湖南大学

硕士学位论文

单处理器环境下实时调度算法研究

姓名：赵公怡

申请学位级别：硕士

专业：计算机科学与技术

指导教师：任小西

20110528

摘 要

随着网络、通信、多媒体计算的迅猛发展，嵌入式系统得到了广泛的应用，实时系统的应用也逐渐从传统的科学研究、国防、工业控制等领域扩展到人类社会的方方面面。实时系统的研究主要集中在两个最关键的问题上，一个是对实时调度算法的研究，另一个是对实时任务集可调度性判定的研究。本文对应分别提出一种硬实时混合调度的可调度性判定算法 IISS(Improved Idle Slack Stealing)和一种改进的最小空闲时间优先(LSF)调度算法 DPTLSF(Dynamic Preemption Threshold LSF)。

IISS 算法主要是解决硬实时周期任务和偶发任务混合调度情况下的可调度性判定问题，以保证偶发任务的可调度性。基于调度与逆调度的概念，分析了最早截止期优先(EDF)调度中任意时刻的最大可挪用时间的计算方法；IISS 算法将偶发任务安排在周期任务的执行空隙与推迟周期任务执行后出现的可挪用时间中执行。根据不同偶发任务特征，确定一个动态挪用时间点 $T_{dynamic}$，得出偶发任务可调度性判定的充分条件。仿真结果表明，IISS 算法的预测准确率比已有算法 ISS 有明显提高，并且对于不同实时任务集的判定更具灵活性。

DPTLSF 算法是针对经典 LSF 调度算法中任务上下文切换频繁及任务截止错失率较高的缺点提出的。通过分析不同空闲时间的任务抢占对 LSF 调度算法性能的不同影响，基于抢占阈值策略，设计合理的动态抢占阈值，来避免任务切换频繁造成的“颠簸“现象的发生。仿真结果表明，改进后的算法在不同处理器负载、不同周期任务数情况下，都能够显著地减少上下文切换次数，降低任务集的截止期错失率。

关键词：实时调度；可调度性；可挪用时间；颠簸；截止期错失率

Abstract

With the rapid development of networking, communications and multimedia computing, embedded systems are widely used, and the application of real-time systems have been expanded from its traditional domains, such as scientific researches, national defences, industrial control and so on, to many other areas of human society. The research of real-time systems focuses on two key issues: one is the scheduling algorithms of real-time tasks, and the other is deciding the schedulability of a set of real-time tasks. To address these two problems, this thesis proposes a schedulability decision algorithm for a set of hard real-time periodic tasks and sporadic tasks named IISS (Improved Idle Slack Stealing), as well as an improved Least Slack first (LSF) scheduling algorithm named DPTLSF (Dynamic Preemption Threshold LSF).

IISS algorithm is designed to solve the hybrid scheduling schedulability decision problems of hard real-time periodic tasks and sporadic tasks and ensure the schedulability of sporadic tasks. Starting from the concept of EDF scheduling procedures and the reverse EDF scheduling procedures, the maximum time which can be diverted at any time of the EDF scheduling procedure is analyzed; IISS algorithm will schedule sporadic tasks to run either between the gaps of periodic tasks' execution time, or the time diverted from periodic tasks. According to different characteristics of sporadic tasks, it will determine a dynamic time point named $T_{dynamic}$ and output a sufficient condition for deciding the schedulability of sporadic tasks. Simulation results show that, IISS is more accurate and flexible than the existing algorithms.

DPTLSF scheduling algorithm is designed to address the limitations of a classical scheduling algorithm named LSF which has high context switching frequency and high deadline missing rate. DPTLSF is based on the preemption threshold strategy and the analysis of the performance impacts that different idle time preemptions imposed on the LSF scheduling algorithm, and it uses a dynamic preemption threshold to avoid the "bumps" phenomenon resulted from switching tasks too frequently. Simulation results show that the proposed algorithm can reduce the number of context switches significantly and can also reduce the deadline missing rate of tasks under different processor load and the number of tasks in different cycles.

Key Words: real-time scheduling; schedulability; appropriation time; bumps; missed deadline percentage

插图索引

附表索引

第1章 绪 论

1.1 实时调度的研究背景和研究意义

1.1.1 实时系统的应用背景

随着计算机技术的飞速发展，实时系统广泛应地用于科学研究、工业控制、国防、航天及互联网等领域。如今，移动电话、手表、电子游戏机、PDA、MP3、MP4、视频点播、数码相机、先进医疗仪器、电视、冰箱等电子与通讯产品，电动车、电动汽车、航空航天飞机等，无不与实时系统紧密相关。

在计算机的很多应用领域，要求对实时任务进行及时处理并输出结果，否则可能造成灾难性后果。例如卫星发射过程中，必须对出现的各种事件做出响应，这种系统是批处理和分时系统无法满足的，于是产生了实时系统。实时系统初期仅用于军事领域，例如，IBM 先后开发出飞行模拟系统 Whirlwind 和防空系统 SAGE。到五十年代末期，实时系统进入工业过程控制领域。七十年代开始，伴随处理器技术的发展，实时系统逐渐走向民用领域。

二十世纪九十年代，在分布式控制、柔性制造、数字化通信和信息家电等巨大需求牵引下，嵌入式实时系统飞速发展，而面向实时信号处理算法的 DSP 产品则向着高速度、高精度、低功耗的方向发展。随着硬件实时性要求的提高，嵌入式实时系统的软件规模也不断扩大，逐渐形成实时多任务操作系统，并开始成为嵌入式实时系统的主流。近几年网络、通信、多媒体技术的发展为嵌入式实时系统应用开辟了广阔的天地，使嵌入式实时系统成为继 PC 和 Internet 之后，信息科学中新的研究热点。

1.1.2 实时调度

调度的实质是分配系统资源，包括处理器和其它运算、交互、存储资源。调度算法就是研究如何将这些资源合理分配各个任务，它是一种服务于系统目标的策略[1]。对于不同系统目标应设计不同的调度算法。通用操作系统的调度算法的目标是优化系统平均性能，如公平性、平均等待时间、系统吞吐率等性能指标，但这样的调度算法不适合实时应用的需要。

因为实时计算必须同时满足计算结果的逻辑正确与任务时限要求，所以，实时调度的一个重要目标是对任务的执行时序做出正确安排并对能否完成任务集中所有任务做出预测。只有这样才能在充分利用系统资源前提下满足实时应用的需求。目前，实时调度问题的日益复杂化与人们对实时性能需求的多样性导致实时

系统架构的复杂化。尽管在过去近四十年时间里，实时调度得到了飞速的发展，但是，大多数实时调度算法还不够完善，亟待进一步的研究。

1.1.3 研究目的与研究意义

实时调度理论研究成果对于提高我国国防武器装备、载人航空航天、工业自动控制、关键通信系统等领域的关键系统任务调度具有重要的现实意义。实时调度理论的研究就是要为满足复杂多任务实时系统的实时性与可靠性提供一条有效途径。

1.2 实时调度研究现状

1.2.1 硬实时周期任务和非周期任务的混合调度算法研究现状

在单处理器、任务间可抢占的实时系统中，对于硬实时周期任务的调度已经有一些很成熟的调度算法，而对于周期任务与非周期任务的混合调度研究的不多，主要分为三类：后台运行法[2]（background）、带宽保留法[2]（bandwidth-preserving）和时间挪用法[2]（stealing time）。这三类算法的目标都是在保证硬实时周期任务时限的前提下，达到对非周期任务的最快响应。它们都没有判断是否能满足非周期任务的时限要求，所以只适用于软实时非周期任务的调度，而不适用于偶发任务（即有时限要求的非周期任务）的调度。因此，有必要对硬实时周期任务与偶发任务的混合调度进行深入研究。

后台运行法是在周期任务的执行间隙安排非周期任务执行，这种方法实现简单直接，但是效率不高，没有充分考虑偶发任务的执行需求。带宽保留法是另外增加一个专门用来执行非周期任务的周期任务，当该周期任务执行完现有的非周期任务，还有剩余的执行时间时，对这些剩余时间的处理方法不同又出现了优先级交换 PE（priority exchange）[3]、可延期服务器（deferrable server）[3]、间发服务器（sporadic server）和时间挪用等算法。时间挪用法的基本思想是在保证满足周期任务的时限要求前提下，尽量挪出时间来处理非周期任务。Lehoczky 等人提出基于固定优先级调度的 OFP 算法[4]，通过计算在超周期中每个任务的最大延迟时间，得到最大的可挪用时间。何军等人在固定优先级调度的基础上结合动态优先级调度提出 ODD 算法[5]，进一步改进和提高了处理器的利用率和非周期任务的响应时间。之后，涂刚等人根据 EDF 动态优先级调度算法的性质和 Houssine Chetto 等人在文献[6]中对 EDF 算法及其逆调度、空闲时间的研究结果，求出了基于 EDF 动态优先级调度的最大可挪用时间。基于上面的最大可挪用时间，文献[2]提出了一个满足偶发任务可调度性的充分条件。

1.2.2 最小空闲时间优先（LSF）调度算法研究现状

LSF 调度算法是一种常用的动态优先级实时调度算法，是最早截止期优先（EDF）算法的改进算法。LSF 算法中任务优先级由任务的预测完成时刻到截止期之间的空闲时间决定，这段空闲时间越小，优先级越高。由于等待队列中的任务的空闲时间随时间严格递减，当前执行任务的空闲时间不变，等待任务的优先级随时可能超过当前执行任务的优先级，产生频繁的任务抢占，降低系统性能[7]。因此如何避免 LSF 调度算法中过多的任务抢占是一个具有重大意义的研究课题。

为了降低抢占式多道系统中 CPU 带宽的浪费及内存总开销，Express Logic 公司的 W.Lamie 于 1997 年提出了抢占阈值调度思想，并运用到该公司开发的商用实时系统 ThreadX 中。所谓的抢占阈值调度指的是任务集中其他任务要抢占当前正在执行的任务时不仅要满足任务优先级高于当前任务优先级，还必须满足高于当前执行任务的抢占阈值[8]。研究证明，抢占阈值思想能够显著的减少过多的上下文切换，提高系统资源利用率。

在抢占阈值调度思想提出之前，实时调度算法要么是抢占式的，要么是非抢占式的。抢占式调度算法的优点是注重紧急任务的执行，缺点是资源的浪费和较高的任务截止错失率；非抢占式调度算法的优点是较低的任务错失率和上下文切换，缺点是对紧急任务的响应不够及时。文献[9]研究了在固定优先级，确定任务集的环境下，抢占阈值调度优于抢占式和非抢占式调度。既然抢占阈值思想可以改进目前的调度算法，那最优的抢占阈值如何确定呢？这个问题其实是抢占阈值思想的核心问题。因为偏大的阈值会导致紧急任务的响应受到很大影响，而偏小的阈值则不能完全将过多的上下文切换消除。文献[10]中提出了一种任务集不确定，优先级动态变化的情况下，将抢占阈值思想引入 LSF 中的算法。即以当前任务完成率为参照，若完成率小于 20%，则阈值取为当前任务优先级；若完成率大于等于 20%，阈值取为足够大。这种抢占阈值设定方法简单但仍然不能很大限度的改进 LSF 算法性能。文献[11]中提出按如下公式确定阈值 h：h=ceil($K \bullet p$)，其中比例系数 $K \in [1, 2)$ 为给定常数，p 为与任务空闲时间相对应的优先级，函数 ceil(x)表示取大于 x 的最小整数。这种抢占阈值设定算法简单，但是算法对改进的效果存在不确定性，而且 K 值在实际应用中不易确定。文献[12]中提出将优先级继承协议集成到抢占阈值调度中，使得在最坏情况下内容切换数最小。文献[13]基于云模型给出了一个阈值的确定方法，抢占阈值由任务裕度和任务重要度两个特征参数决定。文献[14]中提出阈值的一种线性计算算法。然而，以上设计的阈值确定算法没有考虑到 LSF“颠簸”现象影响调度性能的根本原因，改进过的算法的上下文切换次数仍然比较大，任务截止错失率仍然比较高，应该改进 LSF 算法以进一步减少过多的上下文切换，降低任务截止错失率，让 LSF 算法在纯抢占式调度和非抢占式调度之间获得平衡，达到性能最优。本文针对空闲时间较小时，“颠簸”现象对任务集的截止错失率及上下文切换的重大的影响，设计一种新的

抢占阈值确定方法，并在此基础提出 LSF 的改进算法 DPTLSF。

1.3 本文的研究内容及主要工作

论文的主要研究内容是实时系统中的任务调度问题。首先研究了在硬实时系统中周期任务和非周期任务的混合调度问题，提出了一种新的偶发任务可调度性判定算法。其次，通过研究现有经典调度算法 LSF 的优缺点，找出影响 LSF 算法性能的主要原因，设计一种改进方案。最后给出了实验方案，用实验结果说明研究成果。具体地，本文主要完成如下两方面工作：

1. 研究了硬实时系统中周期任务与非周期任务的混合调度问题。根据不同偶发任务集特征动态设置可挪用时间点 $T_{dynamic}$，提出一种硬实时周期任务与偶发任务混合调度中偶发任务可调度性判定算法 IISS。实验结果表明，相比已有 ISS 算法，IISS 算法有更好的灵活性与预测准确率。

2. 研究了 LSF 调度算法“颠簸”现象的产生原因，找到了影响 LSF 性能的根本原因，并通过引入动态抢占阈值策略，设计出一种改进的算法 DPTLSF。实验结果表明，相比经典的 LSF 算法和已有的同类算法，DPTLSF 算法的任务抢占次数减少到经典 LSF 算法的 10%，任务集的截止期错失率减少 10%左右，显著提高了 CPU 有效利用率。

1.4 论文组织结构

论文共分为四章。

第 1 章为绪论，给出了课题研究背景和意义。本章首先介绍了实时系统和实时调度的研究背景，然后介绍了实时调度方面的相关研究现状，最后是本文的主要研究内容和论文组织结构。

第 2 章主要研究了当前实时系统中相关基础知识，主要包括实时调度基本策略、常用实时调度算法、实时任务集可调度性分析。

第 3 章研究了硬实时系统中周期任务与非周期任务的混合调度问题。通过分析已有算法 ISS 和最大挪用时间法的优缺点，提出了一种新的偶发任务可调度性判定算法 IISS，并通过仿真实验进行了算法改进效果验证。

第 4 章研究了影响 LSF 调度算法性能相关问题。针对 LSF 算法中任务抢占过多的现象，提出一种基于动态抢占阈值的改进算法 DPTLSF，给出了详细设计过程，并通过仿真实验进行了算法改进效果验证。

最后，总结了本文研究的主要内容，并对未来工作进行了展望。

第 2 章 基本理论与相关研究

2.1 实时系统相关理论

2.1.1 实时系统定义

与通用系统的一般追求系统平均响应时间和用户系统体验不同，实时系统是一类能及时响应外部事件，并且在规定时限内完成对事件的处理的计算机应用系统。可移植操作系统接口 POSIX（Portable Operating System Interface for uniX）将实时系统定义为：产生的系统输出的时间对系统至关重要的系统。系统的正确性不仅要保证计算结果的正确性，而且必须满足任务的时间约束，如果逻辑或时序出现偏差将会引起严重后果。

2.1.2 实时系统的特点

实时系统是必须在确定时间内完成对任务的响应和处理的计算机系统。和通用系统相比，它有如下基本特征：

1. 可预测性

可预测性指的是系统根据实时任务的响应时间来判断，确定是否满足任务的时间约束。由于实时系统中时间约束的重要性，使可预测性成为实时系统的一项重要性能指标，既体现在硬件延迟方面的可预测性，也体现在软件件中程序执行时间方面的可预测性。实时系统对外部事件的响应必须是可预测的，即使在最坏情况下，系统也要严格满足截止时间限制。例如，系统 A 的最坏情况下响应时间是 100 微秒，系统 B 的响应时间绝大多数情况下是 1 微秒，偶尔会是 10000 微秒。根据系统可预测性要求，我们知道，系统 A 的实时性能优于系统 B。

