分配结果进行可调度性测试
  - ✧ 对硬实时任务进行优化处理及可调度性再测试
- 任务调度管理器的功能
  - ✧ 根据调度策略分派器将实时任务集合中的任务分为ScheduableTasks和ManageableTasks两个集合的基础上，确定任务的就绪队列
  - ✧ 对就绪队列中已经按照优先级排好的任务进行调度，过程见4.3

简单来说，实时任务集成调度器就是利用调度策略分派器来为预定义的任务集按合理的方式分配调度策略，再利用任务调度管理器来调度就绪队列中的任务分配处理机使之执行。下面将分别介绍调度策略分派器和任务调度管理器的设计和实现。

4.2 调度策略分派器

4.2.1 原始任务集的描述和读取

由于实际的计算机实时控制系统在连续生产中始终处于工作状态，且实时作业在设计时，其规模、要完成的工作、负荷量等都比较清楚，因此实时任务往往在系统开工初始化时静态创建，创建后就一直驻留在系统中，直至系统停止运行。因此，我们假设本系统中的原始任务集也是静态一次性创建，直至系统关闭时撤销。

原始的任务集以文件的形式存放，其中:

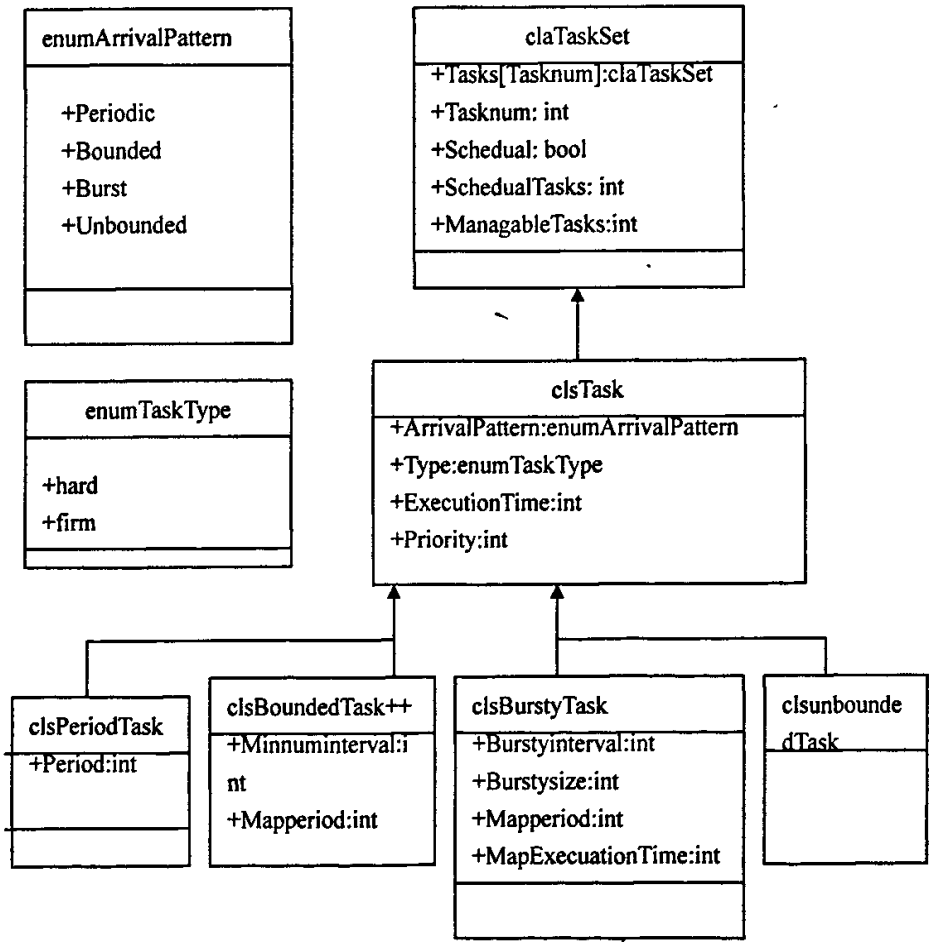
周期性任务以关键字Periodic开始，被描述为:T (C, D, T, Type)的四元组，其中C为最坏情况下的执行时间，D为截止时间，T为周期，Type为截止时间的类型(可为Hard或Firm)。例如，Periodic (50, 100, 200, Hard)表示最坏执行时间为50ms，截止时间为100ms，周期为200ms的周期性硬实时任务。

有限制的非周期任务以关键字Bounded开始，同样被描述为: T (C, D, T, Type)的四元组，其中C, D, Type的含义与周期性任务相同，不同的是这里的T应被理解为任务的最小触发间隔，而不是任务触发的周期。例如Bounded (50, 100,200, Firm)

