分类号：_____ 密级：

ＵＤＣ：_____ 编号：

工学硕士学位论文

支持多核处理器的
RTEMS 嵌入式操作系统的研究

硕 士 研 究 生：牛莹姣

指 导 教 师：张国印 教授

学 科、专 业：计算机系统结构

论 文 主 审 人：姚爱红 副教授

哈尔滨工程大学

2013 年 03 月

分类号：_____ 密级：

ＵＤＣ：_____ 编号：

工学硕士学位论文

支持多核处理器的
RTEMS 嵌入式操作系统的研究

硕士研究生：牛莹姣

指导教师：张国印　教授

学位级别：工学硕士

学科、专业：计算机系统结构

所在单位：计算机科学与技术学院

论文提交日期：2012 年 12 月 31 日

论文答辩日期：2013 年 03 月 14 日

学位授予单位：哈尔滨工程大学

Classified Index :
U.D.C:

A Dissertation for the Degree of M.Eng

Research of RTEMS Embedded Operating System with Support for Muliticore Processor

Candidate: Niu Yingjiao

Supervisor: Prof. Zhang Guoyin

Academic Degree Applied for: Master of Engineering

Specialty: Computer Architecture

Date of Submission: Dec. 31, 2012

Date of Oral Examination: Mar. 14, 2013

University: Harbin Engineering University

哈尔滨工程大学
学位论文原创性声明

本人郑重声明：本论文的所有工作，是在导师的指导下，由作者本人独立完成的。有关观点、方法、数据和文献的引用已在文中指出，并与参考文献相对应。除文中已注明引用的内容外，本论文不包含任何其他个人或集体已经公开发表的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者(签字)：

日期：　　年　月　日

哈尔滨工程大学
学位论文授权使用声明

本人完全了解学校保护知识产权的有关规定，即研究生在校攻读学位期间论文工作的知识产权属于哈尔滨工程大学。哈尔滨工程大学有权保留并向国家有关部门或机构送交论文的复印件。本人允许哈尔滨工程大学将论文的部分或全部内容编入有关数据库进行检索，可采用影印、缩印或扫描等复制手段保存和汇编本学位论文，可以公布论文的全部内容。同时本人保证毕业后结合学位论文研究课题再撰写的论文一律注明作者第一署名单位为哈尔滨工程大学。涉密学位论文待解密后适用本声明。

本论文(□在授予学位后即可　□在授予学位12个月后　□解密后)由哈尔滨工程大学送交有关部门进行保存、汇编等。

作者(签字)：　　导师(签字)：

日期：　　年　月　日　　年　月　日

摘　　要

近年来，随着信息量爆炸式增长，计算机系统的性能和功耗之间的矛盾日益凸现，单核处理器已不能满足各领域电脑用户的需求。单芯片多核处理器通过在芯片上集成多个频率较低的可执行核来解决单核发展遇到的瓶颈问题，而研究支持多核处理器的操作系统是多核得以应用的基础。RTEMS 嵌入式操作系统对多处理器提供了很好的支持。但是具体硬件体系结构不同，多核化设计过程中需要实现机制不一样。本文主要研究支持 SMP 体系的 RTEMS 多核化实现机制。

论文首先剖析 RTEMS 操作系统内核，在此基础上研究 RTEMS 系统对多核处理器的支持。系统采用了共享内存的体系架构，通过中断机制实现多核之间的通信。针对单核情况下的同步互斥机制在多核系统中不能保持原有语义的问题，系统采用改进的任务自旋锁与中断自旋锁相结合的机制，该机制较好的实现了 RTEMS 多核化的同步与互斥。针对 RTEMS 多核任务调度采用统一分配策略带来的不足，在对现有动态调度算法研究的基础上，提出根据系统资源动态变化自动进行任务调整的两级动态任务调度算法。

本文在 M5 多核仿真平台上利用 TGFF 工具生成测试任务的方法进行实验，验证 RTEMS 多核化系统的可行性和高效性，并选择加速比和负载平衡效率作为策略性能评价指标。测试结果证明了系统正确性，并且随着计算规模不断增大，新机制缩短了任务执行时间、改善了系统并行效率。

关键词：多核处理器；对称多处理器；RTEMS；互斥；动态调度

Abstract

In recent years, with the explosive growth of the information, coupled with Moore's Law limit, single-core processors can not meet the needs of computer users in various fields. Single-chip multi-core processors solves the bottleneck problems encountered in the development of the single-core through integrated multiple lower frequencies executable nuclear on-chip. Researching operating system that supports multi-core processors is the basis of the multi-core applications. RTEMS embedded operating system provides a good support for multi-processor. But realizing what kind of mechanism in the multicore design process should be based on the specific hardware architecture. This dissertation studies the RTEMS implementation mechanism which support SMP architecture.

The dissertation first analyze the RTEMS operating systems kernel, then research RTEMS system which provides support for multi-core processors. The system uses a shared memory architecture. Multicore achieve communication through the interrupt mechanism. For Synchronization mutex mechanism in the case of the single-core can not maintain the original semantics in the multicore system, the system uses the task spinlock and the interrupt spinocked combination mechanisms, and to prevent processor core has been in busy state it limits the waiting time for the spinlock. For the deficiencies of the unified allocation strategy implemented in RTEMS multicore system, proposing two dynamic task scheduling algorithm which can automatically adjusted the task according to the dynamic changes of the system resources on the basis of the study in the existing dynamic scheduling algorithm.

This dissertation experiments by use of random task graph to generate test tasks on the M5 multicore simulation platform to verify RTEMS multi-core system feasibility and efficiency, and select the speed up and load balancing efficiency as a strategic performance evaluation. The test results prove the system correctness, and with the calculation of the increasing scale, the new mechanism shortens the task execution time and improves the parallel efficiency of the system.

Keywords: multicore processor, symmetric multi processing, RTEMS, mutex mechanism, dynamic scheduling

目　　录

第 1 章　绪论

1.1　研究背景和意义

近年来，随着计算机处理的信息量越来越大，计算机性能成为用户选择产品考虑的重要因素[1]。处理器对计算机性能发挥起着决定性的作用，对于单核处理器来说，提高性能的主要途径是优化逻辑结构、扩大缓存以及提高芯片频率。但是芯片的逻辑结构一般有 4、5 年的生命周期，一旦设计稳定就很难从根本上对其做出改变，提高缓存容量虽然能够小幅度的提高性能，但付出的代价相当高，同样提高工作频率能够一定程度上提高芯片性能，但会导致运行功耗大增，现在芯片运行在风冷条件之下已经接近极限[2]。

多核处理器的出现就很好的解决了硬件系统性能提升的瓶颈问题。所谓多核处理器就是通过在同一个芯片上增加频率较低的处理器核数目，实现真正意义上的线程并行达到提高性能的目的[3]。多核处理器作为单枚芯片能够直接插入单一的处理器插槽中，操作系统会将其每个执行内核作为独立的逻辑处理器进行运行。多核处理器能够提高整个系统的并行度和执行效率，主要有 3 个方面的优点[4]。首先由于集成了多个处理器核心，使得整个处理器可以同时执行的任务数是单处理器的数倍，这极大地提高了处理器的并行性能和运算能力。其次，由于多个核都集成到了片内，核间的连线大大缩短，这就使得核间通信延迟降低，通信效率得到提高，数据传输带宽也大大增强。第三，由于多个核可以共享资源，因此这就提高了资源的利用率，而且也降低了器件的功耗。最后，多核的结构可以被设计成易于扩展的模式，这为以后的发展提供了便利。所有的这些优势使得多核处理器不可避免的取代单核成为未来发展的主流和方向。