2. 时间约束性

时间约束性指任务不仅要满足任务之间的拓扑约束，还要满足各自的绝对时间约束和相对时间约束。在实时系统中，CPU 时间是最为重要的因素，也是系统需要管理和控制的第一资源。实时计算要求按照某个调度算法在某个时间片内完成对任务调度和执行，时序正确性和逻辑正确性一样重要。

3. 可靠性

在一些重要的实时应用场合，如航空航天、核反应堆、武器控制等，任何不可靠因素或者计算机的微小故障都可能导致难以预测的严重后果。因此，实时系统需要采用静态分析和保留资源的方法及冗余配置，使系统在最坏情况下都能正常工作。可靠性是实时系统中不可或缺的重要指标，针对提高系统可靠性的容错

研究也是实时系统研究中的重要内容。

4. 交互性

所谓交互性，是根据应用对象的不同和应用要求的不同，对交互操作的方便性和交互操作的权限性有特殊的要求[15]。

实时系统通常运行在一定环境下，外部环境是实时系统不可缺少的组成部分。计算机子系统一般是控制系统，必须满足任务各自的时限。外部物理环境一般是被控制的子系统，两者之间相互影响构成整个实时系统。例如，控制系统在某些情况下不允许用户干预或只允许授权用户干预。

随着实时系统应用范围的不断扩大，系统复杂性不断提高，实时系统具有以下新的特征：

5. 多种任务类型

在实时系统中，不但包括周期任务、偶发任务、非周期任务，还包括非实时任务。实时任务要求满足其时间约束，而非实时任务要求响应时间尽可能小。在同一个系统中出现多种类型任务，使得系统的调度算法设计与可调度性分析越来越困难。

6. 约束的复杂性

任务的约束包括时间约束、资源约束、性能约束。时间约束是实时系统的固有约束。资源约束是指多个实时任务共享优先资源时，必须按照一定的资源访问控制协议进行同步，以避免死锁和优先级反转。性能约束指必须满足如可预测性、可靠性、QoS 等性能指标。

2.1.3 实时系统分类

根据不同的分类原则，实时系统分为不同的类型。主要的分类方法如下所述。

1. 根据时间约束的重要性不同，将实时系统分为软实时系统和硬实时系统。

（1）硬实时系统

在硬实时系统中，时间约束是最重要的。系统必须在限定的时间内对任务作出响应，否则可能造成灾难性后果。例如，汽车保险气袋控制器、反锁刹车、自动驾驶飞行器、核工业控制系统，导弹发射控制系统等都属于硬实时系统。因此，在这类系统的设计和实现中应采用各种分析、模拟等方法对系统进行严格检验，以满足系统的时间约束。

（2）软实时系统

在弱实时系统中，对任务执行时间的要求相对宽松，计算时间偶尔超过任务截止期不会造成严重后果，可能仅仅是丢失了一些数据，使系统性能有所下降，用户体验收到微弱影响等。典型的弱实时系统有视频点播系统、信息采集与检索系统、在视频点播系统中，为了让用户流畅的欣赏视频，系统应该及时地将数据

从服务器下载到本地，然而偶尔丢失一些数据帧，对用户来说是感觉不出来的，不会对系统功能产生严重影响。

2. 根据服务对象的不同，将实时系统分为实时控制系统和实时信息系统。

（1）实时控制系统是以计算机为中心的过程控制系统，既可用于工业生产过程中的自动数据采集、监测、自动控制等，也可用于军用的武器制导、火箭发射、无人驾驶飞机等。

（2）实时信息系统指实时信息处理系统，处理信息的服务器接收终端的服务请求，然后进行检索、处理并将处理结果发送给终端。例如火车、航空订票系统等。

2.2 实时任务调度基础知识

2.2.1 实时任务调度理论的产生

关于调度理论可以追溯到上世纪初开始出现的运筹学分支——排队论。排队过程的一般模型如下图 2.1 所示在某服务窗口，顾客源的每位顾客到达服务窗口前，按照排队规则排队等候服务，窗口按照规则开展服务工作，顾客接受服务之后离开。图 2.1 中的排队结构指队列的数目和排列方式，排队规则和服务规则说明顾客在排队系统中按什么规则，以什么次序接受和开展服务的。

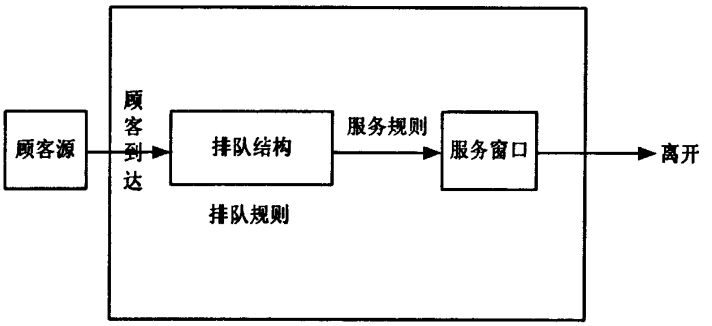

图 2.1 排队过程一般模型

排队论通过对服务对象和服务时间的研究，得出这些数量指标的统计规律，然后根据这些规律来改进服务系统的结构或重新组织服务对象，使得服务系统既能满足服务需要，又能在某些指标上达到最优。上世纪五十年代，计算机一出现，排队论就广泛地运用到处理器调度领域。该领域主要研究的内容是如何在一组处理器上调度任务集，目标是优化一个或几个性能指标。这时的调度理论被称为古典调度理论。其主要性能指标包括：平均响应时间、最大任务延迟时间、超时任务数、任务超时时间等。到了 20 世纪 70 年代，操作系统功能越来越强大，实时调度理论得到迅速发展。1973 年文献[16]的发表，标志实时系统研究的开始。八十年代以后，随着计算机技术的快速发展，特别是嵌入式系统领域，在航空航天，

家电，手机，互联网等领域广泛应用，实时调度理论也逐渐成为实时系统中最关键的技术课题之一。

2.2.2 实时任务的分类

实时任务的分类方法很多，主要有下面几种。

1. 根据任务的周期分类

周期任务：指任务的到达时间满足周期性规律，任务的每个请求称为任务的一个实例。

非周期任务：指随机到达的系统任务。

偶发任务：如果一个任务包含的作业的释放时间（即任务状态跳到就绪态时刻）是随机的但作业又是硬实时任务，称此任务为偶发任务。

2. 根据任务的时间约束严格性分类

强实时任务：指必须在时限内做出响应并完成的任务，不允许任何任务实例超时，否则会造成严重后果。强实时任务考虑的时间约束一般是其临界情况下的时间，如最坏情况下执行时间、最大响应时间等。

弱实时任务：允许任务超时，超时的计算结果仍有意义，超时的任务实例对系统不会有太大影响。

强-弱实时系统：强-弱实时系统通常是周期任务，并允许周期任务的一些任务作业超时，但这些超时的任务作业的分布应满足一定规律。

2.2.3 实时调度的基本概念和相关术语

为了描述和分析实时系统中调度和资源管理方法，建立起了一系列专业术语，下面介绍常用的一些术语。

任务：是完成特定功能的进程实体，是需要获得 CPU 资源的客户。它是实时调度的基本单位。

作业：通常任务有一组相关作业组成，任务的某一次执行称为该任务的一个作业。它是实时调度和执行的基本单位。

到达时间：指作业建立起进程结构，但还没有转为就绪态成为调度对象的时刻。

释放时间：任务实例获得除 CPU 外的所有必要资源的状态称为就绪态。一个作业由其它状态转为就绪态的时刻称为它的释放时间。

执行时间：指作业需要 CPU 在不间断执行情况下的执行时间。

相对截止期：指作业从释放时刻到作业完成所允许的最大时间。

绝对截止期：绝对截止期是作业完成所允许的最晚时刻，绝对截止期=释放时刻+相对截止期。

响应时间：指作业从释放时刻到完成时刻之间的时间段长度。

最坏情况下执行时间：指任务的所有实例中执行时间的上界。在硬实时系统中，这是一个重要的参数，需要用到该参数来判定任务集的可调度性。

截止错失率：指在某个确定的时间段内，错失截止期的作业与实时作业总数之比。它常用来衡量调度算法的性能。

CPU 利用率：单个任务的 CPU 利用率为任务的最坏情况下执行时间裕任务周期之比，整个任务集的 CPU 利用率为所有任务的 CPU 利用率之和。

任务集超周期：指任务集中所有任务周期的最小公倍数。理想状态下，周期任务集在任意时刻 t 和$(t+k*H)(k\in Z)$的执行过程过程完全相同，其中 H 为任务集超周期。

2.2.4 实时调度算法分类

实时调度有多种分类方法，以下介绍五种常用分类。

1. 根据调度顺序产生的时机和方式可以分为静态调度和动态调度。

若在编译阶段就确定了任务调度的时序，则这样的调度算法为静态的。这种调度算法假设实时任务的很多参数已知，根据这些参数及某个调度规则脱机地进行可调度性分析与判断，最后产生一个任务调度表。系统运行时，调度器根据调度表选择任务运行。静态调度适合于问题需求比较容易确定，并且运行中不会有较大变化的情况，比如简单的工业过程控制。静态调度算法的优点是运行开销小，可预测性强，实现简单，没有过多的调度器时间开销。然而，它的缺陷是不能根据环境变化动态调整调度时序。一旦任务集或者任务参数发生变化，整个调度表将失效，需要重新分析、生成新的调度表。

如果调度器不是预先确定好各任务的运行时序，而是在运行期间根据某种调度规则临时决定选择某个就绪任务运行，那么这类调度称为动态调度。它根据当前就绪任务集的参数选择任务执行，不考虑可能即将到来的任务。这种调度算法能够对随机的外部事件做出很好的反应，因此适合于有任务动态生成的环境，如无人驾驶汽车、飞机的控制系统。动态调度算法的缺点是调度时间开销较大，可预测性较差。

2. 根据任务是否允许被其它任务抢占，分为可抢占式调度、非抢占式调度和抢占阈值调度。

如果满足某个调度条件时允许当前任务在被其它就绪任务抢占，当前任务转变为就绪任务，则这种调度算法称为可抢占式调度。抢占是调度的优点优先调度满足抢占条件的任务，资源利用率高。缺点是容易导致任务截止期错失率增大，而且任务切换开销不容忽视。

如果在调度中，当前任务不运行被其它任务抢占，任务一旦获得执行权，便会一直运行直到任务完成，则这种调度算法称为非抢占式调度。它比较适合于任

务运行时间较短、可以顺序执行的系统。非抢占式调度的优点是可预测性好，实现简单。缺点是不够灵活，资源利用率相对较低。抢占阈值调度除给每个任务分配优先级外，还分配一个抢占阈值，发生任务抢占必须满足抢占阈值条件。抢占阈值集中了抢占与非抢占调度的优点，可以保持抢占调度的响应时间小的优点，又能减少任务抢占次数，降低切换开销等。

3. 根据调度算法是否具有自适应性，分为自适应调度和非自适应调度。

自适应调度通常具有闭环结构，能够在运行中采样系统输出，经过对采样数据的加工和提炼，最后自动调整调度部件，以适应外部事件的动态变化。自适应调度利用了反馈控制系统的的优点，一般建立在启发式基础上，广泛应用于家用网络系统，车辆控制系统等嵌入式系统中。非自适应调度一般是开环结构，不会根据反馈信息调整调度部件。

4. 根据系统中处理器数量，可分为单处理器调度和多处理器调度。

若系统运行平台为为单处理器，则对应的调度为单处理器调度。单处理器调度考虑的问题相对较少。若系统运行平台为多处理器，则对应的调度为多处理器调度。尽管很多单处理器环境下的调度理论可以应用于多处理器环境，但是仍然具有很多新问题，如负载均衡，死锁等问题。

5. 根据调度器体系结构分为集中式调度和分布式调度。

这两种调度都存在于多处理器环境。在集中式调度中，有一个专门负责调度的处理器，将任务分配给其它处理器。而在分布式调度中，每个处理器都有调度功能，而且相互之间地位平等，各自调度各节点上的任务。可以通过远程过程调用（RPC）技术迁移任务到其它节点运行。

2.2.5 实时调度基本策略

主要的实时调度策略有三大类，分别是时间驱动调度、优先级驱动调度和共享驱动调度。本节主要介绍这三种策略。

1. 时间驱动调度

时间驱动调度[17~19]是指在系统运行前，根据实时任务的相关参数，设计并得到一张任务调度表；系统运行时，依据该调度表在每个调度决策时间点选择任务执行。周期性做出调度决策的调度器通常用到硬件定时器，在系统初始化后，调度程序根据调度表的决策时间点设置好定时器，当定时器到期，调度器被唤醒，执行调度。时间驱动是一种静态调度，其调度开销非常小，概念简单，可预测性好，对于实时任务参数非常清楚且稳定的情况下，时间驱动调度是很好的选择。它的缺点包括：作业的释放时间必须固定，所有周期任务组合必须事先预知，因此缺乏灵活性；另外，如果系统环境在运行时出现一些变化，就可能引起整个系统调度失败。

2. 优先级驱动调度

优先级驱动调度是实时系统中最常用的调度方式。相对于时间驱动，它是一种在线调度，不需要在系统运行前设计出任务调度表。实现方法是在作业创建后，根据某种优先级分配方案，为该作业分配一个优先级别，并加入一个优先级有序的就绪作业队列。在每个调度时间点，调度器选择具有最高优先级作业执行。不同的优先级设计方案对系统的性能有很大影响，根据作业的单一参数设计优先级的方法简单有效，并且容易实现。例如，有的调度依据作业的截止期设计优先级，有的调度根据任务的空闲时间设计优先级，还有的根据作业的周期长短设计优先级。还有的优先级根据两个甚至多个作业参数来设计优先级。例如，综合作业的重要度和紧急度设计作业优先级等。

基于优先级的调度方式又可以分为两大类：静态优先级调度和动态优先级调度。静态优先级调度中，作业的优先级从创建时刻到执行结束期间是保持不变的，而动态优先级调度中作业优先级在运行过程中会动态更新。静态优先级调度算法中经典的算法是速率单调（Rate Monotonic，RM）[20,21]算法和截止期单调（Deadline Monotonic，DM）算法。动态优先级调度中经典算法是最早截止期优先（Earliest Deadline First，EDF）[22]算法和最小空闲时间优先（Leats Lacking First）算法。静态优先级调度的优点是可预测性好，调度算法工作量比较小；缺点是不够灵活，缺乏对偶发事件的实时响应能力。相对于静态优先级调度算法，动态优先级有更大的灵活性，能够达到很高 CPU 利用率；缺点是调度算法复杂，可预测性较差。

3. 共享驱动调度

共享驱动调度[23~25]按照一定比例对一组需要调度的任务进行调度，任务的执行时间与其获得的比例成正比。系统中每个实时任务都可以确保其所要求的处理器资源，不会受到其它占用超出比例时间的任务的干扰。UNIX 采用的分数调度策略就属于共享驱动调度。共享驱动调度对于那些包含硬实时任务、软实时任务和非实时任务等多类应用能提供不同级别 QoS 保障，随着网络与多媒体技术的发展，共享驱动调度策略的应用越来越充分。共享驱动调度的缺点是任务的服务时间比例是决定其获得 CPU 资源的唯一因素，其他反映任务重要程度和紧急程度的因素都没有考虑在内，没有为各类应用提供有差别的 QoS。共享调度算法可以分为以下几类：轮转法、公平共享、公平队列、彩票调度法等。