的任务是最坏执行时间为50ms，截止时间为100ms，周期为200ms的有限制的非周期固定实时任务。

突发性的非周期任务以关键字Burst开始，被描述为：T (C, D, BI, BS, Type)的五元组，其中C, D, Type的含义与周期性任务相同，新增的两个属性为突发间隔BI (Burst Interval)和突发量BS (Burst Size)。表示任务T在突发间隔BI内最多可能被触发BS次。例如Burst (50, 100, 500, 10, Firm)为最坏执行时间为50ms，截止时间为100ms，在每500ms中最多被触发10次的固定实时任务。

无限制的非周期任务以关键字Unbounded开始，被描速为三元组:(C, D, Type)。如3.2节所述，由于理论上无限制非周期任务可能以无限大的密度触发，要保证每个任务都在其截止时间内完成是不可能的。因此我们在无限制的非周期任务中只考虑固定截止时间的任务，所以这里任务类型Type只能为Firm. 如Unbounded (100, 200, Firm)表示执行时间为最坏执行时间为100ms，截止时间为200ms的固定实时任务。



4-2 任务集类图

实时任务管理器从文件中读取任务，保存在如图4-2所示的任务集中。其中clsTask是任务的基类，由它派生出四个类型的任务。任务集是包含多个任务的链表。

4.2.2 有限制的非周期任务向周期性任务的映射

在将原始任务集表示出来之后，需要对任务集分派调度策略和进行可调度性测试。我们在第2章第4节中讨论的可调度性测试的方法都是针对周期性任务的。

对于实时任务调度器中存在三类非周期性任务，这类可调度性测试方法都不适用。这个问题的一个解决方法是用保守的方式将非周期性任务映射为周期性任务。

对于有限的非周期任务，由于任务的到达有最小时间间隔。因此在最坏情况下任务将以最小间隔时间为周期被触发，此时任务可以被看作周期性的任务，周期为任务的最小间隔。映射的结果使我们仍然可以使用第2章讨论的方法测试任务集的可调度性。

图4-2中，clsBounded类的MapPeriod属性用于保存映射的任务周期。映射过程的伪代码如图4-3所示

```
For(i=0; i<TaskSet.TaskNum; i++){
 If(TaskSet[i].Type==Bounded)
    TaskSet[i].MapPeriod=TaskSet[i].MiniNumInterval;
    }
```

图4-3有限制的非周期任务向周期性任务映射的伪代码

4.2.3 突发性非周期任务向周期性任务的映射

与有限制的非周期性任务类似，突发性非周期任务也可以用保守的方法映射为周期性任务。由于突发性任务在BurstInterval(突发间隔)时间内最大触发次数为BurstSize次，而最坏情况是这BurstSize同时被触发。因此，我们可以将这任务的BurstSize个同时触发当作一个任务来处理。用图4-4所示的方法可以将突发性任务映射为周期性任务：

```
For (i=0; i < TaskSet.TaskNum; i++){
 if (TaskSet[i].Type==Bursty){
 TaskSet[il.MapPeriod=TaskSet[i].BurstInterval;
 TaskSet[i].MapExecutionTime=
 TaskSet[il.BurstySize*TaskSet[i].ExecutionTime;
```

图4-4突发性非周期任务向周期性任务映射的伪代码

这里需要注意的是映射后的任务的执行时间为MapExecutionTime==BurstySize*ExecutionTime。可能发生MapExecutionTime>Deadline的情况。这种情况说明映射后的任务执行时间大于其截止时间，是不可调度的。这种情况下不能将其映射为周期性任务处理。

4.2.4 两个任务集合—ScheduableTasks和ManageableTasks

在将两种非周期任务映射为周期性任务之后，我们就可以为这两种非周期任务(BoundedBurst Tasks)和周期性任务(Periodic Tasks)分配调度策略(无限制的非周期任务另外考虑)，具体来说就是为每个任务分派其优先级，并测试其可调度性。

由于系统中存在无限制的非周期性任务，其可调度性在理论上无法保证。对于其它三类任务，我们可以对其进行可调度性的测试。但是由于任务集本身的性质，测试的结果可能仍然是不可调度。

基于以上考虑，实时任务调度器将任务集划分为两个子集：子集ScheduableTasks(见图4-2, TaskSet类)是调度器在调度算法计算完成后求出的可调度性能够保证的任务的集合，他们排在任务集队列的前面，按优先级顺序排列，高优先级的在前面；子集ManageableTasks(见图4-2, TaskSet类)是剩余的可调度性不能保证的任务的集合，它包括所有的无限制的非周期任务和部分的其它任务，这些任务也按优先级排序。

对调度算法的一个要求是要最大化ScheduableTasks。这里所说的最大化是指ScheduableTasks包含“尽可能多”的任务--将ManageableTasks中的任何一个任务加到ScheduableTasks中都将导致ScheduableTasks静态不可调度，由于硬实时任务的重要性，调度算法最起码要保证系统中的所有硬实时任务都是可调度的。按照这种思路，在实时任务调度器中，如果按调度算法计算结果ScheduableTasks中包括