在当前网络技术和数字信息技术高速发展的后时代，嵌入式系统应用越来越普及，已经渗透到人们日常生活、工程设计、科学研究、产业和商业、军事技术、娱乐业、文化艺术等各领域中[5]。随着国内外嵌入式产品，如 PDA、机顶盒、车载电脑等进一步设计开发和推广，嵌入式技术越来越和人们的生活紧密结合。从家用电子产品如电冰箱、微波炉和洗衣机，到办公用品如打印机、传真机和远程会议系统，到各种交通工具如摩托车、轮船、汽车和飞机等等，都或多或少地使用了嵌入式技术。

实时多处理器系统（Real-Time Executive for Multiprocessor Systems，RTEMS），由美国国防部开发设计，用于国防部的导弹控制系统[6]。RTEMS 操作系统是一款非常优秀的嵌入式实时操作系统，其不但实时性高、内核可裁剪、易于开发、具有良好的可移植性和可重用性。

那么基于单核版本的 RTEMS 能不能对多核处理器，尤其是在多核间的同步互斥通

信和调度机制方面提供高效的管理机制呢，答案是否定的。多核处理器改变了传统的（单机版）多任务、基于优先权的编程模式[7]，这体现在多核处理器允许多个线程被并行执行，而这是建立在单个操作系统运行于拥有多个核的处理器，或运行于多个处理器相互连接成为拥有多个处理单元的逻辑处理器背景上的。我们常说在某个单核的多任务环境中，多个线程可以“同时运行”，可实际上处理器一次只能执行一个线程——处理器依靠操作系统任务调度器以及中断机制来运行不同的线程。由于这一差异，除非在代码的设计和执行即将多核处理器系统作为立足点，否则从传统的多任务系统向多核处理器系统转移代码是行不通的。因此单核版本的 RTEMS 不能直接应用于多核处理器的硬件平台上。另外由于支持 SMP 架构的操作系统易于实现、易于负载平衡、具有高性能功耗比[8]，因此探讨支持 SMP RTEMS 操作系统的实现机制具有重要的现实意义。

1.2 国内外研究现状

随着多核处理器的发展，市场上也出现了相应的多核操作系统。目前主流的支持多核处理器的操作系统有：VxWorks，Linux 和 RTEMS。

VxWorks 操作系统于 1983 年由美国温瑞尔（WindRiver）公司设计开发，是一种嵌入式实时操作系统（RTOS）[9]。由于 VxWorks 支持多种嵌入式处理器核控制器和目前主流的 CPU，并且在安全可靠性、实时性和内核移植性等方面表现非常出色，得到业界人士的青睐。在 2007 年该公司推出了 VxWorks SMP 用于支持 SMP 嵌入式硬件平台，使得嵌入式系统真正的进入了多核时代。但是 VxWorks 是商用嵌入式操作系统，需要购买 Licence 才能使用。

Linux 从 2.0 版本开始对内核进行了比较大的修改以支持对称多处理器的并行调度，一直到 2.4 版本，其中主要在初始化和启动、内核的通信、任务调度算法、中断处理和同步以及为支持并行所进行的相应数据结构的修改[10]。Linux2.4 版本已经能很好地支持对称多处理器的要求。在该版本中，采用了时间复杂度为 o(1)的任务调度算法[11]。系统为每个处理器提供 2 个任务队列，一个是过期的任务队列，一个是活动的任务队列，每个任务队列都有 140 个优先级。任务按时间片轮转调度，时间片结束后任务由活动的任务队列进入过期的任务队列，每过一段时间，系统执行一次负载均衡调度，重新分配任务负载。最新的 Linux2.4 与单处理器系统的主要差别是执行任务切换后，被换下的任务有可能会换到其他 CPU 上继续运行。在计算优先权时，如果任务上次运行的 CPU 也是当前 CPU，则会适当提高优先权，这样可以更有效地利用 Cache 缓存。为实现多核间的正常通信，Linux 采取自旋锁机制保证对共享资源的互斥访问[12]。目前，Linux 对 SMP

支持的技术已经非常成熟，这为本文的研究提供了指导和依据。

RTEMS 是一个开源的无版税实时嵌入操作系统 RTOS。它最早用于美国国防系统，早期的名称为实时导弹系统（Real Time Executive for Missile Systems），后来改名为实时军用系统（Real Time Executive for Military Systems），现在由 OAR 公司负责版本的升级与维护。目前无论是航空航天、军工，还是民用领域 RTEMS 都有着极为广泛的应用。同大多数嵌入式操作系统一样，RTEMS 采用微内核设计思想，将内核主要功能集成在一个小的执行体中，附加的功能在包裹内核层的外层实现，应用可以根据实际系统配置，裁剪、链接相应的资源[13]。另外 RTEMS 提供了大量的资源管理接口，很大程度上加速了应用程序开发。

RTEMS 提供简单和灵活的实时多处理器功能。RTEMS 的内核既适用于紧耦合，又适用于松耦合目标系统硬件的配置。此外，RTEMS 还支持由同构和异构混合的处理器组成的系统。虽然 RTEMS 提出了对多处理器进行支持的设计思路并对多处理器支持层的接口进行了定义，但是具体到某个系统，因为涉及到系统结构的不同，多处理器通信层的实现需要根据实际的硬件结构进行设计[14]。

在对支持多核的操作系统的研究和设计中，多核的任务分配是首要解决的问题，是提高系统性能的重要途径[15]。国内外很多组织都对其进行了深入的探讨。

S.Vakili 等[16]比较了静态调度和动态调度，指出静态调度在寻找调度映射的最优解上具有潜在的优势，因为程序员或者编译器能够完整地了解应用程序并能够比较各种不同解的优劣。而动态调度需要迅速地根据可用资源信息和待调度任务的特性来完成调度，因此不可能详细地比较各种方案。另一方面，对于静态调度，系统资源必须在调度前已经确定并且不能改变，因此静态调度限制了并行程序的扩展性，而动态调度则不存在这个问题。在静态调度的多核系统中，控制信息和同步信息包含到程序中，但动态调度并非如此，因此一般需要一个专门的调度单元来完成控制和同步。调度单元可以由操作系统的模块、中间件或者硬件实现。

J. Barbosa 等[17]指出动态调度和静态调度的目的并不完全相同：静态调度追求最小化单个应用的调度开销，获得尽量短的执行时间；动态调度的目的是提高整个系统的性能，特别是在多核系统执行多个并行应用的情况下。动态调度可以从调度策略（scheduling strategy）和调度算法两个方面来进行研究。调度策略决定何时、对哪些对象进行调度，调度算法决定如何根据并行任务的特性和系统负载的情况来分配和调度并行任务。文中提出了两种调度策略，分别是即时策略和批策略，即时策略只对新的并行应用中的任务进行调度；批策略则在有新的应用执行时，对各个节点上已分配但还未执行的任务进行

回收，和新的应用一起重新进行调度。从调度结果上比较，批策略能具有更好的性能，但批策略在实现上需要关注各个节点上分配任务的执行情况，其过程更加复杂，在回收-重分配的过程中可能会引入额外的拷贝开销，因此当应用的规模较小时，批策略不会提升性能。

Juan 等[18]使用了中间件来实现任务的映射。该系统中包含了一定数目的可重配置单元(Reconfigurable Units, RU)以及处理器、DSP、GPU 等，在可重配置单元上任务可以被重新配置并执行，由中间件负责将任务分配到处理单元上。为了在找到一个较好的调度方案的同时不引入过多计算开销，该系统使用了一种混合的设计/运行时映射方法。在设计时并行应用的任务图被分析并得到一些在调度时会使用的有用信息。该系统使用权重、延时以及迁移率来描述任务的特性，权重决定运行时调度算法重配置的次序，重配置的第一批单元将对任务图中的关键路径产生重大影响；延时指明了每个任务重配置引入的延时，延时较大的任务将被赋予较高的优先级；迁移率则用来在重配置过程中取消重配置任务所在节点所进行的调度优化。此外，他们还比较了进行重配置前后的并行应用的性能变化以及软件和硬件实现调度器的性能开销。软件实现产生了巨大的开销，这是由于调度中使用了复杂的数据结构以及较多的软硬件通信。