2.3 经典实时调度算法

2.3.1 周期任务常用调度算法

由于在任务调度算法中，优先级驱动调度占绝大多数，因此本节介绍几种基于优先级的调度算法。比较经典静态优先级周期任务调度算法有 Liu 和 Layland

提出的速率单调算法和 Leng 提出的截止期单调调度算法；比较经典的动态优先级周期任务调度算法有 Liu 和 Layland 提出的最早截止期优先调度算法和最小空闲时间优先调度算法。

1. 速率单调调度算法（RM）

RM 调度算法是静态优先级调度算法中的最优算法之一，任意一种静态优先级调度算法可以调度的任务集，用 RM 调度算法也可以调度；用 RM 调度算法不能调度的任务集，用其它静态优先级调度算法也是不能调度的。RM 的优先级由任务的周期长短决定，周期越短，优先级越高，优先级一经分配就不再改变。RM 算法是为周期任务设计的，周期任务总是在下一周期作业到来之前完成当前周期作业，否则就会错过任务截止期。运行时，调度器总是选择优先级最高的就绪任务执行，具有高优先级的任务到来可以抢占低优先级的当前任务。实现中，将所有就绪任务按优先级从高到底排队，若当前运行任务完成完成或挂起，则把当前任务节点插入等待队列，把就绪任务队列中具有最高优先级的首节点从就绪队列摘下，交给 CPU 执行该任务；若在调度时间点有其它任务状态已变为就绪态，则需要和当前执行任务的优先级进行比较判断是否抢占当前任务，如果没有发生抢占，就将该就绪任务插入就绪队列适当位置。RM 算法实现简单，可预测性好，运行开销小，被广泛应用于要求高安全性周期任务系统中。RM 算法的缺点是灵活性不够，不能处理突发事件，并且其 CPU 利用率不高。

2. 截止期单调调度算法（DM）

截止期单调调度算法的调度策略与 RM 调度算法类似，是在 RM 调度算法基础上发展而来，其优先级按截止时间长度来分配，截止时间长度越短，优先级越高。DM 调度算法和 RM 调度算法具有类似的优点，对于任务截止时间小于或等于其周期的情况下，DM 调度算法为最优静态优先级调度算法。

3. 最早截止期优先调度算法（EDF）

EDF 调度算法是一种动态优先级可抢占调度算法，其优先级按照任务的截止时刻与当前时刻之间距离长短来分配，该段距离越短，优先级越高。EDF 调度算法是一种最优的单处理器动态调度算法，如果其他动态优先级调度算法能够调度的任务集，用 EDF 调度算法也可以调度；用 EDF 调度算法不能调度的任务集，用其它动态优先级调度算法必然也不能调度。EDF 调度过程中，若当前执行任务完成或挂起，则将当前执行任务插入等待队列，从按照优先级从高到低顺序排列的就绪任务队列中选择具有最高优先级的首节点执行；若有新任务到达就绪任务队列，首先更新所有就绪队列中任务优先级，然后判断新到达任务优先级是否高于当前执行任务，若是则抢占当前任务。EDF 调度算法的优点是其 CPU 利用率可达到 100%，灵活性好，能够很好地支持任务周期动态变化的系统；其缺点是调度开销比较大，对系统过载很难做出判断，并且某个任务错失截止期可能产生

多米诺效应影响很多其它任务也错失截止期。

4. 最小空闲时间优先调度算法（LSF）

LSF 调度算法和 EDF 调度算法一样，也是一种最优的动态优先级调度算法，CPU 利用率也可以达到 100%。它是对 EDF 调度算法的改进算法。EDF 算法保证的是截止期早得任务优先执行，但是这可能使其它任务到最后都发现不了其实在很早就已经不可能在截止期限内完成。LSF 调度算法正是从这点出发，通过计算任务的空闲时间来克服 EDF 调度算法的不足。任务空闲时间定义为任务截止期与当前时刻之间的时间长度减去任务剩余部分需要的执行时间。LSF 调度算法根据任务空闲时间来分配优先级，空闲时间越小，优先级越高。实现过程中，在每个时间片结束时都要更新任务的空闲时间、任务优先级，并且夭折空闲时间小于 0 的任务。若有高于当前执行任务优先级的任务到达，则抢占立刻当前任务的执行。LSF 调度算法能够很好的工作于 CPU 负载较高的实时系统中，其显而易见的缺点是由于频繁计算空闲时间和优先级而产生很大调度开销，并且由于任务的频繁切换而大大增加系统开销。

2.3.2 周期任务与非周期任务混合调度常用算法

非周期任务调度算法可以分为两类：基于服务器的算法和基于空闲时间的算法。基于服务器算法的基本思想是在保证满足硬实时周期任务满足截止期前提下，设置几个额外的周期任务使用指定的 CPU 带宽来处理非周期任务。基于空闲时间算法主要有时间挪用算法、时间片移位算法等，都是通过各种方法从周期任务调度的空隙获得尽可能多的处理器时间给非周期任务。

1. 基于服务器的调度算法

轮询（PS）算法是一种最简单的静态优先级服务器算法，实现中建立一个轮询任务以固定周期来处理等待处理的非周期任务。如果当前时刻没有非周期任务，则该任务挂起，这段 CPU 时间被放弃。

Lehoczky 提出了 DS（Deferrable Server）与 PE（Priority Exchange）算法来减低非周期任务响应时间，基本思想和 PS 算法一样，只是 DS 算法和 PE 算法保留了为处理非周期任务准备而该时刻没有非周期任务的处理器时间。PE 算法和 DS 算法的区别就在于保留该部分时间的方式不同。

DS 算法使用一个固定的高优先级周期任务作为服务器为非周期任务提供服务，在该任务周期内维护其非周期执行时间 C，只要 C 没有被用完，非周期任务就能够在这个周期的任何时刻执行，在每个服务器周期开始时刻，C 被累计到其总的可使用处理器时间。PE 算法通过与较低优先级周期任务交换来保持其全部处理器时间。服务器周期到达时，如果这个时候有挂起的非周期任务，就处理非周期任务；否则选择一个高优先级周期任务（优先级低于服务器任务优先级，因为

服务器任务优先级为最高优先级）来执行，而非周期任务的执行时间被积累到该周期任务的优先级上。结果，这个周期任务得到优先执行，非周期任务的执行时间也没有丢失。这种优先级交换将持续到服务器周期结束。

在 DS 算法和 PE 算法基础上，Sprunt 等提出一种 SS 算法[26]。SS 算法也要定义一个非周期执行时间 C，但不是在每个服务器周期的开始时刻累计到总的非周期任务处理器时间。而是按照一定规则进行。SS 算法比 PE 算法简单，不需要在每个优先级层次上维护其执行时间。SS 算法比 DS 算法有更高的 CPU 利用率，因为 SS 服务器推迟执行时间的同时也推迟其执行时间的补充。

2. 基于空闲时间的调度算法

在某个时间段上的空闲时间等于能够用来执行非周期任务而不会导致周期任务集不可调度的总时间。Lehoczky 和 Ramos-Thuel 最早提出固定优先级调度中空闲时间挪用算法（SSA）来处理软截止期的非周期任务。系统运行时，当有非周期任务到达，则利用所有可用的空闲时间执行该任务；若所有可用空闲时间用完而该任务没有完成，则该任务被挂起直到系统中出现可用空闲时间。SSA 算法相比于 DS 和 SS 算法有很大改进，随着周期任务负载增加 SSA 算法性能变化不大。为了解决硬截止期周期任务与软截止期非周期任务的混合调度问题，Ismeal Ripoll 等将 SSA 算法扩展到动态优先级抢占系统，提出 EESS（EDF exact slack stealer）算法。它为每个非周期任务分配一个截止期，和周期任务一样被调度。从周期任务的可调度性和非周期任务的响应时间来说，EESS 算法是最优的。

Fohler 针对静态调度中混合调度问题，提出时间片移位（slot shifting）算法。它首先为周期任务构造一个静态调度表，满足任务的时间、同步与通信要求；接着离线计算空闲处理时间的大小与分布，记录每个空闲处理时间的位置以及随后的周期任务执行容许的偏移时间。由于所有未用的处理时间的数量与分布是离线确定的，只需在线地增量维护，因此运行机制简单且内存需求较少。

SSA 算法是一种利用空闲时间的有效方法，但是实现较困难，并且时间与空间需求昂贵。时间片移位算法指适用于静态调度系统，存在的问题与 SSA 算法类似。

2.4 任务集可调度性分析

任务可调度性判定是实时系统调度理论研究的核心问题之一[27~29]。本节介绍几种算法可调度性判定方法。

2.4.1 RM 算法的可调度性判定

1. RM 调度模型

任务集 S 是由 N 个周期性任务 τ_i（$1 \leqslant i \leqslant N$）构成，任务 τ_i 由一个六元组（T_i，

C_i，D_i，P_i，U_i）所确定。其中：

（1）T_i表示任务的周期。

（2）C_i表示任务的最坏情况下执行时间。

（3）D_i表示任务的绝对截止期。

（4）P_i表示任务的优先级。约定优先级越大，优先级越高。

（5）U_i表示任务τ_i的 CPU 利用率。

整个任务集 CPU 利用率 U 定义为：

$$U=\sum_{i=1}^{n}U_i=\sum_{i=1}^{n}(C_i/T_i) \tag{2.1}$$

假设 $D_i=T_i$，任务之间相互独立，任务切换时间忽略不计。

Liu 和 Layland 证明了 RM 算法是最优的静态优先级调度算法，证明思想如下：假设任务集采用其它静态优先级算法可调度，设τ_i和τ_j是其中两个优先级相邻的任务，$T_i>T_j$ 而 $P_i<P_j$，将τ_i和τ_j优先级互换，可以证明这时的任务集仍然可以调度。这样其他任何静态优先级调度最终都可以转换为 RM 调度，即 RM 算法是最优的。

2. Liu 和 Layland 的 RM 算法可调度性判定

定理 **2.1** 对于给定具有 n 个任务的任务集，如果任务集 CPU 利用率满足下面条件：

$$U<n(2^{1/n}-1)=L(n) \tag{2.2}$$

则该任务集是可调度的。L（n）表示 n 个任务的 CPU 利用率最小上界，它只和任务数 n 有关，极限值约为 0.69。定理 2.1 是 RM 算法可调度的充分条件而不是必要条件。

例 2.1 任务集 S 包括下面三个任务：τ_1：C_1=40，T_1=100；τ_2：C_2=50，T_2=250；τ_3：C_3=100，T_3=400。该任务集的 CPU 利用率 U=0.825>L（3），根据定理 2.1 无法判定其可调度性。而实际上该任务集是可调度的，在第一个超周期的调度情况如图所示。

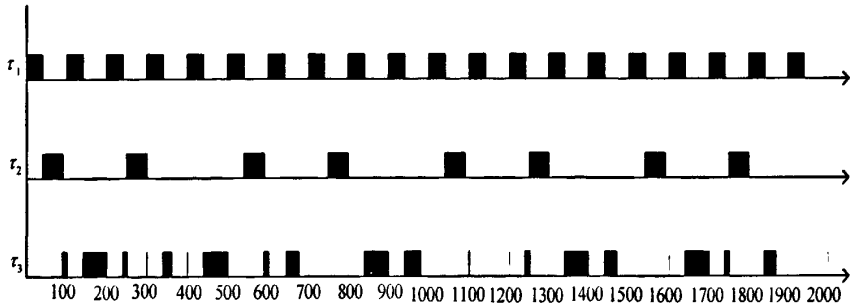

图 2.2 任务集 S 调度示意图

3. RM 算法可调度性充要条件

大量事实证明，在 RM 调度应用中，CPU 利用率在 90%左右的可调度任务集不少，这说明定理 2.1 的应用存在很大局限性。Lehoczky 等人对静态优先级算法特征进行更精确研究后，提出了 RM 算法可调度性判定的充要条件。

对于具有 n 个任务的任务集，用

$$W_i(t)=\sum_{j=1}^{i}\lceil t/T_i\rceil C_j \tag{2.3}$$

表示该任务在时间段[0，t]内前 i 个任务对 CPU 的累计时间需求量。定义 L_i（t）与 L_i 如下：

$$L_i(t)=W_i(t)/t$$

$$L_i=\min_{\{0<t\le T_i\}} L_i(t) \tag{2.4}$$

$$L=\max_{\{1\le i\le n\}} L_i \tag{2.5}$$

定理 2.2 对于具有 n 个任务的任务集，第 i（$1\leqslant i\leqslant n$）个任务 τ_i 在在 RM 算法下可调度，当且仅当 $L_i\leqslant 1$，该任务集在 RM 算法下可调度，当且仅当 $L\leqslant 1$。从公式 2.1 和 2.2 可以看出，在[0，T_i]的任一子区间[a，b]中（a，b 是 $\{kT_j \mid j=1,\ldots,i;k=1,\ldots,\lfloor T_i/T_j\rfloor\}$ 中两个值相邻的数），W_i（t）先不变后在该区间右端点跳跃到一个更高值，t 在整个区间单调递增，因此 L_i（t）在该区间是单调递增的，最后在右端点跳到一个更高值。即当 t 是 T_j（$1\leqslant j\leqslant i$）的倍数是，L_i（t）具有局部最小值。因此要求 L_i（t）的最小值，只需比较下面这些离散点处的值。令

$$E_i=\{kT_j \mid j=1,\ldots,i;k=1,\ldots,\lfloor T_i/T_j\rfloor\} \tag{2.6}$$

则

$$L_i=\min_{\{t\in E_i\}} L_i(t) \tag{2.7}$$

根据公式 2.5、2.6 可以判断上面例 2.1 中任务集 S 是可调度的。

2.4.2 EDF 算法可调度性判定

Liu 和 Layland 证明了对于具有 n 个任务的周期任务集，用 EDF 算法可以调度，当且仅当任务集 CPU 利用率 U 满足

$$U=\sum_{i=1}^{n}(C_i/T_i)\le 1 \tag{2.8}$$

这是 EDF 算法可调度的充分必要条件。在不考虑释放抖动（即任务到达但延迟一段时间后才转为就绪状态），任务切换时间等实际情况时，该判定定理很有效。

2.4.3 静态优先级周期任务调度算法可调度性判定

J.Lehockzky 首先提出 level-i 忙周期[30]概念，用于静态优先级的任务集可调度性判定。level-i 忙周期指的是在某个时间区间[t_1，t_2]内，只有优先级大于等于 i 的任务的执行。且在[t_1-δ，t_1]或[t_2，t_2+δ]时间范围内没有优先级大于等于 i 的任务释放（即任务就绪），其中 δ 是大于 0 的极小值。忙周期分析通过模拟最坏情况下任务的调度情况，计算出各任务实例的最坏情况下的响应时间。对于给定的任务集，Lehockzky 的可调度性判定从 0 时刻起的第一个 level-i 忙周期内所有任务实例，如果存在某个任务实例的最坏响应时间超过其截止期，则该任务集为不可调度的，否则为可调度的。用忙周期分析方法计算任务的最坏情况下响应时间公式如下：

$$L^{(m+1)} = \sum_{\forall j, P_j > P_i} \left\lceil \frac{L^{(m)}}{T_j} \right\rceil \cdot C_j \tag{2.9}$$