了所有的硬实时任务，我们说这个任务集是可调度的，反之，如果不能保证所有硬实时任务的可调度性，我们说任务集是不可调度的，这用任务集类中的Scheduable属性来表示(见图4-2)。

4.2.5 可调度性测试

这里我们先抛开优先级分配策略，首先考虑可调度性测试的方法。在优先级分配算法结束后，ScheduableTasks中的任务已按照优先级由大到小排序，而且这些任务都是周期性任务或者已经映射为周期性任务。同时实时任务集成调度器被实现为静态优先级分配。

以上条件的具备，使我们的模型符合2.4中所述的可调度性测试方法的应用条件。从而可以通过计算ScheduableTasks中每个任务的最坏响应时间来确定ScheduableTasks的可调度性.

具体的测试算法如2.4节所述，这里不再重复。

4.2.6 可调度任务集的确定

按照上一节讨论的原则，可调度任务集确认的原则是：

- 最小化原则:可调度任务集至少应该包括所有的硬实时任务，否则整个任务集将视为不可调度。
- 最大化原则:将任何一个任务从ManageableTasks移到ScheduableTasks都将导致ScheduableTasks静态不可调度。

从这个思路触发，我们提出了如图4-5所示的确定可调度任务集的算法。简单说来，可调度任务集的确认分为以下两个步骤进行

- 第一步，将ScheduableTasks初始化为所有的硬实时任务的集合(最小化原则)，ManageableTasks相应的被初始化为其它的任务的集合(即所有的固定实时任务的集合)。将ScheduableTasks按截止时间单调算法排序，并按照2.4所述的算法测试其可调度性。如果ScheduableTasks任务集可调度，那么我们认为整个任务集可调度，转到第二步。反之如果ScheduableTasks不可调度，整个任务集不可调度，算法结束。
- 第二步，在保证可调度性的前提下，将尽可能多的固定实时任务(Unbounded类型的除外)加入到ScheduableTasks中(最大化原则).具体的做法为对ManageableTasks中的每个任务进行一次单循环(Unbounded类型除外)，将每个任务加入到ScheduableTasks,重新按照截止时间单调算法分配优先级并进行可

调度性测试。如果可调度性仍然，尝试将下一个任务加入到ManageableTasks中；如果可调度性不成立，则将该任务移回ManageableTasks中，测试下一个任务。

以上两步完成后，ScheduableTasks和ManageableTasks都被确定了下来。ScheduableTasks中的任务按照截止时间单调算法可调度并且具有最大化的性质。需要注意的是，“最大化”的ScheduableTasks不是唯一的，如果在第二步中用不同的顺序测试将任务加入到ScheduableTasks中，可能会得到不同的结果。但我们认为这些结果都是符合最大化原则的，即将任何一个ManageableTasks中的任务移动到ScheduableTasks中都将导致ScheduableTasks不可调度。

为了避免程序的多次执行得到的ScheduableTasks结果不同，我们在具体实现时规定了第二步中的任务测试的次序。ManageableTasks中的任务将按照周期性任务，有限制的非周期性任务，突发性的非周期任务的次序测试;同一类型的任务按照截止时间由小到大的次序测试。