Zexin Pan 等[19]使用专门的硬件调度模块（Dynamic Task Scheduling Unit, DTSU）来完成动态任务调度，DTSU 模块中包含 Task table、Task Issue Unit、Task Priority Assignment Unit、Task Queue、Task Scheduler 等几个子模块系统中的处理单元分为 Idle、Reconfiguration、Wait、Switch 和 Run 等几个状态，在不同的状态下处理节点可执行不同的操作。DTSU 单元用来执行任务调度算法，根据处理节点的资源信息，包括处理节点所处的状态、任务的表示号和任务类型等来调度任务队列中的任务。DSTU 在两种情况下执行，第一种是系统中某个逻辑单元或处理器进入了空闲状态，第二种是任务队列中有新的并行应用进入时。

1.3 主要研究内容

论文的研究内容主要包括以下几个方面：

1）剖析 RTEMS 操作系统内核，为对多核的支持研究奠定基础。主要探讨了 RTEMS 体系结构和微内核设计思想，并对 RTEMS 内核中重要构件如任务管理、存储管理、信号管理等进行分析。

2）研究 RTEMS 系统对多核处理器的支持，主要工作集中在探讨 RTEMS 多处理器支持模块的实现机制。针对单核下的同步与互斥机制不能直接应用在多核系统中的问

题，提出中断自旋锁和任务自旋锁相结合的机制保证多核的同步与互斥。

3）针对 RTEMS 系统多核任务调度采用统一分配策略带来不足，在对现有调度策略研究的基础上，提出两级动态任务调度算法。该算法根据系统资源动态变化，自动进行任务调整，以充分发挥多核系统并行性，提高系统整体性能。

1.4　论文的组织结构

论文共分为 5 章，具体内容如下：

第 1 章阐述课题背景与研究 RTEMS 多核化操作系统的意义。

第 2 章对 RTEMS 内核进行深入剖析， 包括 RTEMS 体系结构，以及构成内核的核心组件。

第 3 章研究 RTEMS 操作系统对多核处理器的支持，探讨了 RTEMS 多核化实现机制，提出改进的自旋锁机制保证多核的同步与互斥。

第 4 章研究 RTEMS 多核任务调度策略。针对目前 RTEMS 内核在多核任务调度实现上采用统一分配带来不足，研究现有任务调度策略，并在此基础上提出一种两级动态任务调度算法。

第 5 章设计实验验证新算法的可行性和高效性。

论文最后对工作进行总结，指出主要的研究内容和不足之处，并对进一步工作提出了设想。

第 2 章　RTEMS 操作系统内核剖析

RTEMS 对多核处理器的支持是在原有系统的基础上进行改造与扩展的，因此对 RTEMS 操作系统的内核进行深入剖析，是 RTEMS 多核化研究得以顺利进行的前提。

2.1　RTEMS 简介

RTEMS 操作系统全称为实时多处理器操作系统，于 1993 年美国研制开发，并用于本国的国防控制系统，1999 年开始对外开放源代码，后由 OAR 公司负责升级和维护，目前已被广泛应用于通信、航空航天、工业、科研、民用领域。其主要具有以下几个特点[20]：

① 支持同构多核处理器和异构多核处理器；

② 采用非面向对象的编程语言实现面向对象的编程思想，程序设计具有模块化特点，易于移植和重用；

③ 支持多任务；

④ 支持多种调度算法，包括基于事件驱动的任务调度算法、基于优先级的任务调度算法和占先的调度算法；

⑤ 支持任务间的通信和同步机制；

⑥ 支持任务间的优先级继承算法，防止优先级反转现象的发生；

⑦ 支持动态内存分配，用户可配置；

⑧ 支持丰富的 API 接口，除专有的 API 之外，还提供对标准 API 接口的支持，包括 POSIX API、ITRON API，应用程序的可移植性非常好。

RTEMS 操作系统是一款非常优秀的实时嵌入式操作系统，由于最早用于美国国防控制系统，因此研制初期就瞄准了高可靠性、高稳定性及实时性，并且采用了面向对象和构件的先进技术理念。

实时操作系统最重要的一个特性是实时性[21]，即当有外部事件发生时能够在规定的时间内对其进行响应，并且事件处理的延迟时间、处理的持续时间、处理的截止时间都有下限和上限，所有的动作都必须在这个时间内完成。RTEMS 操作系统支持多种任务调度算法，并且提供了一种可快速响应外部中断以满足事件对时间严格限制的机制，加之简单的内存分配策略，系统的实时性得到了保证。

实时操作系统另一特性是对大规模并发进程的处理能力[21]。由于受到处理器和硬件资源的限制，即使任务是并发的，指令也只能一条一条执行。虽然目前很多理论提出了

解决多进程并发问题的办法，但是设计者对于多进程之间的同步与异步处理仍头痛不已。RTEMS 使开发者从繁杂的工作中解脱出来，在该系统中一个程序中可包含多个任务，并且每个任务都是可控的，逻辑上都在同步执行，多个任务之间的硬件资源管理协调工作由系统组件完成，大大简化了开发者的工作。

另外，RTEMS 提供了简单、灵活的实时多处理器功能。RTEMS 的内核既适用于紧耦合，又使用于松耦合目标系统硬件的配置。同时还对同构、异构以及混合的多核处理器系统提供了很好的支持。

综上所述，对于实时嵌入式操作系统，RTENS 是一个非常不错的选择。

2.2 RTEMS 体系结构

RTEMS 体系结构采用了分层的设计模式[22]，每层由若干模块构成，这些模块协同工作，为实时应用系统提供一系列服务，其总体结构如图 2.1 所示。

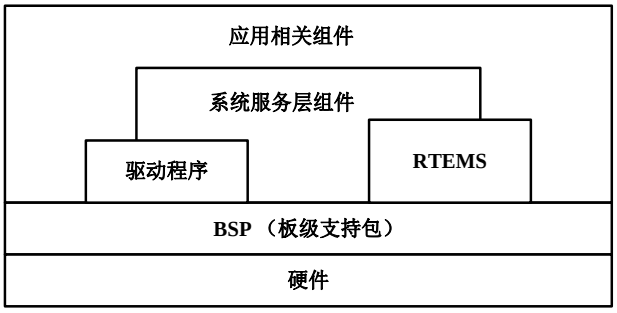

图 2.1 RTEMS 体系结构图

其中 BSP 包括底层硬件相关的所有代码实现，完成硬件初始化工作，包括时钟、串口控制台、定时器、硬件 RTC、引导设备、NVMEM 等，为驱动程序和 RTEMS 操作系统的运行创造环境。由于 BSP 是针对目标板进行设计的，因此随着目标板结构和功能的不同，BSP 的设计也会有很大的不同，因此，BSP 设计是体现系统可移植性的重要方面。

由图 2.1 可以看出，RTEMS 内核在应用层和底层硬件之间起到了一个桥梁的作用，其屏蔽了底层硬件的具体实现，为上层提供服务。内核包括以下管理器：存储工作空间管理器、看门狗定时器管理器、对象管理器、线程管理器、线程队列管理器、内核信号量管理器、内核消息队列管理器、多级处理器管理器等。

I/O 管理将驱动程序整合到一起加入到系统中，主要包括硬盘、网络、串并口等驱动，这样应用程序在访问硬件资源时可以使用统一的机制。

系统服务层提供面向应用的系统资源管理器，其包含的各组件可根据用户需求灵活配置，包括：初始化管理器、多任务管理器、中断管理器、时钟管理器、定时器管理器、信号量管理器、消息队列管理器、事件管理器、信号管理器、存储器分区管理器、存储