其中

$$L^{(0)} = \sum_{\forall j, P_j > P_i} C_j \tag{2.10}$$

$$S_i(k) = (k-1)\cdot C_i + \sum_{\forall j, P_j > P_i} (1 + \left\lfloor \frac{S_i(k)}{T_j} \right\rfloor) \cdot C_j \tag{2.11}$$

$$F_i(k) = S_i(k) + C_i + \sum_{\forall j, P_j > P_i} (\left\lceil \frac{F_i(k)}{T_j} \right\rceil - (1 + \left\lfloor \frac{S_i(k)}{T_j} \right\rfloor)) \cdot C_j \tag{2.12}$$

$$R_i = \max_{k \in [1,\ldots,m]} (F_i(k) - (k-1)\cdot T_i) \tag{2.13}$$

公式 2.9 计算第一个忙周期长度，递归计算初值由公式 2.10 得到。公式 2.11 计算任务 τ_i 的第 k 个作业的开始时间，在它开始执行前，所有[0，S_i（k）]之间就绪的高优先级任务必须在 S_i（k）前结束，S_i（k）的初值为：

$$kC_i + \sum_{\forall j, P_j > P_i} C_j \tag{2.14}$$

公式 2.12 计算任务 τ_i 的第 k 个作业的结束时间，F_i（k）的初值为：

$$S_i(k) + C_i \tag{2.15}$$

计算 F_i（k）直到满足：

$$F_m(k) \le kT_i \tag{2.16}$$

同时，S_i（k）也停止计算。

公式 2.13 计算任务 τ_i 的最坏响应时间，当且仅当

$$R_i \le D_i \tag{2.17}$$

任务集合可调度。

2.5 本章小结

本章主要介绍了实时调度相关理论背景知识，包括实时系统基础知识、实时调度基本理论、经典实时调度算法及任务集可调度性分析几个内容。由于实时任务调度算法及任务集的可调度性判断在实时系统中的重要地位，本章着重介绍了实时调度相关内容，特别是对周期任务与非周期任务混合调度方面做了比较系统的归纳。

第 3 章 硬实时周期任务与偶发任务混合调度的可调度性判定

3.1 引言

硬实时周期任务与偶发任务的混合调度是实时系统中的研究重点，已有的研究集中于充分保证硬实时周期任务满足截止期的基础上尽量减少偶发任务的响应时间，而对如何保证偶发任务的截止期的研究很少。本章基于“调度”与“逆调度”概念，根据不同偶发任务特征设置动态可挪用时间点 $T_{dynamic}$，得到偶发任务可调度性判定的一个充分条件，最后提出调度偶发任务的可调度性判定算法 IISS（Improved Idle Slack Stealing）。

3.2 系统模型

本章讨论的硬实时任务混合调度使用以下系统模型：

（1）单处理器环境；

（2）任务间可抢占，忽略上下文切换和调度开销；

（3）任务间相互独立；

（4）除非被抢占，否则任务不会自挂起。

系统中周期任务集由 N 个周期任务 τ_i（$1 \leqslant i \leqslant N$）构成，任务 τ_i 由一个四元组（O_i，T_i，C_i，D_i）所确定。其中：

（1）O_i 表示任务的起始释放时间；

（2）T_i 表示任务的周期；

（3）C_i 表示任务的最坏情况下执行时间；

（4）D_i 表示任务的相对截止期。

硬实时周期任务满足以下假设：

（1）O_i=0，即周期任务都在 0 时刻释放；

（2）D_i=T_i，即周期任务必须在它的下一作业到来前完成；

（3）U<1，即系统除了可以调度周期任务，还有空闲的处理器时间。

系统中偶发任务集由 M 个偶发任务 s_i（$1 \leqslant i \leqslant M$）构成，任务 s_i 由一个三元组（S_i，R_i，D_i）确定，其中：

（1）S_i 表示第 i 个偶发任务的到来时间；

（2）R_i 表示第 i 个偶发任务的最坏情况下执行时间；

（3）D_i表示第 i 个偶发任务的相对截止期。

3.3 问题描述

硬实时周期任务和硬实时偶发任务是实时系统中常见的两种任务类型，它们的混合调度可判定条件是本章研究重点。首先假设周期任务采用 EDF 算法是可调度的，然后使用硬实时周期任务调度过程中的空闲时间和可挪用时间来调度偶发任务。可挪用时间是在保证不影响周期任务调度性的前提下，为保证偶发任务的截止期而挪出的最大时间。如果当前时刻到来的偶发任务不经过计算验证就插入偶发任务调度队列等待调度，可能会使之前到来的偶发任务错过截止期，导致调度混乱。因为当前时刻到来的偶发任务的截止期可能小于之前到来的偶发任务，刚到达的偶发任务会推迟它们的执行，而这样很可能影响整个任务集的可调度性。所以，我们需要找到判定当前到达的偶发任务是否可以加入偶发任务调度队列参与调度而不影响整个任务集可调度性的条件，以保证偶发任务集的所有任务满足截止期。

挪用法[29]是混合任务调度中的一种有效算法，它的基本思想是在保证硬实时周期任务满足截止期的同时，尽量推迟其开始执行时刻而空出处理器时间，尽量获得周期任务集“挪出”的时间。调度器可以利用这些空出的时间执行硬实时偶发任务。

例 3.1 周期任务集中只有两个任务 τ_1=（0，1，3，3），τ_2=（0，1，4，4），偶发任务集中只有一个任务 s=（3，1，1），s 在时刻 3 到达，这时 τ_1，τ_2 的第二个作业都可以推迟 1 个时间单位执行，s 可以在时间区间[3，4]内完成。这就使 τ_1、τ_2 和突然到来的任务 s 都满足截止期。调度示意图如图 3.1：

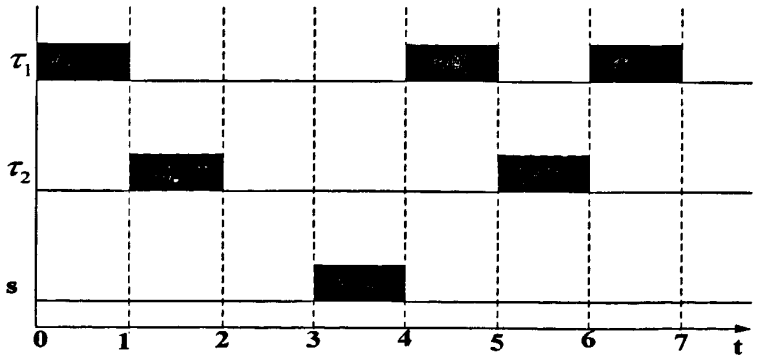

图 3.1 时间挪用法调度示意图

3.4 EDF 算法调度中空闲时间分布

空闲时间分布指在任意一段时间内的空闲时间分布情况。文献[2]通过模拟运行一个超周期时间，给出了一种获得空闲的算法。但是这种算法具有很大局限性。

因为周期任务每次执行时间可能不同，当在这个超周期中的执行时间偏离它的最大执行时间较大时，得到的空闲时间就会有较大偏差。文献[2]提出下面的某个超周期内空闲时间分布计算算法。

因为 EDF 算法在 CPU 空闲时，只要有任务到达就立即执行，所以如果存在空闲时间，那每一段空闲时间肯定都是在周期任务到来时刻结束。这样就可以按照在超周期中每个任务作业到达的时间顺序构造一个到达时间向量。

定义 3.1 某个超周期内到达时间向量 $U=\{u_0, u_1, \ldots, u_i, u_{i+1}, \ldots, u_p\}$。其中 $u_0=0$，$u_p=H$，$u_i \leqslant u_{i+1}$（$i=0, \ldots, p$），$p \leqslant N-n+1$，H 表示超周期长度，其值为各周期任务的最小公倍数，其中

$$N=\sum_{i=0}^{n}\frac{H}{T_i} \tag{3.1}$$

表示在一个超周期内累计任务到达次数。

定义 3.2 某个超周期内空闲时间向量 $F=\{f_0, f_1, \ldots, f_i, f_i+1, \ldots, f_p\}$。其中 f_i 为 u_i 时刻前的空闲时间长度。f_i 的递推计算公式如下：

$$f_i=\max(0, e_i-\sum_{j=1}^{n}\left\lfloor\frac{e_i}{T_j}\right\rfloor\cdot C_j-\sum_{k=0}^{i-1}f_k), (i=1,2,\ldots,p) \tag{3.2}$$

$$f_0=0$$

例 3.2 周期任务集中两个任务 $\tau_1=(0, 1, 3, 3)$，$\tau_2=(0, 1, 5, 5)$，空闲时间向量分量 $f_2=1$，$f_3=1$，$f_5=2$，如图 3.2。

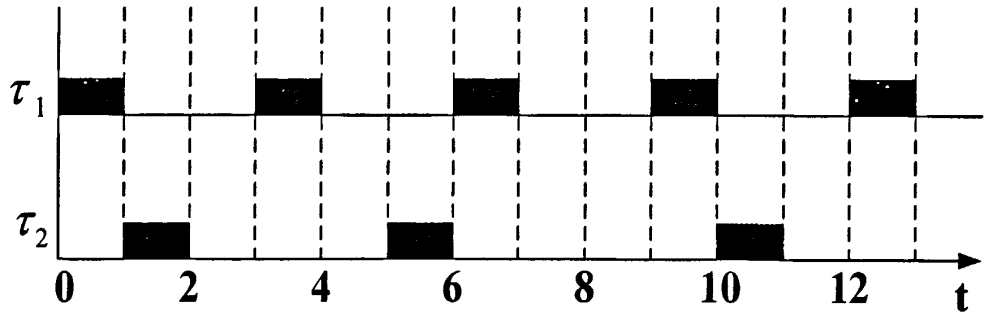

图 3.2 空闲时间向量示意图

定义 3.3 某个超周期内空闲时间分布函数 F（t_1，t_2）为时刻 t_1 和 t_2 之间的处理器空闲时间。

某个超周期中时间区间[t_1，t_2]内的空闲时间按照如下方法计算：首先在到达时间向量 U 中查找出在 t_1 时刻后最近的一个任务到达时间点 u_i，得到对应的空闲时间段 f_i；然后依次向后遍历向量 U 和 F 直到 u_i 超过 t_2 时刻。最后将得到的空闲时间段累加得到 t_1 和 t_2 之间的空闲时间。

3.5 EDF 算法调度中可挪用时间

1. EDF 算法调度与 EDF 算法逆调度

EDF 算法调度和 EDF 算法逆调度是为描述周期任务集在超周期中的执行情况而引入的概念。

定义 3.4 执行区间：某任务在时刻 s 得到处理器，在 e 时刻让出处理器，若在（s，e）区间该任务未被其它任务抢占，则时间区间（s，e）称为该任务的一个执行区间。

定义 3.5 作业执行区间集 S_{ij}：任务 τ_i 的第 j 个作业从开始执行到完成所有执行区间的集合。S_{ij}={（s_1，e_1），…，（s_r，e_r）}，其中（s_r，e_r）为作业的最后一个执行区间。

定义 3.6 任务执行过程 S_i：任务 τ_i 的所有作业的执行区间集，S_i={ S_{i1}，…，S_{im} }，m=H/T_i。

定义 3.7 EDF 算法调度 S：EDF 算法调度 S 记录了使用 EDF 调度算法在超周期内具有 n 个周期任务的周期任务集的调度情况，是任务集中所有周期任务的执行过程集合，S={S_1，…，S_n}。

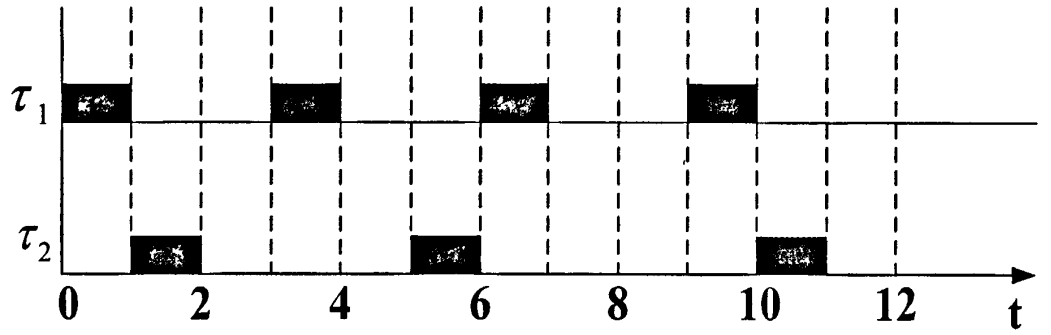

图 3.3 EDF 算法调度

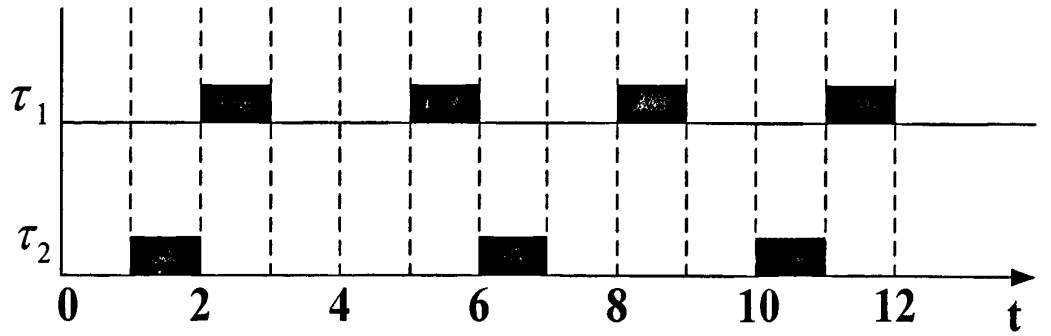

图 3.4 EDF 算法逆调度

定义 3.8 S（t）：S（t）为调度 S 在时刻 t 以前执行完的任务执行区间集合。

例 3.3 在例 3.2 中，当 t=12 时，S(12)={{ S11 ， S12， S13， S14 }，{ S_{21} ，S_{22}，S_{23}}}={{{（0，1）}，{3，4}，{（6，7）}，{（9，10）}}，{{1，2}，{5，6}，{10，11}}}.

定义 3.9 EDF 算法逆调度 S^{-1}：用于记录周期任务集在超周期中的调度情况，S^{-1} 是相对 S 来说的。对于任何属于 S^{-1} 的任务执行区间（s，e），存在且仅存在一个对应的任务执行区间（H-e，H-s）属于 S。

由定义 3.7 可知，EDF 算法逆调度 S^{-1} 也有执行区间、作业执行区间集、任务执行过程概念。

例 3.4 周期任务集中两个任务 τ_1=（0，1，3，3），τ_2=（0，1，5，5），其 EDF 算法调度 S 与 EDF 算法逆调度 S^{-1} 分别如图 3.3、图 3.4 所示。

定义 3.10 S^{-1}_{ij}- S_{ij}（t）：记 S^{-1}_{ij} ={[s_1，e_1），…，[s_k，e_k），[s_{k+1}，e_{k+1}），…，[s_q，e_q）}，S_{ij}（t）={[s'_1，e'_1），…，[s'_p，e'_p）}，其中 $e'_p \leqslant t$；其中 $e_k \leqslant s_{k+1}$，

$$len(S_{ij}(t)) = \sum_{i=1}^{p}(e'_i - s'_i)$$

则

$$S^{-1}_{ij} - S_{ij}(t) = \begin{cases} \varnothing, 当 \sum_{i=1}^{p}(e'_i - s'_i) = C_i, \\ \{[s_{k+1}, e_{k+1}), \ldots, [s_q, e_q)\}, 当 \\ \quad \sum_{i=1}^{p}(e'_i - s'_i) < C_i 且 \sum_{i=1}^{p}(e'_i - s'_i) = len(S_{ij}(t)). \end{cases} \tag{3.3}$$

其中 C_i 为任务最坏情况下执行时间。S^{-1}_{ij}- S_{ij}（t）是从 S^{-1}_{ij} 中按照时间先后移走等量于任务作业 S_{ij} 在时刻 t 以前已经执行完成的时间段。

定义 3.11 S^{-1}-S（t）：S^{-1}-S（t）={{…，S^{-1}_{1p}- S_{1p}（t），…}，…，{…，S^{-1}_{nq}- S_{nq}（t），…}}，（$1 \leqslant p, q \leqslant k$，k=H/T）。