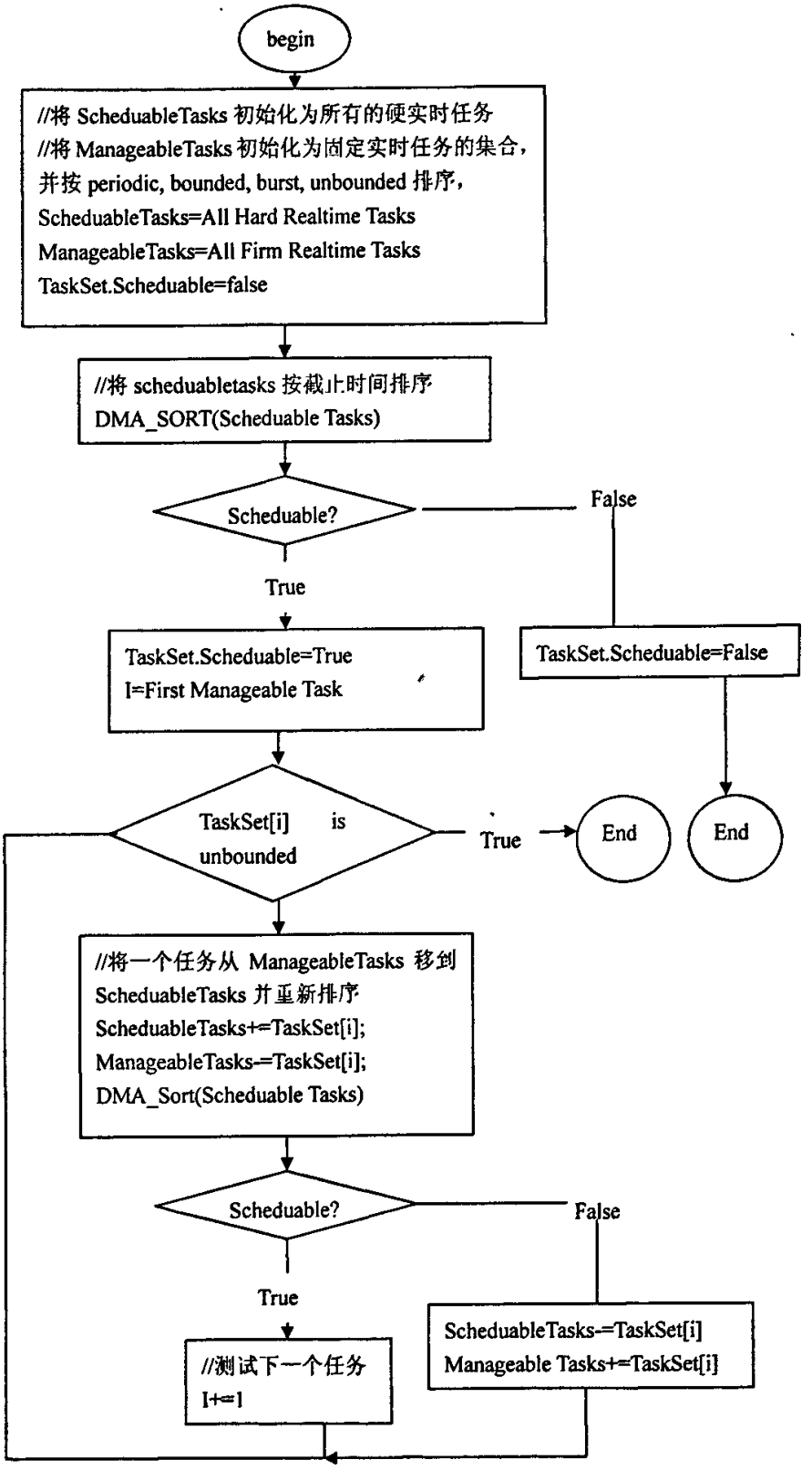

begin
//将 ScheduableTasks 初始化为所有的硬实时任务
//将 ManageableTasks 初始化为固定实时任务的集合，
并按 periodic, bounded, burst, unbounded 排序，
ScheduableTasks=All Hard Realtime Tasks
ManageableTasks=All Firm Realtime Tasks
TaskSet.Scheduable=false
//将 scheduabletasks 按截止时间排序
DMA_SORT(Scheduable Tasks)
Scheduable?
False
True
TaskSet.Scheduable=True
I=First Manageable Task
TaskSet.Scheduable=False
TaskSet[i] is unbounded
True
End
End
//将一个任务从 ManageableTasks 移到
ScheduableTasks 并重新排序
ScheduableTasks+=TaskSet[i];
ManageableTasks-=TaskSet[i];
DMA_Sort(Scheduable Tasks)
Scheduable?
False
True
//测试下一个任务
I+=1
ScheduableTasks-=TaskSet[i]
Manageable Tasks+=TaskSet[i]

图 4-5 可调度任务集的确定算法

4.2.7 对硬实时任务的优化

在ScheduableTasks集合和ManageableTasks集合确定下来以后，两个集合中的任务分别都是按照截止时间单调算法排序的，其中ScheduableTasks中的任务按照这种顺序来分配优先级，其可调度性是能够保证的。考虑到ScheduableTasks中既有硬实时任务，也有固定实时任务。由于硬实时任务的重要性，有必要对硬实时任务的调度进行优化，以把特殊情况下(例如，任务集中的某个任务以超过预先定义的频率发生，从而影响到优先级较低的其它任务的执行)硬实时任务失败的可能性进一步降低。这里我们采用的方法是在不破坏ScheduableTasks任务集的可调度性的基础上，尽量提高硬实时任务的优先级。具体算法如图4-6所示。算法描述如下:

- 第一步，对ScheduableTasks中所有的硬实时任务作一次循环，设当前考察的硬实时任务为 i，如果第i-1个任务是固定实时任务，则交换第 i个任务和第i-1个任务的优先级(即将提高硬实时任务的优先级，同时降低固定实时任务的优先级)。
- 第二步，测试交换优先级后的任务集是否可调度。如果可调度，则优先级交换成功；如果不可调度，则取消优先级的交换。同时进行下一个硬实时任务的优化测试。
- 第三步，一次循环完成后，如果至少有一个硬实时任务的优先级优化成功(表明存在进一步优化的可能)，则转第一步(尝试进一步的优化)；否则算法结束(硬实时任务占据最高的优先级或者是提高任何一个硬实时任务的优先级都将导致任务集不可调度)。