器区域管理器、双端口存储器管理器、I/O 管理器、致命与异常管理器、单调周期管理器、用户扩展管理器、多处理器管理器。

RTEMS 使用了微内核的设计思想，内核只实现了最基本的操作系统功能，负责多种机制的实现，而把策略的制定留给各种管理器。对内核的深入剖析，是对操作系统功能进行扩展的关键所在。

2.3　RTEMS 内核剖析

2.3.1　对象

利用非面向对象的编程语言实现面向对象的思想是 RTEMS 的一大特色[23]。RTEMS 内核将所有资源都视为对象，如任务，互斥量，信号量，存储空间，消息队列等等。面向对象的思想符合人们的思维习惯，便于开发，并且代码的可重用性、可扩展大大增强，很大程度地缩短了应用程序开发周期，而且易于维护。

内核提供了对对象操作的基本接口，包括对象的创建，删除，查看等。

RTEMS 内核通过为每一个对象分配唯一 ID 进行标识，对象的名字由用户实现，因此，不同对象允许使用相同的名字。每一个对象 ID 存储于 32 位无符号整数结构内，结构如图 2.2 所示。

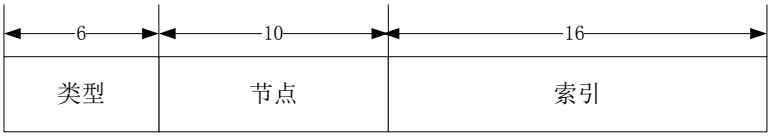

图 2.2　对象 ID 存储结构

其中前 6 位表示对象的类型，中间 10 位表示对象所属的节点。在多核处理器上，内核通过分配节点号来区别不同处理器，这样最小化了处理器之间的区别，有利于管理。最后 16 位区别属于相同类型的不同对象，因此每种对象类型可包括 65535 个对象。

RTEMS 每一种对象类型都对应一个对象信息表结构，存储属于该对象类型的所有信息，其中所有属于该对象类型的本地实例通过双链表进行管理。实例控制块的结构定义如下：

```
typedef struct                    /*实例控制块的定义*/
{
    Chain_Node Node;              /*双向链表的节点*/
    Objects_Id id;                /*实例的识别编号*/
    Objects_Name name;            /*实例的名称*/
```

} Objects_Control;

对象控制块的第一个数据成员为实例控制块，而实例控制块的第一个成员即为节点，无论这个小内存块被解释成双向链表的节点也好，或解释成互斥体对象也好，都不影响系统对链表的操作。这里可以看成对象控制块继承了双向链表的节点，这是 C 语言表达 OO 思想的一种方法。

2.3.2 任务管理

任务是操作系统资源分配的最小单位，也是运行的最小单位，拥有独立运行环境[24]。每一个任务都对应一个任务控制块，保存着以下信息：（1）等待信息，当任务处于阻塞态时，记录等待信息，以便条件满足时由阻塞态进入运行态；（2）上下文控制信息，当任务被打断时，记录任务恢复时需要的信息，可以保证任务从中断位置继续运行；（3）任务所在的就绪链表，当任务进入或脱离就绪态时，能从相应链表中加入或删除。

任务调度算法是嵌入式实时操作系统实现实时性的关键，RTEMS 操作系统采用基于优先级的抢占式调度算法，对于处于同一优先级的任务则采用轮转调度算法。

（1）基于优先级的抢占调度

RTEMS 支持 256 级的优先级（0-255），数字越高，优先级越低，系统中存在一个 idle 任务，优先级最低，不做任何事情，只是空循环，以防止无任务时系统崩溃。

为了保证每个时刻运行的都是最高优先级的任务，RTEMS 采用将最高优先级的任务放在就绪队列的首部，当添加新任务的时候，将该任务放在同一优先级的尾部。这样可以保证调度模块以最短的时间调度优先级最高的任务，满足了系统的实习性。

RTEMS 就绪队列通过多级链表实现，每个优先级对应一个链表，链表中存储属于该优先级的就绪状态任务。

为了满足 o(1)时间内完成调度，RTEMS 利用了_Priority_Major_bit_map 和_Priority_Bit_map[16]两种数据结构，结构如图 2.3 所示。

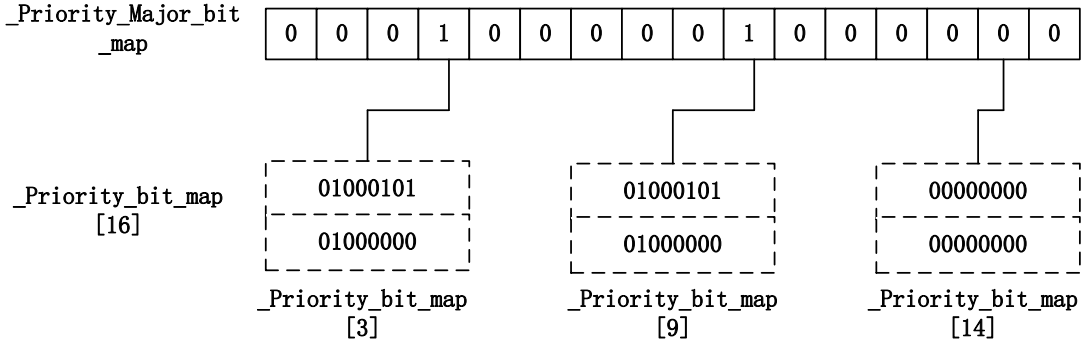

图 2.3 优先级数据结构

_Priority_Major_bit_map 记录每 16 个优先级段中是否有任务存在，每个优先级段对应_Priority_bit_map 数组中的一个元素，每个元素都是 16bit，每一位置 1 表示该优先级有就绪任务存在。任务调度管理模块通过该位图能快速定位目前就绪队列中最高优先级任务，直接运行。

在任务运行过程中，如果有更高优先级任务到达，当前任务就会被抢占，运行更高优先级任务。用户在创建任务的时候可以设置抢占模式，如果设置禁止抢占，则在任务终止、阻塞或重新设置抢占模式之前，即使有更高优先级的任务出现，也不会终止。

（2）轮转调度

处于同一优先级的就绪任务，采用轮转调度算法。创建任务时，系统会为每个任务分配一个时间片，时间片随着时钟到来而不断减少，用完之后放弃处理器，并将该任务放到链表末尾，如果队列中还有其余的就绪任务则调度队列中首元素，如果没有则为该任务重新分配时间片，获得处理器后重新运行。

（3）优先级反转

系统运行过程中可能会出现高优先级的任务无法运行，而低优先级的任务先要运行的情况。例如当前系统有一任务 A 正在运行，并且获得互斥量进入临界区，此时任务 B 产生，并且其优先级高于 A，根据优先级抢占调度算法规则任务 B 获得处理器，进入执形态，任务 A 进入阻塞态。又有一任务 C 产生，其优先级高于 A、B，因此中断任务 B，获得处理器。如果任务 C 也需要获得互斥量进入临界区，但是此时该临界区的互斥量正在被任务 A 占有，未被释放，所以任务 C 进入阻塞态，通过系统调度任务 B 获得处理器得以运行。示意图如图 2.4 所示。