S^{-1}-S（t）给出的是一种调度形式，在 t 时刻前按照 EDF 算法调度，在时刻 t 之后按照延迟任务执行的调度方式调度任务。

例 5. 周期任务集中两个任务 τ_1=（0，1，3，3），τ_2=（0，1，5，5），则其对应的 S^{-1}-S（5）调度如图 3.5 所示。

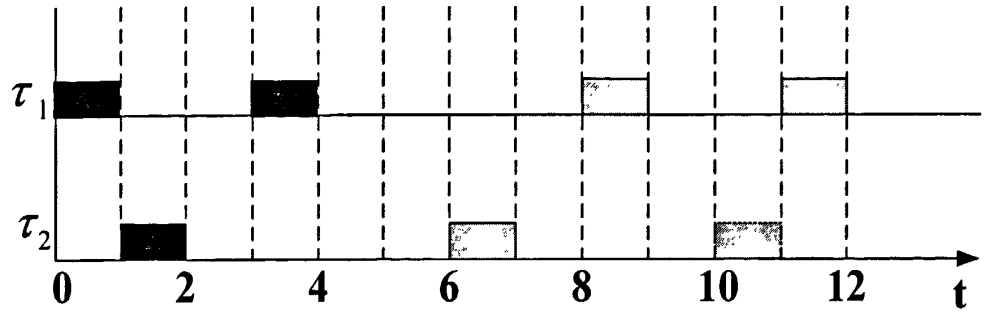

图 3.5 S^{-1}-S(5)调度示意图

3.6 EDF 算法调度中的最大可挪用时间

定理 3.1 EDF 算法调度 S 的任何执行区间都不能前移，EDF 算法逆调度 S^{-1} 的任何执行区间不能后移。

证明：首先证明 EDF 算法调度 S 的任何执行区间都不能前移，用反证法证明。假如存在某个执行区间[s，e]可以前移一段长度δ的情况，如图 3.5 所示，

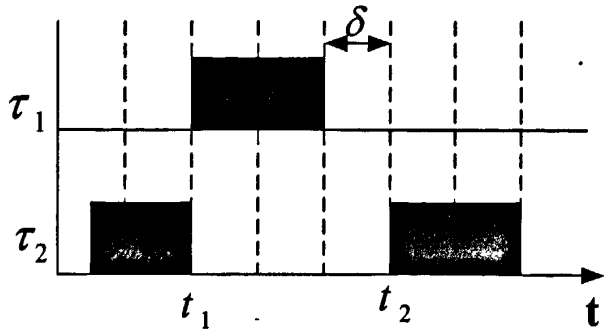

图 3.6 EDF 算法调度区间不能后移

假如 t_2 时刻执行的任务是刚到达的新的任务作业，新作业到达立即开始执行，不能在未到达就开始执行，所以不能前移执行任务。那么 t_2 时刻执行的作业必然是前面没有执行完的任务部分，那么该任务肯定在 t_1 时刻之前到达，已知 t_2 时刻前存在一段长度δ的空闲时间段，然而根据 EDF 算法调度可知，在有任务到达并且没有执行完的情况下，不可能出现空闲时间δ，产生矛盾。因此命题 EDF 算法调度 S 的任何执行区间都不能前移得证。同理可证 EDF 算法逆调度 S^{-1} 不能后移。证毕。

根据上面定理，EDF 算法逆调度 S^{-1} 中所有任务执行区间不可后移，但将任务执行区间从 S^{-1} 移动到 S（t）得到 S^{-1}-S（t）后，S^{-1}-S（t）的所有执行区间不一定不能向后移动。例 3.5 中的 S^{-1}-S（5）存在任务执行区间[6，7]可以向后移动两个时间单位。那么，可以通过一定操作将 S^{-1}-S（t）改进为满足定理 3.1。

分析将 S^{-1} 中某任务作业 S_{ij} 的一段任务执行区间[a，b）从 S^{-1} 前移到 S（t）的过程，假设 S_{ij} 的到达时刻为 s_1，绝对截止期 d_1，S_{pq} 的到达时刻为 s_2，绝对截止期为 d_1，则这两个任务作业之间的相对关系总共有四种情况：

1）$s_1 < s_2$，$d_1 < d_2$；2）$s_1 > s_2$，$d_1 < d_2$；3）$s_1 < s_2$，$d_1 > d_2$；4）$s_1 > s_2$，$d_1 > d_2$；

在第 1），3）两种情况下，S_{ij} 的到达时刻早于 S_{pq} 的到达时刻，因此前移任务执行区间[a，b）不会使 S_{pq} 的任务执行区间后移。在第 4）种情况下，S_{pq} 的的到达时刻和绝对截止期都早于 S_{ij} 的到达时刻和绝对截止期，等到 S_{ij} 开始执行时，S_{pq} 的所有任务执行区间都移到了 S（t）。在第 2 种情况下，前移区间[a，b）后，S^{-1} 中 S_{pq} 的剩余部分可以向后移动。这时可以通过将 S_{pq} 的剩余部分向后移动长度（b-a）来填充原来的区间[a，b）。这样得到的 S_{pq} 剩余部分是不能后移的。示意图如下：

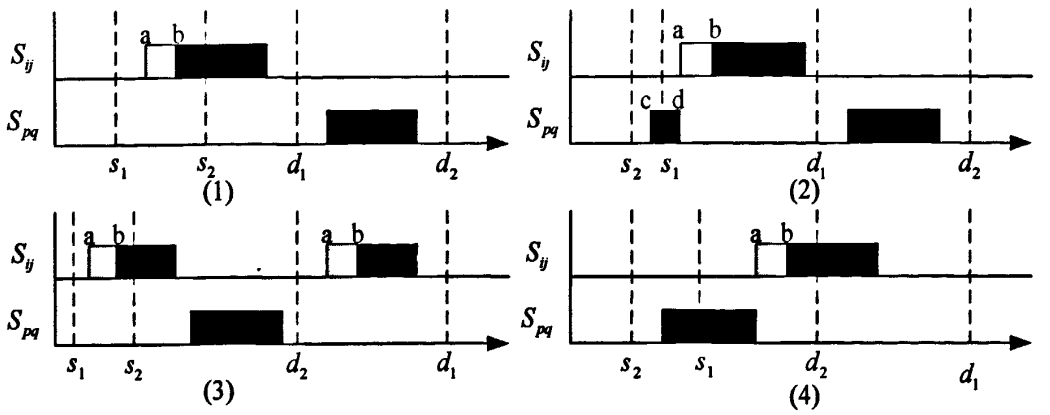

图 3.7 任务到来之间的四种关系

定义 3.12（$S^{-1}(t)$）：$S^{-1}(t)$ 与 S^{-1}_{ij}- $S_{ij}(t)$ 都是从 S^{-1} 中移走等量于在 t 时刻前已经完成的执行区间，不同的是对 t 时刻后的调度进行了额外的调整。具体规则如下：

假设[a，b）为作业 S_{ij} 将要执行的时间段，$[a^{-1}, b^{-1}) \in S^{-1}_{ij}$，$[a^{-1}, b^{-1})$ 为按 S^{-1}-S（t）定义将 S^{-1} 移出的时间段，$[a^{-1}, b^{-1})$ 从 S^{-1} 移到 S（t）中后成为[a，b）。S_{pq} 代表还未完全从 S^{-1} 移动到 S（t）的作业，按照如下方法移出[a-1，b-1）：

若 S^{-1}_{ij} 的到达时间早于所有的 S^{-1}_{pq}，则直接从 S^{-1} 中移出$[a^{-1}, b^{-1})$；

否则，在移出[a-1，b-1）后，将 S^{-1}_{pq} 的任务执行区间中结束时间早于时刻 a^{-1} 的任务执行区间向后移动长度（b-a）。

定理 3.2 假设 Q 为一个超周期内 t 时刻前的已挪用时间和 CPU 空闲时间和，s 为 S^{-1}（t）为 t 时刻后第一个任务执行区间的开始时间，则（s-t-Q）为周期任务集在 t 时刻的最大可挪用时间。

证明：由 S^{-1}（t）的任务执行区间不能后移，且 s 为 S^{-1}（t）的 t 时刻后第一个执行区间的开始时间，则在 t 时刻，s 达到最大值，即（s-t-Q）为最大可挪用时间。证毕。

定理 3.3 使用 EDF 算法时，系统初始时刻的最大可挪用时间为超周期中最后一个空闲时间向量中最后分量 fp。

证明：由定理 3.2，此时 t=0，Q=0，s 为 EDF 算法逆调度的第一个执行区间的开始时间，也就是 EDF 算法调度最后一个空闲时间区间，因此（s-t-Q）= fp。证毕。

定义 3.13 时间区间$[t_1, t_2]$内最大可挪用时间函数 $G(t_g, t_1, t_2)$：假设必须在 t_g 时刻之后才能挪用时间，在保证周期任务可调度的前提下，时间区间$[t_1, t_2]$内最大可挪用时间。

如果 $t_g > t_1$，则 $t_1=t_g$。如果 $t_2>H$，则 $t_2=H$，其中 H 为超周期长度。

$[t_1, t_2]$区间的最大可挪用时间，可以按如下步骤计算：

1. 查找 t_1 时刻后第一个任务到达时间点 u_i 及对应的空闲时间段 f_i；

2. 若 $t_1 \geq E(i)-F(i)$，说明 t_1 时刻在空闲时间段中，则最大可挪用时间 $g=f_p+(E(i)-t_1)$；否则 $g=f_p-(F_0(i)-F(i))$，其中 $F_0(i)$ 为之前没有挪用过时间情况下的初始空闲时间；

3. $F(j)=F(j)-g$，$j \geq i$。如果调整后的 $F(j) \geq 0$ 就可以不用调整其后面的空闲时间向量；如果 $F(j)<0$，那么下一个空闲时间向量应该为 $F(j+1)=F(j+1)+F(j)$。

4. 转步骤 1 求得 u_{i+1} 任务实例到达时间时的可挪用最大时间，直到超出时刻 t_2；

5. 累加所有的 g 得到$[t_1, t_2]$区间总的最大可挪用时间。

3.7 IISS 算法

3.7.1 硬实时周期任务与偶发任务混合调度的可调度性判定分析

首先考虑只有一个偶发任务（S，R，D）和周期任务集$\Gamma=\{\tau_1,\tau_2,...,\tau_n\}$在第一个超周期内混合调度的情况，其中$\tau_i=(O_i, T_i, C_i, D_i)$，$(1 \leqslant i \leqslant n)$。偶发任务始终不允许影响周期任务的可调度性，它只能利用空闲时间和可挪用的时间来执行。文献[2]提出一种 ISS 算法，给出判定偶发任务可调度性的充分条件如下：

$$F(S,t_{\min})+G(0,t_{\min},S+D) \geq R \tag{3.4}$$

其中

$$t_{\min}=\min(t_1,t_2,...,t_n) \tag{3.5}$$

$$t_i=\left\lfloor \frac{S+D}{T_i} \right\rfloor \cdot T_i (i=1,2,...,n) \tag{3.6}$$

其主要思想是偶发任务在时间区间（S，t_{min}）内只能在空闲时间运行，在时间区间（t_{min}，$S+D$）内，偶发任务将按照 EDF 调度算法抢占周期任务的执行，其调度示意图如图 3.8，图中假设 $t_{min}=t_n$。

该算法尽量推迟偶发任务的抢占时间点，这对偶发任务的平均响应时间及偶发任务判定预测准确率（计算方法在仿真实验中说明）会有影响，因为会出现在时间区间（t_{min}，$S+D$）内来不及挪用足够时间满足偶发任务的执行需求而实际上在 S+D 后存在足够满足偶发任务的空闲时间的情况，特别是在 EDF 算法逆调度中初始空闲时间很大的情况下，这个问题的影响会得到放大。

考虑到上面的问题，为了让偶发任务尽快完成，假设从偶发任务到达时刻起就开始尽量挪用时间来执行，这种方法叫最大挪用时间法，示意图如图 3.9。

如果满足在在区间（S，$S+D$）内的最大可挪用时间大于或等于 R，则可以判

定该偶发任务可调度，即偶发任务可调度性判定的充分条件是：

$$G(0,S,S+D)\geq R \tag{3.7}$$

这样设计的优点是偶发任务响应时间小，在文献[2]中已经证明，缺点是在偶发任务到达时间间隔比较大时，会出现后来任务截止期排在前面，而截止期靠后的任务却先挪用了前面的空闲时间的情况，从而影响判定算法的预测准确率及稳定性，文献[2]的实验数据说明了这个问题。

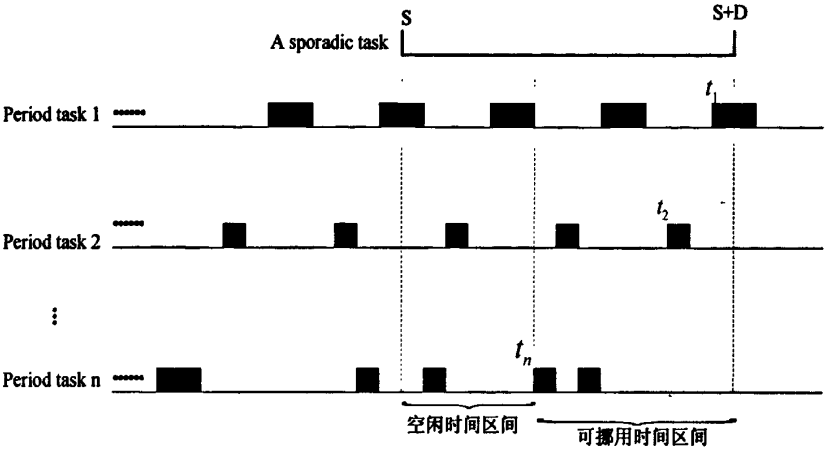

图 3.8 ISS 算法的偶发任务调度时间分配

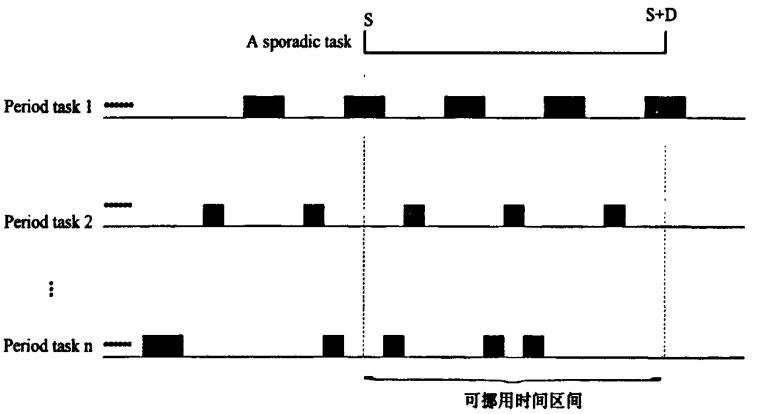

图 3.9 最大挪用时间法偶发任务调度时间分配

3.7.2 IISS 算法设计

综合以上两种特殊情况，我们可以设计一个动态的挪用时间起点 $T_{dynamic}$，将时间区间（$T_{dynamic}$，$S+D$）作为可挪用的时间区间，根据不同的偶发任务集属性参数选择合适的 $T_{dynamic}$，以获得更好的可调度性判定的预测准确率及响应时间。考虑单个偶发任务时，调度时间分配如图 3.10，可调度性判定的充分条件为：

$$F\left(S,T_{dynamic}\right)+G(0,T_{dynamic},S+D))\geq R \tag{3.8}$$

现在考虑有 m 个偶发任务Π=（ r_1， r_2，...， r_m）和 n 个周期任务混合调

度的情况，其中 r_i=（ S_i，R_i，D_i）。按照偶发任务截止期从小到大对 m 个偶发任务排序，放到一个队列中，调度器按照这个顺序调度偶发任务。要判定某偶发任务集的可调度性，可以依次对偶发任务使用可调度性判定条件进行测试，一旦出现不满足判定条件的情况，则说明该偶发任务集不可调度，返回；否则根据当前判定的偶发任务占用的空闲时间和挪用时间调整空闲时间向量 F。