算法完成后，ScheduableTasks成为一个有序的任务的集合。按照任务的排列次序由高到底的为其中的任务分配优先级是一个优化的分配方案。既保证了任务集的可调度性，又为重要的任务(硬实时任务)提供了尽可能好的服务。

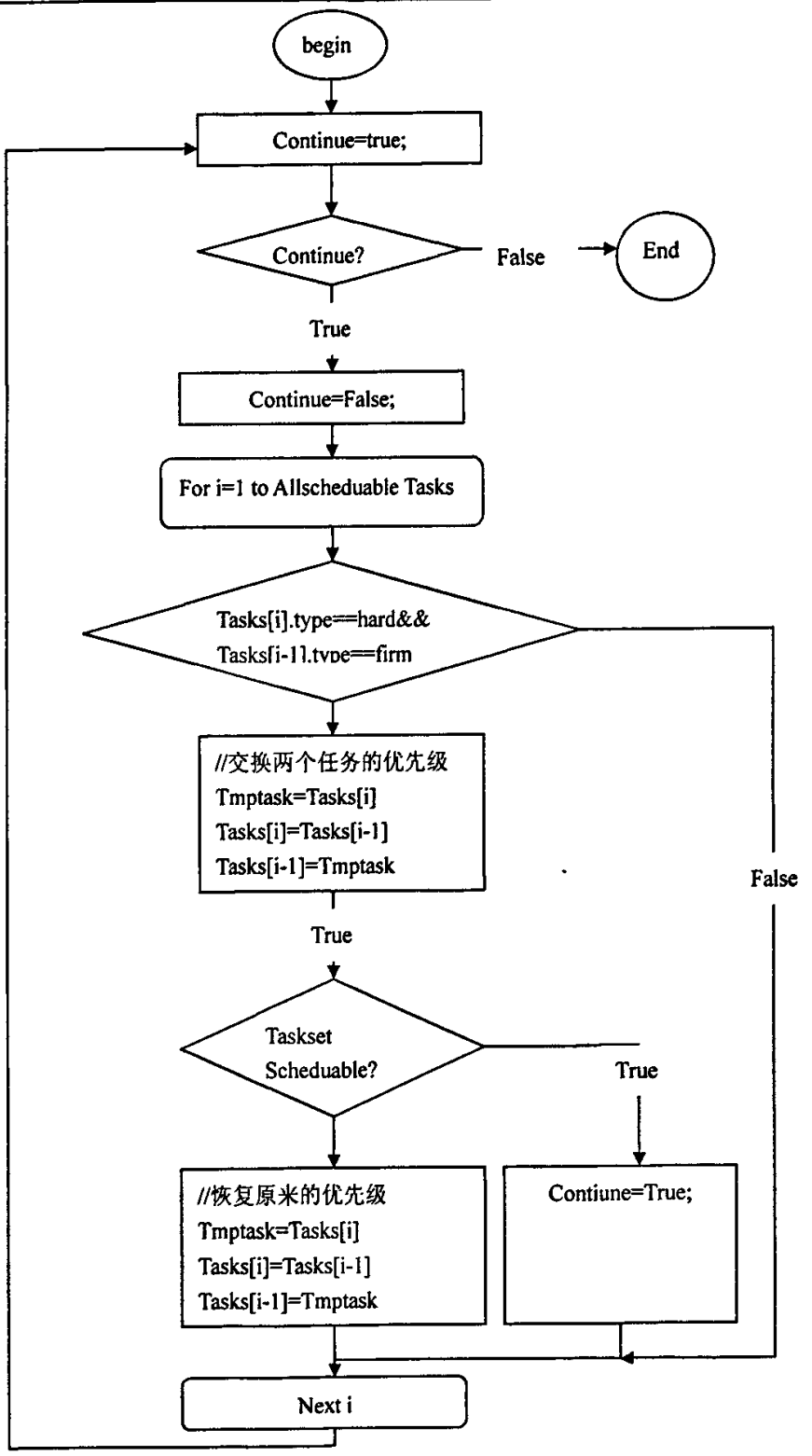


图 4-6 硬实时任务优先级优化算法

4.2.8 ManageableTasks的调度

与ScheduableTasks不同，我们不能从理论上对ManageableTasks中的任务可调度性做出保证。但我们可以使用一个相对合理的方法来为ManageableTasks中的任务分配优先级。对ManageableTasks中的任务优先级的考虑主要基于以下两点：