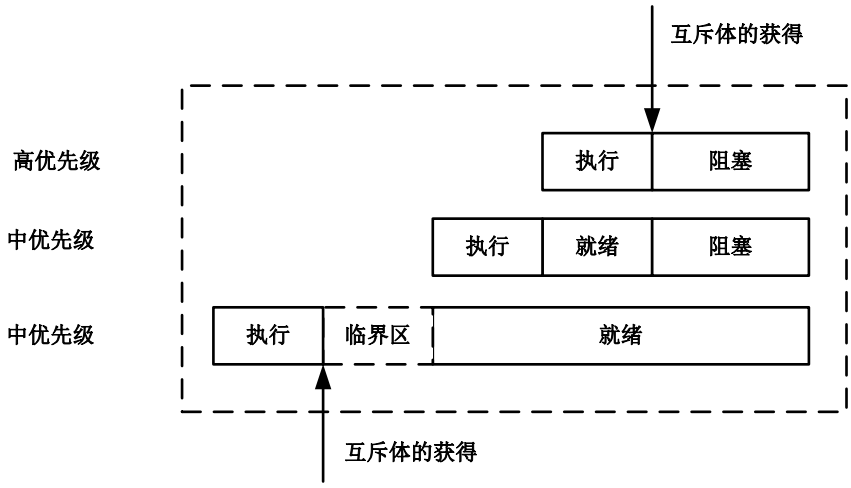

图 2.4　优先级反转

这显然违背了高优先级任务需要先被执行的原则，解决办法是采用优先级反转。首先将获得互斥量的低优先级任务的优先级调整为高优先级任务的优先级，这样低优先级获得处理器进入执形态，完成后，释放互斥量和处理器，并将优先级降为最初值。此时高优先级任务获得互斥量，进入就绪态，等待调度，从而解决了优先级反转问题。

2.3.3 内存管理

（1）固定大小内存分区管理

分区管理指每次为用户提供大小相等的内存区域[25]，命令包括：

- rtems partition create 创建一个分区
- rtems partition ident 获得分区标识号
- rtems partition deletel 删除一个分区
- rtems partition get buffer 申请一个内存块
- rtems partition return bufier 释放一个内存块

分区将物理地址上相邻的一块内存区域分成大小相同的缓冲区，空闲缓冲区由双链表进行管理。当从分区中申请内存块时，按照空闲内存块链表的顺序分配。如果空闲空间不足，调用者不会被阻塞，而是获得一个空指针，以确保申请内存调用的时间确定性。释放内存块时， 将该内存块挂在空闲内存块链表的链尾。分区被删除时将释放出这段连续的内存空间。

（2）可变大小内存区域管理

区域管理指每次根据用户需求分配大小不等的内存区域[25]，命令包括：

- Rtems region create 创建一个分区
- Rtems region ident 获得分区的标识
- Rtems region_delete 删除一个分区
- Rtems region_extend 扩展一个分区
- Rtems region get_segment 申请一个段
- Rtems region return_segment 释放一个段

区域管理最小分配单位是块，块的大小由用户进行限定，但是必须是页的整数倍，例如如果页的大小为 256 字节，用户需求为 350 字节，那么区域管理将为用户分配 512 字节大小的块。

所有的空闲块链接成一个双向链表，分配采用首次适应算法，即每次从链表首部开始查询，遇到满足用户需求大小的块时立即进行分配，为了减少内存碎片，系统设置一

个临界值，如果实际块的大小减去用户需求的大小后超过该临界值，则将该块分为两块，一块分配给用户，另外一块重新放入到链表中。

同理释放块时，区域管理首先检查左右相邻的块是否和该块的地址连续，如果是则进行合并，形成更大的块，然后放入到链表中，如果只有一块和该块相邻，则只与相邻块进行合并，否则直接将该块放入链表结尾。

（3）双端口内存管理

双端口内存(DPMA)是指多个处理器或智能 I/O 处理器均可访问的内存区域，RTEMS 的双端口内存管理器为双端口内存区(DPMA)提供了地址变换功能。命令有：

- rtems_port_create- 创建端口
- rtems_port_ident- 获取端口的 ID 标识
- rtems_port_delete- 删除端口
- rtems_port_external_to_internal-将双端口内存的外部地址变为内部地址
- rtems_port_internal_to_external- 将双端口内存的内部地址变为外部地址

双端口内存属于某一特定处理器，该处理器访问时直接使用内部地址，其余处理器或智能 I/O 处理器通过外部地址进行访问，双端口管理器完成外部地址与内部地址的转换。

2.3.4　同步互斥机制

在支持多任务应用程序的系统中，一次工作的完成往往需要多个任务或多个任务与多个中断程序配合、协调完成。为此，系统必须提供任务间的同步、互斥与通信机制，保证工作顺利完成。RTEMS 提供了一套丰富的机制来满足上述要求，包括抢占锁和关中断，信号，信号量，消息队列和事件等[26]。

（1）抢占锁和关中断

抢占锁是实现任务间互斥的一种最基本的机制，当系统中的临界资源禁止任务抢占时，称为抢占上锁，简称为抢占锁。其互斥实现是通过将临界资源放在_Thread_Disable_dispatch()和_Thread_Enable_dipatch()函数之间，这两个函数都是对全局变量计数器 _Thread_Dispatch_disable_level 进行加减操作，其中_Thread_Disable_dispatch()进行上锁操作，对全局变量_Thread_Dispatch_disable_level 进行加一操作，_Thread_Enable_dipatch() 进行解锁操作，对全局变量_Thread_Dispatch_disable_level 进行减一操作，抢占锁支持嵌套。但是抢占锁只实现了任务之间的互斥，并不支持任务与中断之间的互斥，因此当在禁止抢占期间，有可能有

中断到来使得更高优先级的任务获得临界资源进入就绪态。

关中断是一种最强硬的互斥机制，不仅实现了任务之间的互斥，还实现了任务与中断，中断与中断之间的互斥。关中断一般在系统调用内核数据结构或进行上下文切换时使用，在这种情况下禁止对任何外部中断的响应。关中断机制通过_ISR_Disable()和_ISR_Enable()两个函数实现，其中_ISR_Disable()实现关中断操作，_ISR_Enable()实现开中断操作。

（2）信号量

信号量是一种非常常用的同步互斥机制，既可实现任务之间的互斥，也可实现任务和中断之间，中断和中断之间的互斥，包括三种类型：普通二值信号量、可嵌套二值信号量和计数信号量。

普通二值信号量，又叫互斥信号，只能取值 0 和 1 两种情况，不支持嵌套，否则有可能发生死锁。当互斥信号为 1 时，任务可以进入代码临界区，并且信号值对应减一，当其余任务也要求进入时，会发生阻塞。该任务完成对临界区的访问时，对互斥信号进行加一操作，释放互斥信号，可从等待该信号量的任务队列中选择一任务进行分配。普通二值信号有可能发生优先级反转现象，RTEMS 通过采用优先级继承算法解决该问题，即使低优先级任务的优先级继承高优先级任务的优先级，重新调度，获得处理器后进入执形态，退出临界区后释放互斥量，优先级再恢复为之前的优先级。

可嵌套信号量除了取值只有 0 和 1 两种情况外，还允许进行嵌套。当发生嵌套时，只是嵌套层次进行了加一操作，互斥信号并没有进行加减操作，嵌套层数表明被同一任务访问次数。

计数信号量主要解决共享多资源的问题，其值等于系统中实际资源个数。

信号量解决同步互斥问题是通过任务等待队列实现的，每钟类型的信号量对应一个队列，申请某一资源而被阻塞的任务将被挂在其对应队列下。当有任务释放资源时，检查该队列是否挂有阻塞任务，如果有，进行选择，直接将资源对其进行分配，而不改变信号量状态；如果没有，则对信号量进行加一操作。等待队列有两种实现策略：基于优先级和先来先服务。先来先服务指按照阻塞的顺序对资源进行分配，先被阻塞的任务放在队列的头部，而不管任务的优先级。基于优先级是指，若有空闲资源时先分配给高优先级的任务，为了加快检索速度，实现时利用 Chain_Control 结构，其中包括四个元素，每个元素对应 64 个优先级，当有任务阻塞时，首先除以 64 取余，找到对应的元素，然后将其优先级与 0x20 做比较，如果小于则从开始进行查找，如果大于则从尾部进行查找，放入到对应优先级的队列中。