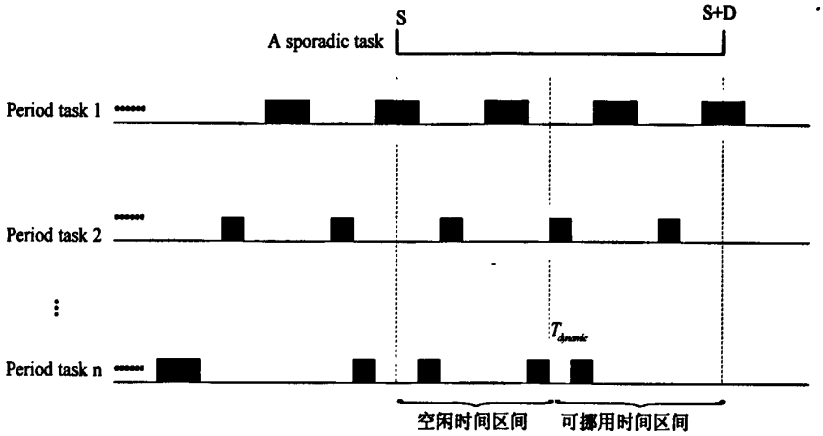

图 3.10 IISS 算法偶发任务调度时间分配情况

假设偶发任务随机到达，在 t 时刻有一个偶发任务到达，在 t 时刻之前偶发任务调度队列中还剩下 k-1（k>0）个任务没有完成。对这 k-1 个任务分别分配一个空闲时间向量 F_i，$1 \leqslant i \leqslant k$-1，用来保存运行完第 i 个任务后剩余空闲时间向量；同时分配变量 et_i 保存第 i 个任务的完成时刻。令 F_i（x）表示空闲时间向量 F_i 的第 x 个分量，G_i（et_i，t_1，t_2）表示在对应于 F_i 的最大可挪用时间。判定 t 时刻是否接受刚到达的任务的判定算法如下：

（1）将到达的偶发任务 r 按照截止期递增顺序插入偶发任务调度队列，设该任务在队列中的位序为 j。

（2）若整个超周期内空闲时间小于偶发任务总共需要的执行时间，则不可调度。

（3）while（（F_{j-1}（S_j，$T_{dynamic}$）+G_{j-1}（et_{j-1}，$T_{dynamic}$，S_j+D_j））$\geqslant R_j$）{

根据偶发任务 r 占用的空闲时间和实际挪用时间更新空闲时间向量 F_{j-1}，并记录在 F_j 中；将偶发任务 r 的结束时间记录在变量 e_{tj} 中。

if（j<k）{

j++;

}else return True;

}

return False;

当出现处理器空闲情况时，说明所有已到达的偶发任务已经完成，这时空闲时间向量回复到初始状态；当新的超周期到来，则将空闲时间向量重新初始化。

3.8 仿真实验

IISS 算法采用一种类似 EDF 的调度算法来调度偶发任务。由于给出的判定条件仅仅是充分条件，因此下面通过仿真实验来验证 IISS 算法判定准确率。判定准确率通过如下办法计算：假设在判定算法中出现第 j 个偶发任务不能接受，则利

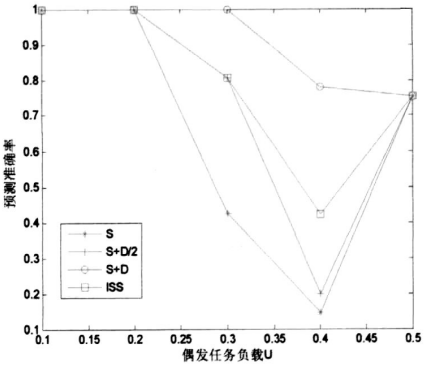

图 3.11 *C* 值为 10 时预测准确率

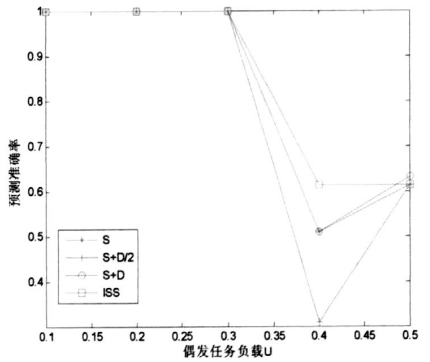

图 3.12 *C* 值为 20 时预测准确率

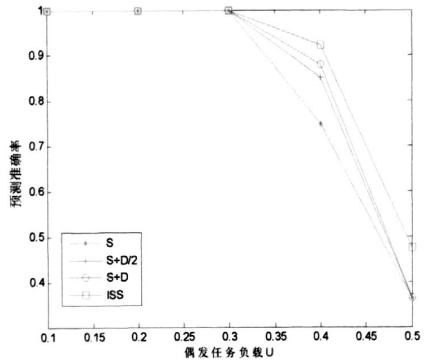

图 3.13 C 值为 40 时预测准确率

用模拟运行法，模拟使用 EDF 统一调度硬实时周期任务和偶发任务的过程，假设得出第 k 个偶发任务出现错失截止期的情况，那么(j/k)就用来做为判定算法的预测准确率。由于 EDF 调度算法的最优性，因此这个准确率可以作为判断算法优劣的依据。实验采用软件模拟的方法，在 Linux 环境下编写仿真代码：首先设定 10 个周期任务 τ1=(0,20,1,20)，τ2=(0,25,1,25)，τ3=(0,45,2,45)，τ4=(0,55,2,55)，τ5=(0,60,2,60)，τ6=(0,65,3,65)，τ7=(0,75,5,75)，τ8=(0,80,4,80)，τ9=(0,90,4,90)，τ10=(0,110,3,110)，任务周期随机生成。设定周期任务负载为 0.5，周期任务的执行时间随机生成。按照偶发任务平均负载的不同随机生成 500 个偶发任务 $r_i(0<i<501)$，$r_i=(S_i,R_i,D_i)$，其中 S_i 为到达时间，R_i 为执行时间，D_i 为相对截止期。偶发任务以一个随机时间间隔 C 到达，D_i 服从[2C,4C]区间均匀分布。偶发任务平均负载 U 按公式(3.9)计算。

$$U=\sum_{i=1}^{n}\frac{R_i}{C} \tag{3.9}$$

偶发任务到达时间间隔 C 值为 10、20、40 个时间单位时，分别设置 $T_{dynamic}$ 值为 Si、t_{min} 及 Si+Di，Si+Di/2，用 IISS 算法判定能否接受偶发任务集，根据判定正确率计算各自在不同偶发任务负载情况下的正确率。其中 $T_{dynamic}$=Si 时蜕化为最大挪用时间法，$T_{dynamic}$=tmin 时蜕化为 ISS 算法。实验结果如图 3.11、3.12、3.13，结果表明在偶发任务到达时间间隔相对周期任务平均周期较小时，取较大 $T_{dynamic}$ 值的判定准确率优于最大挪用时间法和 ISS 算法。

3.9 本章小结

提出一种硬实时周期任务与偶发任务混合调度的可调度性判定算法 IISS。该算法从 EDF 算法调度与 EDF 算法逆调度概念出发，在分析了调度过程中可挪用时间后，得出任意时刻最大可挪用的计算方法；在此基础上，定义了区间空闲时间及其计算方法，并给出了区间最大可挪用时间计算方法；最后，提出 IISS 算法判定硬实时周期任务与偶发任务混合调度中偶发任务的可调度性。相比于 ISS 算法，IISS 算法更加灵活，准确率更高。

第 4 章 基于动态抢占阈值的 LSF 调度算法研究

4.1 引言

最小空闲空闲时间优先（LSF）调度算法是一种经典的动态优先级实时调度算法，是 EDF 调度算法的改进算法，并且 LSF 扩展算法也很多[31]。已经证明，如果用 LSF 调度算法不可调度的任务集，则该任务集用其它动态调度算法也为不可调度的。尽管对于抢占式调度算法来说，任务的多余抢占很难避免，然而，当任务集中有两个以上任务的优先级相同时，LSF 调度过程中会经常因为任务过度抢占而造成“颠簸”现象，导致任务截止错失率的增加及 CPU 资源的浪费。因此，有必要改进 LSF 调度算法。关于 LSF 调度算法的过度抢占问题，基本思路是试图通过引进抢占阈值策略来解决。

本章首先详细分析 LSF“颠簸”现象产生过程，并且分析了不同空闲时间区间发生的“颠簸”对整个调度算法的不同影响。然后介绍了抢占阈值调度策略。接着通过设计合理的优先级分配方案及抢占阈值确定方法，将抢占阈值调度策略[32~34]引入 LSF 调度算法，提出一种基于动态抢占阈值的 LSF 调度算法的改进算法 DPTLSF。最后通过仿真实验证明了 DPTLSF 算法的改进效果

4.2 LSF 调度算法分析

4.2.1 任务模型定义

任务集是由 N 个周期性任务 τ_i（$1 \leqslant i \leqslant N$）构成，任务 τ_i 由一个六元组（T_i，C_i，S_i，D_i，P_i，H_i）所确定。其中：

（1）T_i 表示任务的周期；

（2）C_i 表示任务的最坏情况下执行时间；

（3）D_i 表示任务的绝对截止；。

（4）S_i 表示任务的空闲时间；

（5）P_i 表示任务的优先级。约定优先级越大，优先级越高

（6）H_i 表示任务的抢占阈值。

并且假设系统满足以下条件：

（1）单处理器环境；

（2）任务除非被抢占，否则不会自动挂起；

（3）任务之间相互独立；

（4）不考虑任务等待 CPU 以外其他资源的情况；

（5）周期任务的截止期与周期相等

4.2.2 LSF 调度算法

LSF 调度算法是最早截止期优先调度算法（EDF）的改进算法。EDF 是一种动态优先级调度算法，任务的优先级由截止时间决定，截止时间越小，优先级越高。EDF 算法可以调度 RM 算法不能合理调度的任务集，能够达到很高的 CPU 利用率。然而，EDF 很可能使一些截止时间较大，但是执行时间较长的任务错失截止期，特别是在系统负载较重时，性能退化很快。LSF 从这些缺陷出发，通过引入空闲时间来改进 EDF 的不足，在调度时判断任务是否应该调度，并且夭折不能完成的实时任务，使得 LSF 算法能够良好的工作在系统负载较重的情况下。

在 LSF 算法中，任务的优先级根据空闲时间来分配，任务在 t 时刻的空闲时间定义如下：

$$S_i(t) = D_i - t - C_i \tag{4.1}$$

其中 C_i 为剩余部分任务最坏情况执行时间。空闲时间反映了任务的紧急程度。由上式可知，任务的最小空闲时间随时间变化，对于等待任务，D_i 和 C_i 为常数，所以任务的空闲时间单调递减；对当前运行的任务，运行前后的空闲时间不变。任务的空闲时间的初始值为任务周期减去最坏情况执行时间。空闲时间值小于零表示任务已经无法按时完成，应该被夭折。调度时，调度器选择具有最小空闲时间的任务运行，若有多个任务的最小空闲时间相同，则选择绝对截止期小的任务先运行。

4.2.3 LSF 调度算法的颠簸现象

在 LSF 调度算法中，在具有多个任务的任务集中，当存在两个以上的任务具有相同或相近优先级时，由于运行过程中空闲时间的变化，容易导致过多的上下文切换，产生颠簸“颠簸”。如下面的例子，假设对于给定的三个任务 τ_1，τ_2，τ_3 各参数如表 4.1：

表 4.1 任务参数表

最坏情况下执行时间 C	绝对截止期 D	空闲时间 S
5	12	7
6	13	7
7	14	7

由 LSF 调度算法可知 τ_1，τ_2，τ_3 交替执行，在每个时间片都发生抢占[35]，其运行情况如图 4.1。

由图 4.1 可见，运行过程中产生了过多的抢占，而且三个任务因为均分 CPU

造成两个任务错过了截止期，CPU 有效利用率为 41.7%，因此任务集的截止错失率和空闲时间比较小时过多的任务抢占有直接关系；如果几个任务的空闲时间比较大时发生激烈 CPU 抢占，虽然也会造成过多任务上下文切换，但是对任务集的

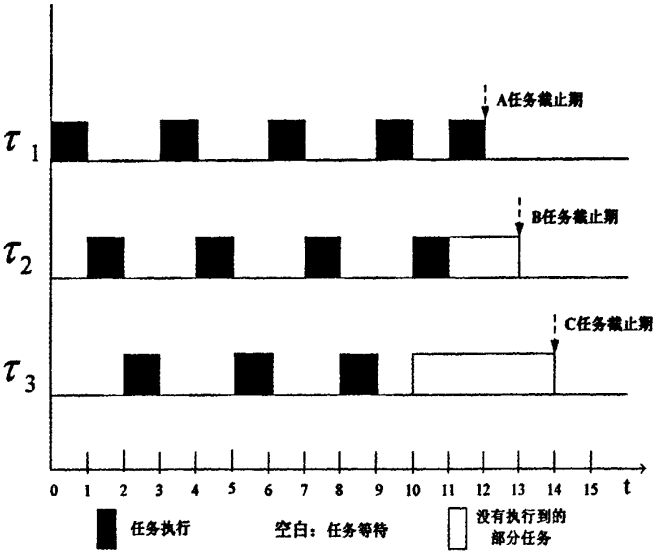

图 4.1 LSF 调度示意图

截止措施率的影响比较小，因为即使均分 CPU，仍然能比较从容完成任务，这在后面的仿真实验结果中也得到了证明。在 LSF 算法中，所有错失截止期的任务都是在最小空闲时间很小的情况下，没有竞争到 CPU 而夭折，因此，趋于零的一段空闲时间区间内的频繁抢占是造成 LSF“颠簸”现象的主要原因，也是增大任务错失率的一个重要原因。

4.3 抢占阈值策略

抢占阈值策略通过合理设置抢占阈值，充分利用了抢占调度和非抢占调度的优点。特别地，当阈值取当前任务优先级时，是抢占调度；当阈值取足够大时，是非抢占调度。可见，抢占调度和非抢占调度只是抢占阈值调度中的特例。将抢占阈值策略引入 LSF 算法，分析上面的例子：约定优先级值越大，优先级越高（以下不再赘述）。假如当任务优先级大于某个值时（对应空闲时间为 7 时的优先级），其抢占阈值设为无穷大或者任务集最大优先级，LSF 算法蜕化为非抢占调度，则运行情况如图 4.2。可见，设置合理的抢占阈值后，任务连续执行，抢占次数为零，只有一个任务错失截止期，CPU 有效利用率为 100%，效果显著，具体实现详见下节基于方案二阈值确定方法的 DPTLSF 算法。

4.4 LSF 算法改进

为了解决“颠簸“现象带来的频繁上下文切换及任务截止错失率问题，需要改

进 LSF 算法。根据上面的分析，要减少上下文切换次数，可以采用抢占阈值思想，每个任务除了分配优先级外，还分配一个抢占阈值，如果其它任务要抢占当前任务，不但要求优先级大于当前任务优先级，还必须满足优先级大于当前任务的抢