- ManageableTasks中的任务优先级不能高于ScheduableTasks中的任务，否则将破坏ScheduableTasks中任务的可调度性。即有Max（Priority of ManageableTasks)<Min (Priority of ScheduableTasks)成立。
- ManageableTasks的优先级分配要让尽可能多的任务能在截止时间内完成。由于截止时间单调算法具有静态最优性，我们将完全按照截止时间由小到大的顺序来为ManageableTasks分配优先级。

至此，整个任务集的优先级分配完成，整个任务集队列按照优先级由高到低排列。

4.3 任务调度管理器

在整个任务集的优先级次序确定以后，我们将对就绪队列中的任务进行调度。任务的调度是在任务调度管理器的协调下进行的。

4.3.1 任务调度中的数据结构

任务调度管理器所涉及到的数据结构主要有两个，一个是用于描述定义任务的数据结构，即任务控制块TCB，二是用来对就绪任务进行管理的数据结构，即就绪队列或就绪表。

1、任务控制块TCB

不同的系统，任务控制块的内容及其长度是有区别的。本系统根据需要TCB主要包括以下内容：

(1)任务标识符ID。任务标识符ID是一个数字，实际是TCB在TCB集合中的顺序号。主要用于对TCB的管理，根据ID计算TCB的起始地址，ID由用户定义，任务标识符在任务间不能重复。

(2)任务优先级。

(3)任务的现行状态。用于记录任务当前处于什么状态，可用一个字节S表示，各位的定义如图4-7所示。

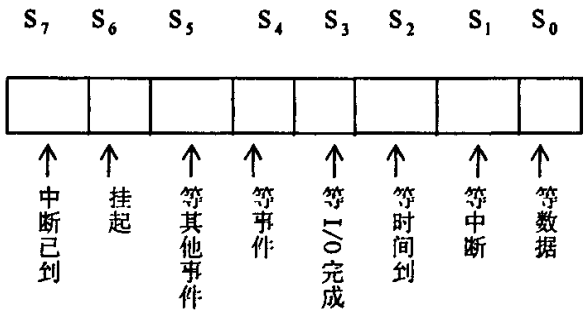


图 4-7 任务状态的定义

(4)任务现场保护区。执行中的任务放弃处理机的控制权，离开执行状态时 ，需要将当前的现场信息存入此处，以便再次运行时恢复现场。

2、就绪任务的组织

某时刻处于就绪状态的实时任务可能不止一个，这就需要对多个就绪的实时任务进行管理，就绪队列是对就绪任务进行管理的一种常用的数据结构，访问也最为频繁。本系统采用的是按照优先级高低排队的队列结构，在任务调度时比较简单省时。

设经过调度策略分派器已有16个并发执行的实时任务在就绪队列中等待执行，就绪队列设16个字节，每个就绪任务在就绪队列中占一个字节，用就绪任务的标识符ID和优先级进行排队，指针始终指向队列中优先级最高的任务。队列并未采用连接结构，而是按优先级由高到低顺序排列，最低优先级的任务始终占据第15号字节，因此，当第15号字节为空时，说明就绪队列已空。如图4-8所示。

	7 4	3 0
0	(ID)	(优先级)
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

图4-8就绪队列结构

4.3.2 任务的调度

根据任务管理中所涉及的数据结构以及ScheduleableTasks和ManageableTasks两个任务集合的特点，我们对于ScheduleableTasks和ManageableTasks的管理各不相同。

◆ 由于ScheduableTasks是可调度性有理论保证的任务集，其执行过程不需要特别的干涉，只需要按照任务的调度程序来处理任务即可。

值得注意的是任务处理完成后有一个计算任务最早下次触发的过程，并在下次触发之前让任务处于阻塞状态。这种策略目的是为了防止某个任务按照超过预定义的频率触发而破坏其它任务的可调度性。其中对于周期性任务，Min Interval的取值为任务的周期；对于有限制的非周期任务，Min Interval的取值为预定义的最小任务触发间隔；对于突发性的非周期任务和无限制的非周期任务，Min Interval的取值为零。

◆ ManageableTasks的执行管理。对于 ManageableTasks，由于任务的可调度

性没有理论保证，任务的实例有可能不能在截止时间内完成。这就要求我们对任务的执行进行监督，对于已到达截止时间但还没能完成的任务做出中止其执行的操作。

第五章 结论以及进一步的工作

本文详细分析了当前存在的各种实时多任务的调度算法及其性能特点，为研究实时多任务调度问题的实时系统设计者提供了参考。

截止时间单调算法是静态最优的调度算法。对算法的理论研究和实际应用都已比较成熟。但是截止时间调度算法本身是针对周期性任务的调度提出的，对于非周期任务的处理需要采用特别的方法将其映射为周期性任务或者使用间歇服务器的方法，特别是对于可调度性不能保证的无限制非周期任务需要有特别的运行时(Run-time)管理方法。