（3）消息队列和事件

消息队列是实现任务间或任务与中断间的通信机制，其中消息是一块数据区域，其大小和类型由用户定义。进行数据通信时，发送方将消息放在目的方的消息队列中，如果队列满，则阻塞该操作；接收方从其对应队列中取出消息，如果该队列无消息，则进入阻塞态，注意如果该操作处于中断处理中，则不会阻塞，而只是返回“条件不满足”。因为消息队列是通过内容拷贝实现的，因此不适合大量数据的通信，在这种情况下可以考虑通过在消息中存储指针或共享内存区域的办法来解决。

事件相对消息队列来说更快捷，效率更高，并且实现了一对多的同步，但是不支持数据通信。具体实现是，每个事件对应一个标志位，只有发生和不发生两种情况，不记录其余信息。每个任务可支持 32 种事件，可对其感兴趣的位进行设置，并且通过其余任务或中断对事件进行触发，同时通知所有关心该事件的任务，由此达到同步和通信的目的。

（4）异步信号

异步信号是 RTEMS 中任务之间和任务与中断间异步通信的唯一机制，当有异步信号到达时，任务中断当前的执行路径，调用相应的异步处理函数对异步信号进行处理，处理完毕后再恢复中断了的执行路径。

RTEMS 定义了一个 32 位整数 rtems_signal_ret 信号集，每一位代表一种信号，异步信号处理函数 ASR 通过调用 rtems_signal_catch 函数进行注册，所有信号都交给一个 ASR，ASR 通过判断信号的类型进行不同的处理。异步处理函数在进行上下文切换时调用，一是当中断返回时，先判断是否有异步信号到达，如果有则先处理异步信号；另一个是当任务进入执形态，重新开始运行时，执行之前代码之前先判断是有有异步信号到达。

2.3.5　中断管理

中断管理为系统提供一种快速响应外部事件的机制，任何任务只要有中断到来都必须终止，系统继而调用相应的中断处理函数。中断有 256 级，0-255，数字越大优先级越低。

中断管理提供的处理中断命令包括：

- _rtems_interrupt_cache-Establish an ISR
- _rtems_interrupt_disable-Disable Interrupts
- _rtems_interrupt_enable-Enable Interrupts

- _rtems_interrupt_flash-Flash Interrupt
- _rtems_interrupt_is_in_progress-Is an ISR in Progress

中断产生之后，首先根据中断向量值查找中断向量表，调用相应的中断处理程序。在中断处理程序中保存被中断任务的内容并调用用户中断处理程序。恢复被中断任务的内容，中断返回。从中断产生到用户中断处理器程序处理完毕，就是系统相应中断响应的时间。中断响应流程图如图 2.5 所示。

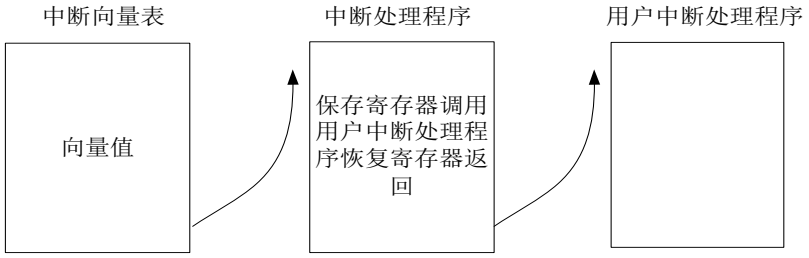

图 2.5　中断流程图

2.3.6　对多处理器的支持

RTEMS 对多核处理器提供了强而有力的支持，不但支持同构多核处理器，还支持异构和混合多核处理器。这是通过多节点的方式实现的[27]，多核处理器中每个处理器都分配一个唯一的节点编号，在多处理器节点配置表中进行定义，处理器由分配的节点编号进行区别。RTEMS 在每个处理器节点中都有一个局部对象表和全局对象表，以此来区分节点执行任务类型，系统据此决定下一步行为。局部对象表中的对象只能由该节点访问，而全局对象表中的对象所有节点都可以访问。RTEMS 的设计目标是跨越所有软硬件的边界，在逻辑上成为一个统一的整体，而这一切都是通过用户层和内核层之间的多核处理器支持层来实现的[28]。

2.4　本章小结

首先对 RTEMS 进行介绍，然后深入剖析了 RTEMS 体系结构和内核的核心组件，包括任务管理组件、内存管理组件、同步互斥与通信机制、中断机制及对多处理器的支持，为后文对多核化关键技术的研究提供理论支持。

第 3 章　基于 SMP 方式的 RTEMS 多核化实现机制

虽然 RTEMS 提出了对多核处理器支持的设计思路和 MPSL 的接口定义，然而具体硬件体系结构不同，多核化设计过程中需要实现机制不一样。主要探讨 RTEMS 对 SMP 系统提供支持的关键技术，针对单核下的同步与互斥机制不能直接应用在多核系统中的问题，提出改进的自旋锁机制保证多核的同步与互斥。任务调度策略在并行系统中发挥重要作用，严重影响着系统的性能，将作为一项关键技术在第 4 章探讨。

3.1　RTEMS 多核化硬件体系结构

对并行系统结构、存储结构模型以及通信机制的了解是探讨和选择 RTEMS 多核化过程中所需机制的依据。

（1）并行系统结构分类

并行系统的结构按照存储器结构可以分为两大类：集中式共享存储结构和分布式存储结构[29]。

集中式共享存储结构中，所有处理器利用一根总线共联实现共用一个存储器，这种情况下，所有处理器访问内存的时间相同，并且处理器的个数容易扩展，但是随着处理器个数的增加，内存访问宽带必定会成为系统瓶颈。集中式共享存储结构图如图 3.1 所示。

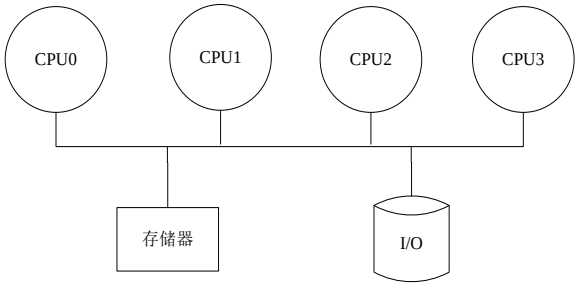

图 3.1　集中式共享存储结构

集中式共享存储结构只适用于处理器数量较少的情况，当处理器数目很多的时候应该采用分布式存储结构。分布式存储结构中，每个处理器都独有存储器，这样可以解决进行海量数据处理时带来宽带瓶颈的问题。但同时也使得处理器间通信复杂，远程节点访问延迟变大[30]。分布式存储结构图如图 3.2 所示。

（2）存储结构模型

目前有两种可供选择的存储结构模型，一种是将所有存储结构视为逻辑上统一的一个存储器，所有处理器对其进行统一编址，而不管存储具体位置，这种存储结构模型适合处理器间通信频繁的情况。第二种是每个存储器都含有私有地址空间，每个处理器的

私有地址并空间不允许其余节点进行访问。即使访问地址相同，也是指向不同的存储位置，这种模型势必增加了处理器间通信的复杂度[31]。

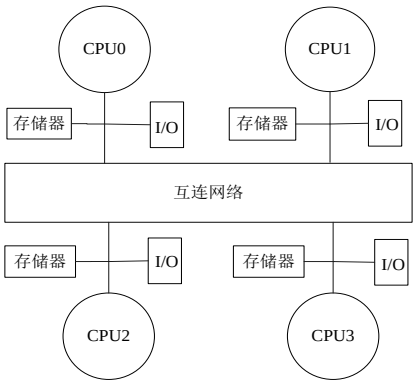

图 3.2　分布式共享存储结构

（3）通信机制

不同的地址空间组织方式采用不同的通信机制，对于共享存储结构的系统，可以利用指令中的地址进行隐式的数据通信。对于采用私有地址进行编址的系统，必须采用消息传递机制进行显示通信[32]。