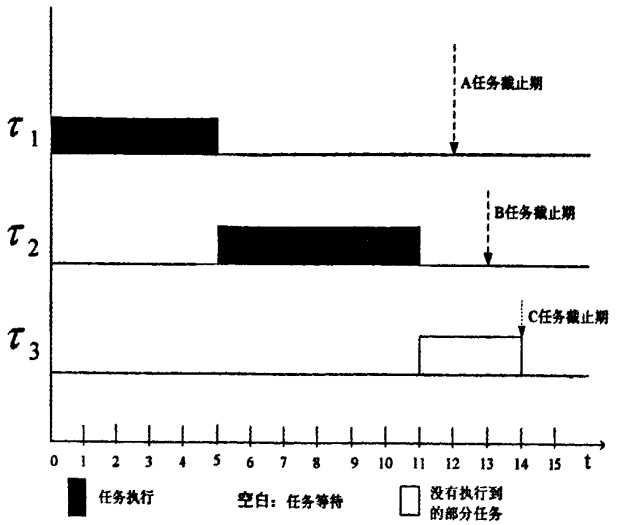

图 4.2 基于抢占阈值的 LSF 调度

占阈值。对于过度抢占带来的截止错失率问题，根据上面的分析，可以在趋于零的很小一段空闲时间区间设置对应的抢占阈值为无穷大或任务集中最大优先级，以保证任务的顺序执行，不至于因为抢占使各任务均分 CPU 资源，导致较高的任务截止错失率。提出的 DPTLSF 算法基于下面方案二的阈值确定方法，仍然采用 LSF 的优先级确定方法，调度时根据优先级与阈值的比较判定是否抢占，若当前执行任务完成则选择优先级最大的任务获得 CPU。

4.4.1 优先级的分配

优先级分配[36~39]是动态调度算法的关键，在 LSF 算法中，优先级仅由任务的空闲时间决定，空闲时间越小，优先级越大。假设空闲时间为 0 时，对应的优先级为 P_max，空闲时间为 L_max 时对应的优先级为 0。不失一般性，设计如下优先级分配函数：

$$P = kL + b, L \in [0, L\_\max] \tag{4.2}$$

其中

$$k = -\frac{P\_\max}{L\_\max}, \tag{4.3}$$

$$b = P\_\max \tag{4.4}$$

P_max 和 L_max 分别为系统中最大优先级和最大空闲时间，不考虑空闲时间小于 0 的情况，由调度器自动夭折这些任务。函数图象如图 4.3。

4.4.2 抢占阈值的确定

提出两种阈值抢占确定[40~44]方案，方案一主要考虑解决过多上下文切换问题，采用一种简单的阈值设计方法，这是一种普通的改进方案，与文献[14]中提出的算法相似，在本文中主要用来与方案二的比较分析。方案二在方案一的基础上，设置在空闲时间比较小区间内的抢占阈值为最大优先级，这是DPTLSF算法的关键。

方案一 和优先级分配一样，设计抢占阈值与空闲时间满足线性关系，即

$$P\_gate = -\frac{P\_\max - M}{L\_\max} L + P\_\max \qquad (4.3)$$

其中

$$L \in [0, L\_\max] \qquad (4.4)$$

P_gate 表示抢占阈值，L 为空闲时间，M 为当 L=L_max 时对应的阈值。对应的函数图象如图 4.4。

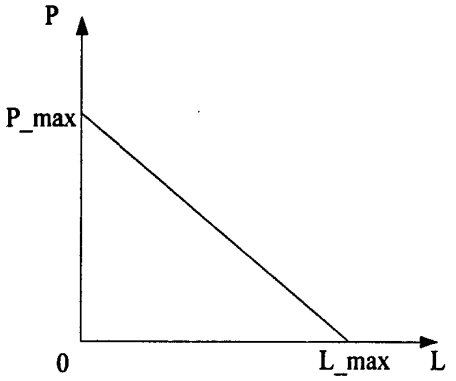

图 4.3 优先级分配函数图象

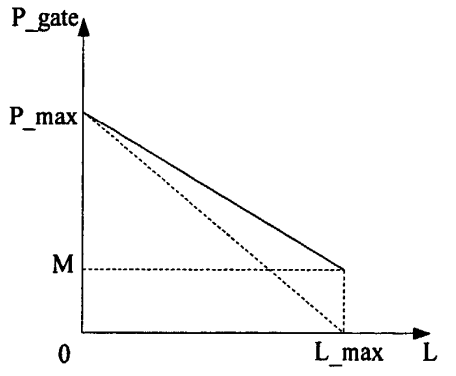

图 4.4 方案一的阈值分配函数图像

方案二 考虑在某个很小的空闲时间范围[0，u）内，将抢占阈值设计为最大优先级，即在该空闲时间范围内，任务不可抢占。因此，可以设计抢占阈值与空闲时间之间满足如下函数：

$$P\_gate = \begin{cases} P\_\max, L \in [0, u), \\ kL + b, L \in [u, L\_\max] \end{cases} \tag{4.4}$$

其中：

$$k = \frac{P\_\max}{u - L\_\max}, \tag{4.5}$$

$$b = P\_\max - ku \text{。} \tag{4.6}$$

对应函数图象如图 4.5。

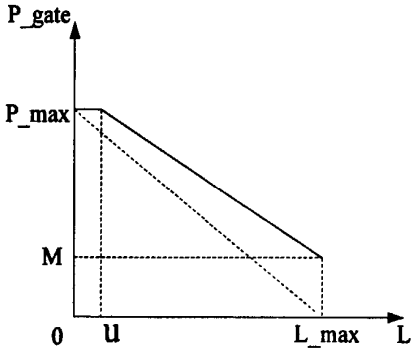

图 4.5 方案二的阈值分配函数图像

4.5 仿真运行模型

运行模型是动态优先级抢占阈值调度，是固定优先级调度模型的泛化。对于就绪任务来说，它根据自己的优先级来竞争 CPU。除非任务运行完成而自动让出 CPU,否则，当任务投入运行后，当且仅当就绪队列中某任务优先级超过当前运行任务的阈值，当前运行任务被抢占。

假设当前时间片内，任务τ_i占用 CPU,那么该任务一定是就绪队列的队首任务，在进入下一时间片后：

(1) 如果当前任务τ_i完成，则将该任务节点摘下，插入等待队列 Q2。

(2) 顺序搜索就绪队列 Q1，找出任务节点的空闲时间小于 0 的节点，摘下，插入等待队列 Q2。

(3) 顺序搜索等待队列 Q2，找出任务节点的周期值是否为 0，若为 0 则摘下该任务节点。生成一个新任务，用新任务的任务参数更新该节点。然后将该节点插入就绪队列 Q1。循环执行步骤(3)直到 队列中不存在周期域为 0 的任务节点。

(4) 更新就绪队列中各任务节点的优先级，空闲时间，抢占阈值各域值。

(5) 如果τ_i在上一时间片执行完毕，则就绪队列队首任务投入运行，执行步骤(7)。否则， 执行步骤(6)。

(6) 判断就绪队列中是否存在任务的优先级大于τ_i的抢占阈值，若存在则抢占τ_i，获得 CPU,将 τi 插入就绪队列正确位置。否则执行步骤(7)

(7) 当前执行任务的执行时间减一，更新队列 Q1，Q2 的任务节点周期域值。

(8) 循环执行以上步骤，直到仿真时间结束。

4.6 算法实现分析

4.6.1 数据结构

采用两个单链表分别作为任务就绪队列和等待队列。单链表节点为任务节点。就绪队列中存储的是调度器选择调度的目标队列，等待队列存储的是错过截止期而夭折或者任务已经完成但是还没有到下一个周期时间的任务。任务产生器由等待队列中产生，当任务等待队列中任务节点的周期变量值降为 0 时，意味着需要产生周期任务的下一个任务实例。任务节点的定义如下：

```
Typedef struct TaskNode{
    float C;//任务的执行时间
    float T;//任务的周期
    float L;//任务的空闲时间
    float P;//任务的优先级
    float P_gate;//任务的抢占阈值
    struct TaskNode *next;//指向下一任务的指针
}TASK;//任务结构体
struct TASK *Q1，*Q2;//Q1，Q2 分别为就绪队列和等待队列
```

在就绪队列中，各任务按照优先级由高到低排列。

4.6.2 任务产生器

当等待队列中某任务节点的周期变量值降为 0 时，在调度前应该生成一个新任务，更新任务节点信息，并且从等待队列摘下，插入就绪队列适当位置。遍历整个等待队列，重复上面的任务产生过程，直到等待队列中不存在任务节点的周期变量值为 0 为止。初始等待队列为空，任务 τi ($1 \leqslant i \leqslant N$)都在就绪队列中。任务产生过程分两步：

(1) 生成新任务参数，抢占阈值按如下设置：

```
if(p->L<=u1)p->gate=P_max;//p 为指向任务节点的指针
else if(p->L>=u2)p->gate=M;
else{
        p->gate=1.0*(P_max-M)*(p->L-u)/(u-L_max)+P_max;
```

```
}
```

(2) 节点生成后插入就绪队列适当位置。

4.6.3 任务调度器

(1) 在每个调度点，先检查就绪队列是否存在任务已经完成或者夭折，若存在，则将其摘下，插入等待队列。

(2) 判断等待队列是否有任务节点周期变量值为 0，若存在则将该节点摘下，生成新任务作业并插入就绪队列适当位置。

(3) 若前一任务是完成后退出就绪任务队列，则从就绪队列中选出优先级最高的任务执行；否则，检查就绪任务队列首节点后是否有任务满足当前任务的抢占阈值条件，若存在多个任务满足抢占条件，则选出其中最高优先级任务执行，若不存在则当前任务继续执行。

4.6.4 完成与夭折策略

如果任务剩余执行时间为 0，或者空闲时间小于 0，则按如下处理：

```
if(t->C==0 || t->L<0 ){
        if(t==Q1){Q1=Q1->next;p=t;s=t=Q1;}
        else{p=t;t=t->next;s->next=t;}
        if(p->L<0)fail++;//统计错失截止期任务数
        if(!Q2){Q2=p;p->next=0;}//插入等待队列
        else{
                p->next=Q2;
                Q2=p;
        }
}
```

4.6.5 算法复杂度分析

假设周期任务集任务个数为 n，对已完成任务和夭折任务的处理需要遍历整个就绪任务队列，因此时间复杂度为 O(n)；新建任务并插入就绪队列适当位置时间复杂度为 $O(n^2)$；选择下一个执行任务执行并对就绪队列每个任务参数进行更新时间复杂度为 O(n)。因此整个调度算法的时间复杂度为 $O(n^2)$。内存空间方面只需要存储就绪任务队列和等待任务队列，因此空间复杂度为 O(n)。

4.7 仿真实验

按照上面章节的分析，在 Linux 下通过软件来仿真 DPTLSF 调度算法的执行过程。

4.7.1 仿真条件

任务集中任务个数 N=5（仿真实验四中任务数可变），且都为周期性任务，仿真运行持续时间为 1000 个时间片。各任务的最坏情况执行时间 C_i 在区间[1，10]内服从均匀分布。任务周期 T_i 按照公式：T_i=N*C_i/ρ 计算。其中 ρ 表示期望的工作负载。任务集中最高优先级 P_max=50，最大空闲时间 L_max=40。

4.7.2 两种性能指标

(1)任务截止期错失率(MDP)等于所有错失截止期的任务数除以任务总数。

(2)上下文切换次数(CSN)等于仿真时间内任务抢占的次数，此时不计任务完成后进行的任务切换

4.7.3 仿真结果与分析

1. 仿真实验一

令处理器负载 ρ=1.5，非抢占区间右端点 u=5，图 4.6 与图 4.7 为基于方案一与方案二的动态抢占阈值 LSF 算法在不同 M 值时对应的任务截止错失率（MDP）

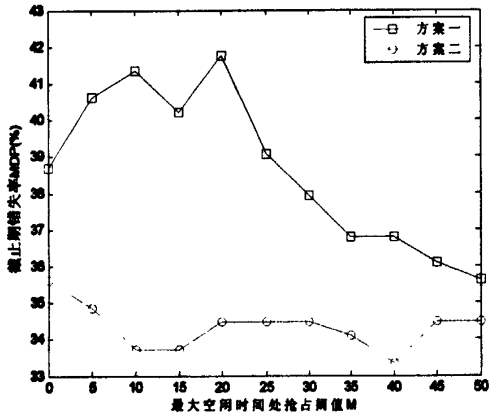

图 4.6 不同最大空闲时间处抢占阈值 M 的任务截止期错失率

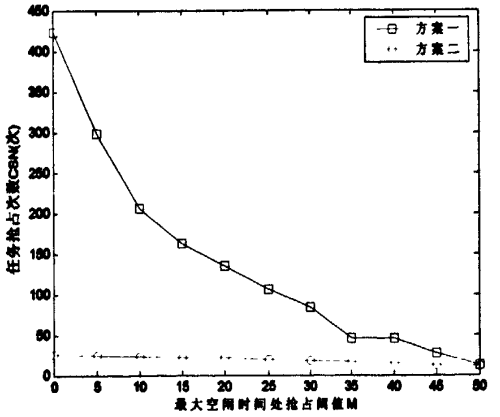

图 4.7 不同最大空闲时间处抢占阈值 M 的任务抢占次数

和抢占次数（CSN）曲线图。由图 4.4 可知，当最小阈值 M=0 时，基于方案一的算法蜕化为已有的 LSF 调度算法；当 M=P_max 时，基于方案一的算法为不可抢占调度。从图 4.6 可以看出，基于方案一的 LSF 算法随 M 值增大，任务截止错失率先是有所增大，而后一直减小。然而基于方案二的 DPLSF 算法受 M 值的

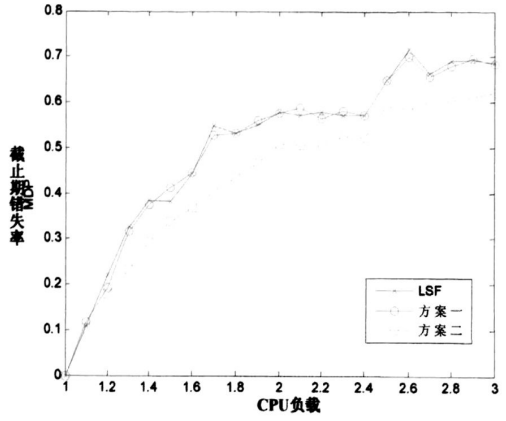

图 4.8 不同 CPU 负载情况下的任务截止期错失率

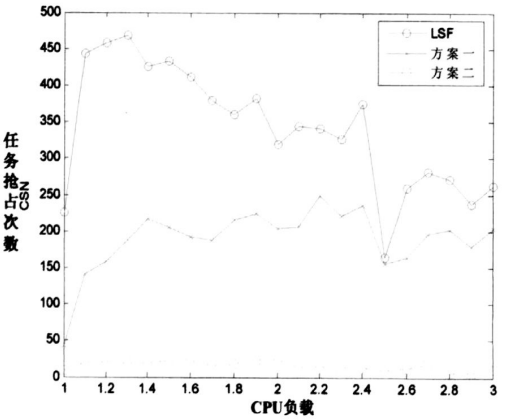

图 4.9 不同 CPU 负载情况下任务抢占次数

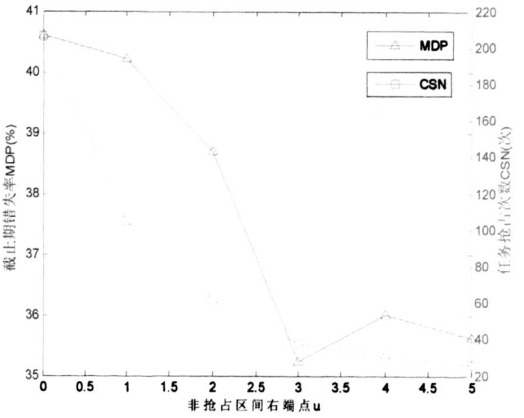

图 4.10 不同 u 值（时间片）的截止期错失率和任务抢占次数

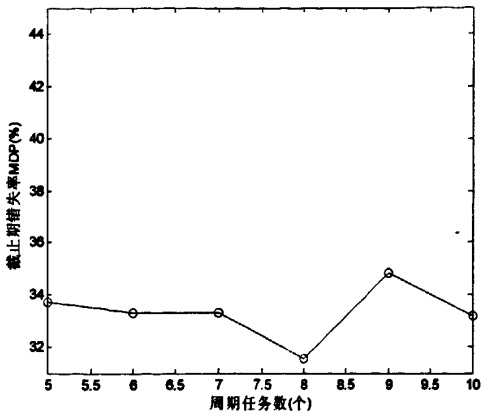

图 4.11 不同任务个数情况下的截止期错失率

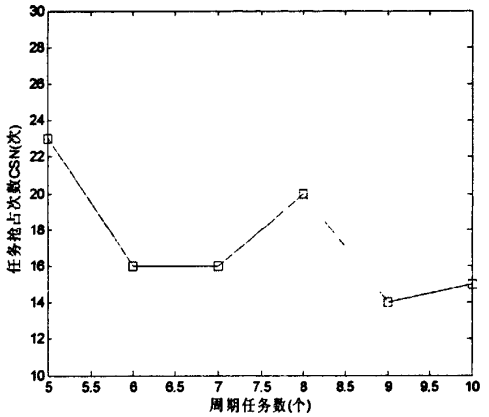

图 4.12 不同任务个数情况下的任务抢占次数

影响不大。图 7 表明，基于两种阈值设定方案的改进算法都能够减少任务的抢占次数，特别是 DPLSF 算法，对于不同的 M 值，其抢占次数平稳地维持在非常低的水平。