本文结合截止时间调度理论，设计并实现了一个实时调度模型—实时任务调度器。完成了如何将截止时间单调理论应用于各种类型的实时任务上。特别是其中的可调度任务集（ScheduableTasks)的确定方法，非周期性任务向周期性任务映射的方法等，都是真实实时系统设计中可以参考和借鉴的。

5.1 论文的创新工作

1.提出了一个 SM 模型将任务划分为可调度性能够保证的 scheduletasks 和可调度性不能够保证的 managabletasks 两个集合。

2.实现硬实时任务优先级优化算法，非周期任务向周期任务映射算法，最大化 scheduletasks 算法。

5.2 应用于真实系统

在将实时任务调度器应用于真实的系统时，还需要额外考虑以下几个方面：

- 任务的来源和触发方式

在实时任务管理器中，任务的触发是以时间为基础或者用任务触发线程来完成。但在真实系统中，许多任务来源于外部环境，如键盘按键产生的中断，雷达系统中新的飞行目标的出现，外部传感器产生的数据，来自于网络的数据等。不同的任务来源和触发方式可能导致不同的处理行为，例如中断服务程序的执行是按照硬件优先级进行的。在真实系统的设计中需要考虑将整个中断服务程序都以硬件优先级执行，还是用另一个进程/线程(按软件优先级运行)处理中断服务的主体部分，而将真正的中断服务程序用于触发服务进程/线程;或者是用轮询方式代替中断方

式等。

◆ 任务间的相互作用

在实时任务调度器的实现中对任务间的相互作用考虑较少。在一些应用中，任务之间可能不完全是独立的，某个任务可能需要等待另一个任务的运行结果才能运行，也就是说，某个任务必须要等待另一个任务完成之后才能开始运行，这样，便带来了任务之间顺序相关性。在实时多任务系统中，如何对这些具有时序约束的任务进行调度，则是一个有待研究的问题。对于共享同一临界资源的任务，虽然我们采用了优先级继承协议来防止无限期的优先级倒置，但任务由于竞争临界资源而产生的最大延时只有在临界资源本身和任务的优先级都被确认后才能计算出来。因此对于真实系统中有资源竞争的任务，需要在任务优先级确认后重新计算任务可调度性。

5.3 进一步的改进

本文就实时多任务调度算法有如下认识与建议：

(1)目前还缺少对这些算法的综合性能的比较与分析，许多研究中只对相关的算法(如改进的算法与最初的算法)进行了性能比较，而针对不同类别算法的性能比较很少；

(2)为了对非周期任务调度算法进行综合的评价与分析，有必要建立一个仿真实验平台；

(3)算法的研究总是基于一定的前提与假设，如要求周期任务的截止期等于周期，不存在任务的就绪抖动(release jitter)与任务同步等问题，如何放宽这些约束需要进一步地研究；

实时任务调度器模型在以下两个方面进行改进：

◆ 对截止时间大于周期的任务的处理

实时任务调度器中对任务截止时间的要求是小于或等于其周期。对截止时间大于周期的任务的分析方法要复杂的多。任务T_i的实例除了等待更高优先级的任务完成以外，还必须等待T_i的上一个实例的完成。Lehoczky等提出的忙周期分析法(Busy Period Analysis, BPA)和Audsley等提出的具有任意截止时间和抖动的任务的可调度性分析算法对这种情况作出了讨论。将这些分析方法包含进来可以增加实时任务调度器的通用性。

◆ 对随机软实时任务的处理

另外，实时任务调度器没有包含对软实时任务的处理，原因之一是通常软实时任务对时间的要求较小，比较容易实现。但近期的软实时任务调度研究趋向于不要

求任务的每个实例都符合截止时间的要求，只要任务延误截止时间的概率在一定范围内即可接受。这种分析方法被称为概率时间分析方法(Probabilistic Time Demand Analisis, PTDA)。在实时任务调度器中实现概率时间分析方法也是一个改进的方向。