采用消息传递进行通信的系统利用简单的协议来完成处理器之间通信。例如，当发送方需要向远程节点发送数据或要求获取数据时，首先向对方发送请求，目的节点通过轮询或中断的方式接收该请求，并做相应的处理，处理结束后生成相应的响应消息，发送到请求方。如果需要目的节点进行数据传输，则接收方继续等待，直到数据传输结束。上述方式属于同步传输，在目的节点的响应消息或数据未到达之前，接收方的任务将一直被阻塞。如果节点直接发送消息到远程节点，而不是等待发送方请求，这种方式属于异步传输，节点发送完数据后不用等待目的节点的响应，可以继续执行原有的任务[33]。

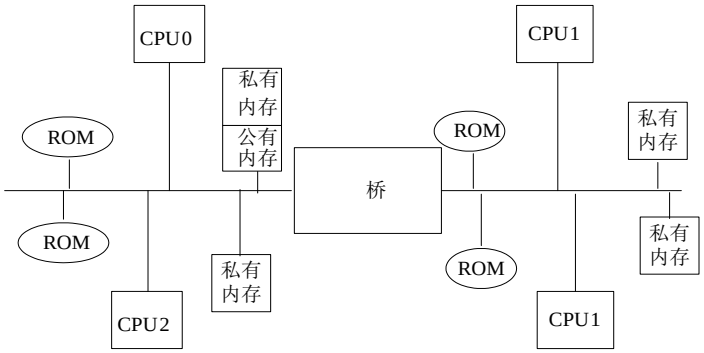

图 3.3　硬件系统结构

（4）总体架构

本文研究的 RTEMS 多核化系统支持如上图 3.3 所示的硬件体系结构，该结构采用

分布式存储结构，且每个节点都有含有私有地址空间。另外，0 节点单独开辟一段地址空间，作为系统所有节点进行通信的共享存储区域。

3.2　RTEMS 系统多处理器支持层

RTEMS 系统对多处理器的支持关键是完成 MPSL 层提供功能[34]。MPSL 层隐藏了底层多处理器之间通信的具体实现，加快了应用层程序的开发进度；同时减轻了内核的工作量，有了 MPSL 层的存在，内核主要完成对本地对象的操作和判断全局对象的具体物理位置等基本功能。

RTEMS 系统提出了 MPSL 的接口定义，为了增强 RTEMS 可移植性，选择保留原有接口。

MPSL 层继承了 RTENS 系统层次化和模块化的设计模式[35]，方便了不同硬件平台之间的移植工作，满足了嵌入式系统对通用性的要求，便于开发。

MPSL 层分为以下三层：MPCI 接口层，共享内存实现层，硬件相关实现层。

3.2.1　多处理器支持层提供的接口

该组接口登记在多处理器通信接口表 MPCIT(MulityProcessor Communication Interface Table)中。MPSL 层主要负责对缓冲池的管理，缓冲池中的基本单位为包，通过对包的操作实现多个节点之间的通信。

包括以下几个接口[36]：

- 初始化 MPSL：initialization
- 获取一个包缓冲：get_packet
- 发送一个包缓冲：send_packet
- 接收一个包缓冲：receive_packet
- 释放一个包缓冲：return_packet

3.2.2　MPCI 接口层

该模块要求实现节点之间进行消息传递的通信机制，为上层应用提供了远程通信需要的接口例程[37]。

应用层访问对象分为两种：局部对象和全局对象。局部对象只有本地节点才能进行访问，而全局对象则可由任一节点进行访问。在每个节点中都有一个用于存储对象信息的局部对象表和全局对象表。所有节点中的全局对象表保持一致。当节点需要访问全局对象时，从该表中查找全局对象的相关信息，确定全局对象的位置后进行访问。对局部

对象的操作，直接交由内核处理，与单处理器下的情况相同。而对全局对象的处理，则需要调用 MPCI 接口层的例程来完成。应用组件根据对象的属性和位置来判断访问的对象属于局部对象，还是全局对象，进而选择调用相应的处理模块。

为了简化系统对于远程请求的处理，采取在每个需要进行远程通信的应用组件中实现 MP（mulityprocess）模块策略[38]，该模块生成一个进程，专门负责监视通信介质，即共享内存中是否有消息到来。当有消息到来时，该代理被唤醒，作为远地节点在该节点中的代理，向内核提交请求，完成相应请求后再将反馈信息发送到被代理节点，完成一次通信过程。

一般情况下，节点之间的通信都是由一方发出请求，等待对方回应后开始进行数据的传输，下面，以节点 1 与节点 2 之间通信的全过程为例，说明 MPCI 层的通信机制，如图 3.4 所示。

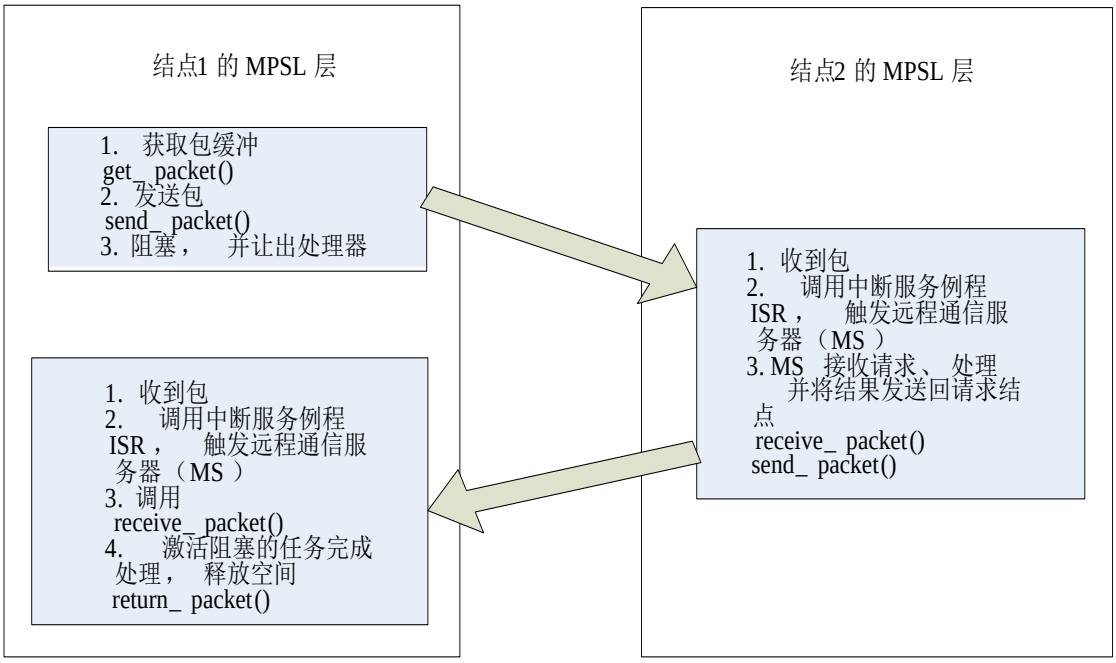

图 3.4 远程通信过程

3.2.3 共享内存实现层

共享内存层的主要功能是实现 MPCI 层的包交换机制以及机间通信中断机制[39]。下面首先给出共享内存层的组织结构，在此基础上深入探讨 MPCI 层的包交换机制以及机间通信中断机制。

（1）共享内存层的组织结构

共享内存层的组织结构如图 3.5 所示。

共包括三个域[40]：节点状态控制块，通信队列控制块和 Envelope 控制块。

- 节点状态控制块

节点状态控制块主要记录各节点的状态信息，方便操作系统掌握各节点的状态信息，利于管理。其中状态信息主要包括节点的运行状态（未觉，准备好，活跃），中断值，中断入口地址和错误码。