2. 仿真实验二

令方案一中最小阈值 M=10，方案二中最小阈值 M=0，非抢占区间右端点 u=5，图 4.8、图 4.9 为基于方案一与方案二的动态抢占阈值 LSF 算法及已有的 LSF 算法在不同负载时对应的任务截止错失率与任务抢占次数曲线图。由图 4.8 可知，在不同负载情况下，基于方案一的算法和已有的 LSF 算法的任务错失率基本一样；而 DPLSF 算法始终保持更低的任务错失率。因此相比较已有的 LSF 算法，DPLSF 算法在降低任务截止错失率方面有很大改进。图 4.9 表明，在不同负载情况下，基于方案一与方案二的算法都减少了任务抢占次数，而且，DPLSF 的任务抢占次数远远小于已有的 LSF 算法和基于方案一的 LSF 算法。

3. 仿真实验三

令处理器负载 $\rho=1.5$，最小阈值 $M=0$，图 4.10 为方案二在不同非抢占区间右

端点 u 值时对应的任务截止错失率与任务抢占次数曲线图。由图 4.10 看出，随着 u 值增加，任务截止错失率急剧下降。当 u=0，PTLSF 算法蜕化为方案一的算法，任务错失率最高。当 u=3 时，MDP 最小，说明趋于零的较小空闲时间区间内的抢占对 LSF 算法的性能具有决定性作用。图 4.11 说明了任务切换次数随 u 值增大而迅速减小。

4. 仿真实验四

令处理器负载 ρ=1.5，M=10，非抢占区间右端点 u=5。在不同周期任务个数情况下，按照方案二调度周期任务集，任务截止期错失情况与上下文切换次数结果如表 4.2。从表 4.2 可以看出，在不同周期任务个数的任务集情况下，DPTLSF 算法的执行效果非常稳定。

上面四个仿真实验的结果充分说明了 DPTLSF 算法改进效果，尽管引入抢占阈值对实时任务的平均响应时间有影响，但是对于负载较重的软实时环境，这样的代价是值得的。

4.8 本章小结

在最小空闲时间优先（LSF）调度算法中，当任务集中有多个任务的优先级相同或相近时，过多的上下文切换产生的“颠簸“现象会大大增加系统开销。基于抢占阀值思想，结合 LSF 算法本身的特点，通过设计合理的动态抢占阀值，提出一种改进的算法 DPTLSF。DPTLSF 算法的优先级分配方案区别于文献[14]之处在于不局限于某个固定斜率，因此，它可以灵活运用于不同空闲时间单位与不同优先级数目的系统中；其次，DPTLSF 首次提出在较小空闲时间区间内采用非抢占调度，以此来避免该时间范围内激烈的任务抢占。仿真结果表明，改进后的算法能够显著地减少“颠簸”现象的发生，降低任务集的截止期错失率。尽管付出了一些响应时间代价，但是对于负载较重的软实时环境具有很大应用价值。

结 论

1. 主要工作

实时系统的任务调度算法是NP难问题，它直接影响系统的实时性和可靠性。因此，实时调度算法的研究一直是计算机理论研究中的热点。本文结合目前实时调度领域的发展现状，对实时调度算法进行了深入研究和讨论，主要工作有：

（1）提出一种硬实时周期任务与偶发任务混合调度的可调度性判定算法IISS。该算法从EDF算法调度与EDF算法逆调度概念出发，在分析了调度过程中可挪用时间后，得出任意时刻最大可挪用的计算方法；在此基础上，定义了区间空闲时间及其计算方法，并给出了区间最大可挪用时间计算方法；最后，提出IISS算法判定硬实时周期任务与偶发任务混合调度中偶发任务的可调性。相比于ISS算法，IISS算法更加灵活，准确率更高。

（2）提出一种基于动态抢占阈值的LSF改进算法DPTLSF。在最小空闲时间优先（LSF）调度算法中，当任务集中有多个任务的优先级相同或相近时，过多的上下文切换产生的“颠簸“现象会大大增加系统开销。基于抢占阀值思想，结合LSF算法本身的特点，通过设计合理的动态抢占阀值，提出一种改进的算法DPTLSF。仿真结果表明，改进后的算法能够显著地减少“颠簸”现象的发生，降低任务集的截止期错失率。

IISS算法对于需要保证到达时间不确定的实时任务的系统具有较大应用空间；DPTLSF尽管付出了一些响应时间代价，但是对于负载较重的软实时环境具有很大应用价值。

2. 进一步的研究工作

在总结分析本文主要贡献的同时，论文进一步的研究工作包括如下方面：

（1）研究IISS算法如何应用到实际的实时系统中。因为对于实际中复杂的应用环境，偶发任务的特征往往是复杂多变的，所以抢占时间点$T_{dynamic}$的确定在实际应用中比较困难，应该继续改进IISS算法以找到更加有效的可调度性判定条件，最好能找到它的充分必要条件。

（2）由于嵌入式系统的迅猛发展，对实时系统的需求非常强烈，而Linux是一种开源的操作系统，目前在嵌入式领域应用很广泛。由于DPTLSF算法对于过载系统环境的良好性能，其应用可以将DPTLSF算法加到Linux调度器中，实现一种新的面向软实时任务的调度策略，是下一步的研究方向。

参考文献

[1] 汤子瀛, 哲凤屏, 汤小丹. 计算机操作系统. 西安: 西安电子科技大学出版社, 2001, 10-11

[2] 张杰, 阳富民, 卢炎生等. EDF 统一调度硬实时周期任务和偶发任务的可调度性判定算法. 小型微型计算机系统, 2009, 30(12): 2383-2388

[3] Lehoczky J P, Sha L, Strosnider J K. Enhanced aperiodic responsiveness in hard real-time environments. In: Proceedings of IEEE Real-Time System Symposium December, 1987, 261-270

[4] Lehoczky J P, Thuel S R. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In:Proceedings of IEEE Real-Time System Symposium, December, 1992, 110-123

[5] He Jun, Sun Yu-fang. An algorithm to improve the responsive performance for scheduling soft-aperiodic tasks. Journal of Software, 1998, 9(10): 721-727

[6] Chetto H, Chetto M. Some results of the earliest deadline scheduling algorithm. IEEE Trans. on Software Engineering, 1989, 15(10): 1261-1269

[7] He Xiaochuan, Jia Yan. 1eakage-aware energy efficient scheduling for fixed-priority tasks with preemption thresholds. Berlin: Spnnger-Verlag, 2008, 379-390

[8] Rony Ghattas, Alexander G Dean. Preemption threshold scheduling: stack optimality,enhancements and analysis. Washington. DC: IEEE Computer Society, 2007, 147-157

[9] Y Wang,M Saksena. Scheduling fixed-priority tasks with preemption threshold. The 16th Int'l conf on Real-Time Computing Systems and Applications (RTCSA'99). Hong Kong, 1999,328-335

[10] 金宏, 王强, 王宏安等. 基于动态抢占阈值的实时调度. 计算机研究与发展, 2004, (3): 393-398

[11] 谭云福, 刘杰, 刘国华. Linux 中一种改进的实时调度算法及其应用. 计算机科学, 2008, 35(10): 256-258

[12] Kim S, Hong S, Kim T H. Integrating real-time synchronization schemes into preemption threshold scheduling. Washington, DC: IEEE Computer Society, 2002, 145-154

[13] 陈旭辉, 于国龙. 云模型优化LSF调度算法的研究. 计算机工程与设计, 2010,

31(13): 3014-3016

[14] 金宏, 王宏安, 王强等. 改进的最小空闲时间优先调度算法. 软件学报, 2004, 15(8): 1116-1123

[15] Y.Luo, R.Galli, M.Mascaro, etal. Real-Time Multi-User Interaction with 3D Graphics via Communication Networks. Information Visualization, IEEE Conference on 29-31 July 1998, pages: 60-68

[16] Liu C L, Layland J. Scheduling algorithms for multiprogramming in real-time systems. Journal of the ACM, 1973, 20(1): 46-61

[17] Ching-Chih Han, Kwei-Jay Lin, Chao-Ju Hou. Distance-Constrained Scheduling and Its Applications to Real-Time Systems IEEE Transaction on Computers, 1996, 45(7): 814-826

[18] Michael B. Jones, John Regehr. CPU Reservations and Time Constraints: Implementation Experience on Windows NT. In: Proceedings of the 3rd USENIX Windows NT Symposium, Jul 1999, 93-102

[19] H.Kopetz. The Time-Triggered Model of Computation. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, Dec, 1998, 168-177

[20] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of ACM, 1973, 20(1): 174-189

[21] Katcher D I, Arakawa H, Strosnider J K. Engineering and analysis of fixed priority schedulers. IEEE Trans.on Software Engineering ,1993, 19(9): 920-934

[22] Sanjoy K. Baruah, N.K.Cohen, C.Greg Plaxton, etal. Proportionate Progress: A Notion of Fairness in Resource Allocation. Algorithmica, 1996, 15(6): 600-625

[23] Kevin Jeffay, F.Donelson Smith, A. Moorthy,etal. Proportional Share Scheduling of Operating System Services for Real-Time Applications. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, Dec, 1998, 480-491

[24] Ion Stoica, Hussein M. Abdel-Wahab, Kevin Jeffay, etal. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. In: Proceedings of the 17th IEEE Real-Time Systems Symposium, Dec, 1996, 288-299

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

[26] Burchard A, Liebeherr J, Oh YF, etal. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. On Computers, 1995, 44(12): 1429-1442

[27] Hsueh C W, Lin K J. Schedulability comparisons among periodic and distance-

constrained real-time schedulers. In: Proc. of the 4th Int'l Workshop on Real-Time Computing Systems and Applications(RTCSA'97). Taipei：IEEE Computer Society Press, 1997, 60-66

[28] Han CC, Lin KJ, Hou CJ. Distance-Constrained scheduling and its anplcations to real-time systems. IEEE Trans. on Computers, 1996, 45(7): 814-826

[29] 涂刚，阳富民，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

[30] 王保进. 抢占阈值调度算法的分析与研究. 微计算机信息, 2005,21(8):83-85

[31] Lee J, Tiao A., Yen J. A fuzzy rule-based approach to real-time scheduling. In: Proceedings of the 3rd IEEE International Conference on Fuzzy Systems, Orlando, Florida, USA, 1994, 2: 1394-1399

[32] 贺小川，贾焰. FPTS: 一种任务间存在共享资源时的抢占阈值调度算法. 计算机研究与发展, 2009, 46(2): 302-309

[33] 贺小川，贾焰. 抢占阈值调度的功耗优化. 计算机学报, 2008, 31(11): 2060-2071

[34] 金宏，王宏安，王强. 模糊动态抢占调度算法. 计算机学报, 2004, 27(6): 812-817

[35] 王保进. 在构件化嵌入式操作系统中应用抢占阈值调度. 计算机工程与应用, 2005, 41(19): 22-25

[36] 刑建生，王永吉，刘军祥等. 一种静态最少优先级分配算法. 软件学报. 2007, 18(7): 1844-1854

[37] 宾雪莲，杨玉海，金士尧. 一种有限优先级的静态优先级分配算法. 软件学报, 2004, 15(6): 815-822

[38] 王多强，鲁剑锋，李庆华. 实时调度中基于多特征参数的任务优先级设计方法. 计算机工程与科学, 2008, 30(1): 73-78

[39] 王保进，李明树，王志刚. 静态实时中间件的优先级映射问题. 计算机研究与发展, 2006, 43(4): 722-728

[40] 黄广君，胡正国. 一种基于阈值的嵌入式实时系统调度算法. 计算机工程, 2006, 32(3): 68-69

[41] 郑志冯，全源. 低复杂度的阈值优化实时调度算法. 计算机工程与设计, 2008, 29(17): 4411-4413

[42] He Dong Zhi，Wang Fei Yue，Li Wei. Dynamic preemption threshold scheduling for specific real-time control systems. Tucson：Proceedings of IEEE Networking, Sensing and Control Proceedings，2005：395-400

[43] Saksena M，Wang Y. Scalable multi-tasking using preemption Thresholds.

Orlando: Proceedings ofthe 21st IEEE Real Time Systems Symposium, 2000, 25-34

[44] He Dong Zhi, Wang Fei Yue, Li Wei, etal. Hybrid earliest deadline first preemption threshold scheduling for real-time systems. Shanghai: Proceedings of IEEE the 3rd International Conference on Machine Learning and Cybernetics, 2004, 433-438

致 谢

难忘的研究生生活即将结束，在这段日子里有过迷茫，有过困惑但更多的是收获，是友谊。从此，这段美好的时光，这些收获的愉悦，这些亲近的朋友将永远留在我记忆的深处，成为我人生最宝贵的财富。

在求学和科研期间，自始至终都是在尊敬的导师任小西副教授的悉心指导和严格要求下进行的。恩师以渊博的理论知识和实践经验、深邃的洞察力对我进行耐心的指导，开拓了我的学术视野，加深了我对课题的认识，使我受到了严格的训练。恩师严谨的治学态度、勤于创新的科研精神、吃苦耐劳的工作作风、正直坦诚和为人师表的优秀品质，都给我留下了深刻的印象，使我懂得了如何做人做事。借此论文完成之际，谨向辛勤培育和时刻关心鼓励我的导师表示深深的谢意。感谢老师对我学业上的指导，生活上的关心和照顾，正是老师的谆谆教诲和无微不至的关怀使我顺利地完成了硕士研究生阶段的学习。老师深厚的理论基础，过硬的动手能力为我树立了良好的榜样，这一切都让我难忘。

感谢我的父亲母亲，是你们不断鼓励我战胜困难，战胜自我，从精神上给我前进的力量，从生活上一直无微不至的照料我，才使得我能顺利完成硕士学业，感谢一切帮助过我的亲人和朋友，你们的鼓励是我前进的动力。

感谢我的同门吴强，同寝室所有兄弟，在平时生活和学习上都给予我无私的帮助，正是从他们的身上让我看到了自己的不足，正是从他们的身上让我学到了很多新的东西，正是有了与他们和睦愉快的相处才使得原本枯燥、单调的求学生活不再平板。

感谢学院中为我研究生生活和学习提供过帮助的老师和同学，是他(她)们为我的学习和生活提供了良好的环境。

感谢实验室的同学们，你们对我的研究工作给予了许多帮助，你们勤奋好学、乐于助人的精神，使我一生难忘。

最后，衷心感谢所有审阅本论文的老师和专家们，感谢你们抽出宝贵的时间来阅读本文，并提出宝贵的意见和建议。

于研究生培养基地

2011.5

附录 A 攻读硕士学位期间所发表的学术论文目录

[1] 任小西, 赵公怡. 基于动态抢占阈值的 LSF 调度算法研究. 计算机工程, 已录用, 待发表. 文章编号: EC0016646