参 考 文 献

[1]何军，孙玉方.提高软非周期任务响应性能的调度算法.软件学报，1998，9(10):721-726

[2]侯爽，宋颖慧.一种实时系统中的多任务可预测调度算法.计算机工程，2004，30(16):67-68

[3]刘怀，胡继峰.实时系统的多任务调度.计算机工程，2002，28(3):43-44

[4]刘怀，胡继峰.控制系统中强实时任务的调度.计算机工程与应用，2001，37(23):30-32

[5]乔颖，邹冰，等.一种实时异构系统的集成动态调度算法.软件学报，2002，13(12):2251-2258

[6]刘云生，陈怡.硬实时环境下截止期缓冲区管理方法.计算机工程，2003，29(7):70-72

[7] 毛羽刚，张拥军，等.强实时系统的调度.计算机工程与科学，2000，22(2):35-36

[8]王志平，熊光泽.实时调度算法.电子科技大学学报，2000，29(2):101-103

[9]涂刚，卢炎生.基于动态优先级策略的最优软非周期任务调度算法.计算机研究与发展，2004，41(11):2026-2034

[10]张玉明，权刚，等.计算机控制系统分析与设计.北京：中国电力出版社，2000，316-330

[11]王振宇，陈利，等译.实时系统与编程语言.北京：机械工业出版社，中信出版社，2004,365-392

[12]戴琼海译.实时系统.北京：清华大学出版社，2004，3，147-167

[13]付文勇，王力生.应用于实时系统的 RMS 算法的改进.微计算机信息，2004，20(11):39-40

[14]王强，王宏安，等.实时系统中的非定期任务调度算法综述.计算机研究与发展，2004，41(3):385-392

[15]王永吉，陈秋萍.单调速率及其扩展算法的可调度性判定.软件学报，2004，15(6):799-814

[16]张惠娟，周水生，周利华.一种混合实时任务系统的公平调度算法.西安电子科技大学学报（自然科学版），2004，31(2):272-275

[17]邹勇，李明树.开放式实时系统的调度理论与方法分析.软件学报，2004，14(1):83-90

[18]金宏，王宏安.一种任务优先级的综合设计方法.软件学报，2003，12(1):63-70

[19]王强，徐俊刚.一种新的基于优先级表的实时调度算法.电子学报，2004，32(2):310-311

[20]乔颖，实时异构系统的集成动态调度算法研究:[博士学位论文].北京：中国科学院软件研究所，2001

[21]李菲，截止时间单调技术研究与实时调度模型的实现：[硕士学位论文].四川：四川大学，2003

[22]Liu C L, Layland J W.Scheduling Algorithm for Multi-programming in a Hard Real-time Enviroment. J.AcM,1973,20(1):40-61

[23]Chetto H,Chetto M.Some Results of the Earliest Deadline Scheduling Algorithm[J]. IEEE Trans on Software Engineering.1989,15(10): 1261-1269.

[24]Dhall S K, On a Real-time Scheduling Problem [J]. Operations Research. 1978. 26(1):127-140

[25]Krishna CM, Shin G K. Real_Time systems[M]. Columbus. OH: McGraw-Hill Companies,Ine,1997.73-80

[26]Sprunt B.Aperiodic task scheduling for real-time sustems [Ph.D.Thesis]. Carnegie Mellon University,1900

[27]Lehoczky J P,Sha L,Strosnider J K.Enhanced aperiodic responsiveness in hard-time environments.In:Proceddings of IEEE Real_Time System Symposium.December 1987. 261-270

[28]Audsley N C,Burns A.On fixed priority scheduling,offsets and coprime task periods.Information Processing Letters Volume:67,Issue:2,July 30,1998:65-69

[29]Bertossi Alan A, Fusiello Andrea. Rate-Monotonic scheduling for hard real-time systems,European Journal of Operational Research Volume:96,Issue:3,February 1,1997:429-443

[30]K. Ramamritham and J.A.Stankovic," Scheduling algorithms and operating systems support for real-time environment," Proc of IEEE, vol.82, pp.55-67, jan,1994.

[31]Liu sha,Rajkumar R,Sathage S S.Generalized Rate-monotonic Scheduling:A

Framework for Developing Real-time systems. In:Proceeding of the IEEE, 1994, 82(1):68-81

[32]Katcher DI, Arakawa H, Strosnider JK. Engineering and analysis of fixed priority schedulers. IEEE Trans. on Software Engineering, 1993,19(9): 920~934.

致　谢

首先，我要由衷地感谢我的导师张玉明教授。在三年的学习生活中，张老师给予了我悉心的指导和无微不至的关怀，引导着我在实时系统研究上的前进方向。张老师渊博的学识、敏锐的洞察力、严谨的治学态度和诲人不倦的师长风范，将使我受益终生。

衷心感谢网管中心老师对我的大力支持。他们为我提供了一个良好的网络环境，在这里极大的方便了我对于所学相关资料的查询，使我在所研究的领域能够有比较快的进步。

衷心感谢实验室的各位同学，感谢他们在学习上与我所进行的讨论以及在生活中给予我的帮助和支持，与他们共同创造出的和谐融洽的工作气氛给我三年的学习生活留下了难忘的印象。

特别要感谢我的父母及丈夫，他们的鼓励和关怀是我最宝贵的财富，谨以此文作为微薄的礼物奉献给他们。

在学期间发表论文和参加科研情况

[1]冯艳红，张玉明.RMS 调度算法的改进及其可调度性分析.微机发展，2005，5

[2]冯艳红，张玉明.实时调度算法分类研究.微型电脑应用，2005，7

[3]冯艳红，张玉明.一种多任务实时系统的调度理论.计算机科学与实践，2005，8