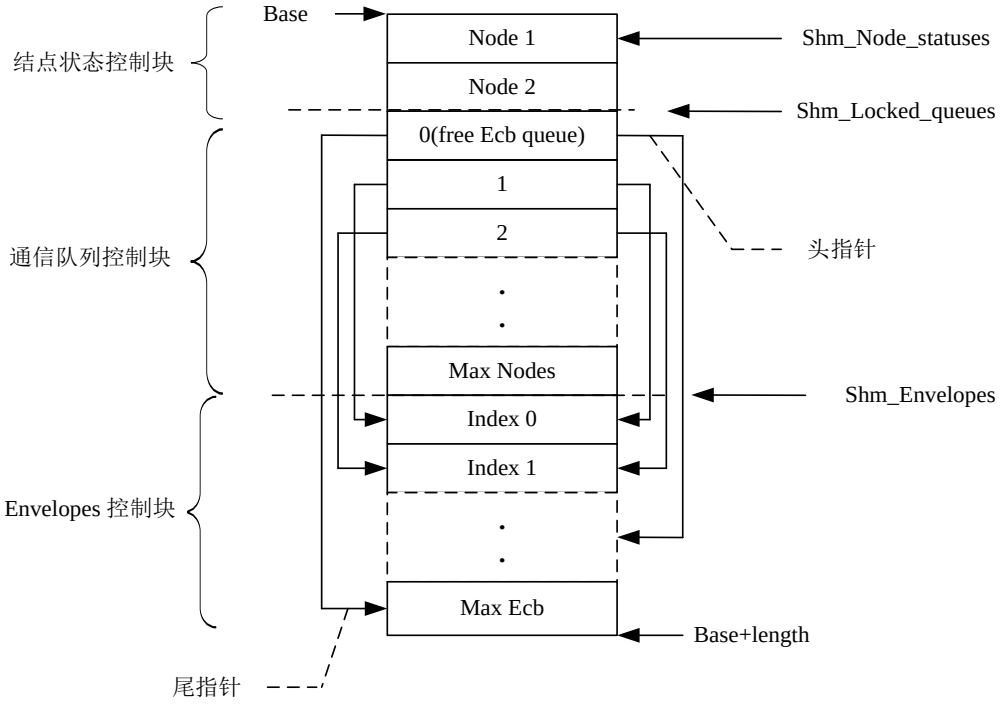

图 3.5　共享存储区组织结构

- 通信队列控制块

第二个区域属于通信队列控制块区域。为实现各节点之间的通信，系统为每个节点分配一个队列，记录从其余节点发过来的信息包[41]。如果系统中有 n 个节点，则共享内存中存在 n+1 个队列。如本例中共有 2 个节点，该区域内实现了 3 个队列，其中队列 0 用于管理空闲的 envelope 包。为便于队列管理，在第二个区域中存储每个队列的控制块。

- 交换区

共享内存的第三个区域保存所有的包，包结构中保存通信内容和所属队列等信息。

（2）多处理器间中断机制

在分析完共享内存层对 MPCI 接口层的支持后，开始探讨如何实现多处理器间的通信触发机制，即当有远程请求到达目的节点的消息队列后，采用什么样的机制通知节点，以启动 MS（远程通信服务器），执行相应的处理。通信触发机制有两种：软件轮训方式和中断方式[42]。由于采用轮训方式 CPU 利用率比较低，所以主要研究通过采用中断方式

完成 MS 的启动工作。中断流程已在 2.3.5 节进行了分析。

为了确定系统中 MS 服务器的启动时机，在每个节点中设置信号变量 MPCI_semaphore[43]。在系统启动建立 MS 服务器时，该信号量同时被定义，并初始化为 0。开始 MS 阻塞在 MPCI_semaphore 信号量上，直到该节点对应的消息队列中有远程请求到达时，通过中断处理函数将信号量值加 1。MS 服务器捕获到信号量的增加，从阻塞队列中唤醒，进入就绪队列，等待系统的调度。MS 获得 CPU 使用权后，根据请求类型，执行相应的处理工作，同时对信号量执行减 1 操作。该流程一直循环执行，直到信号量值为 0 时，MS 服务器重新被阻塞。

多处理器间的通信中断触发机制流程图如图 3.6 所示。

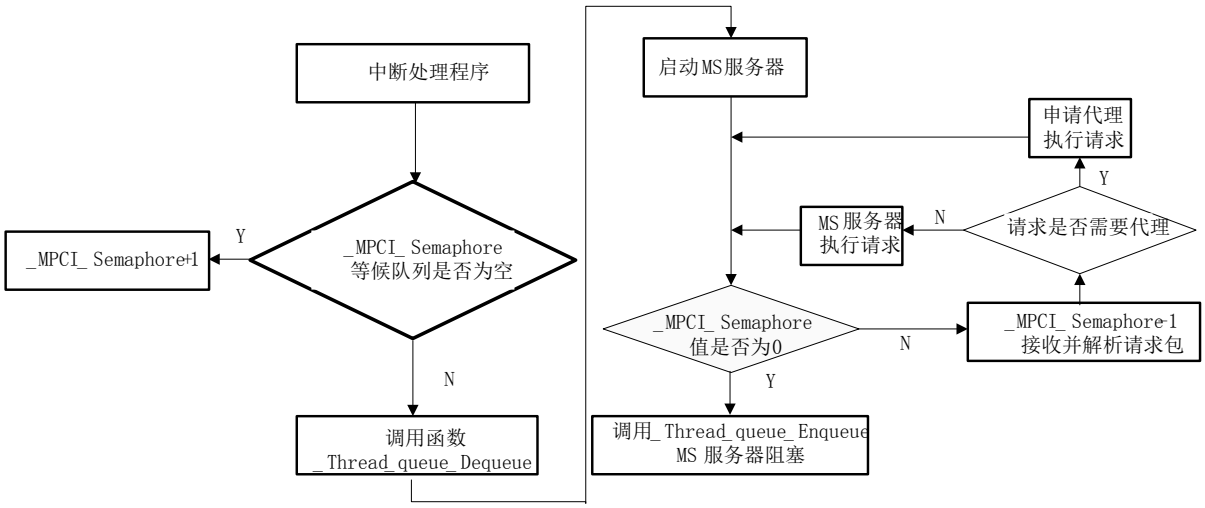

图 3.6　多处理器间的通信中断触发机制流程

图 3.6 中系统调用函数_Thread_queue_Dequeue 表示将 MS 服务线程唤醒，即从阻塞队列中取出；而_Thread_queue_Enqueue 是将 MS 线程再次阻塞，重新加入到阻塞线程队列中。

3.2.4　硬件相关层

由于多处理器间的通信是通过基于总线的共享内存实现的，并且系统采用了机间中断机制来实现对远程请求消息的处理，因此硬件相关层涉及的关键技术是如何实现中断机制[44]。中断机制需要完成的功能很多，包括：共享内存地址信息确定和对中断信息的配置，同步互斥机制和中断触发函数、中断处理函数的具体实现。

已经在共享内存实现层中讨论了中断处理函数的功能，可概括为：释放信号唤醒 MS 服务；清中断位。下面分析中断触发函数完成的功能。

当有远程节点的请求到达时，源节点将中断值写到目的节点对应的地址中，系统根据这些信息来触发中断处理函数，其中中断值和目的节点地址即为中断信息表中记录信息。在 I/O 地址中，有一段地址用来保存每个节点的中断信息。因此中断触发函数需要完成的功能为：取出中断信息表中记录的中断值，并写入目的节点地址中，系统根据该值触发中断处理函数。

在多处理器情况下，很可能在同一时刻有多核多任务在同时运行，显然单核提供的同步互斥机制已不能不满足系统要求，必选提出新的机制保证共享资源的正确使用从而实现任务间的正常通信。下面将重点围绕多核间同步互斥问题进行分析，给出合理的解决思路。

3.3　RTEMS 多核信号量机制

单核下的同步互斥机制已在第 2.3.4 节进行了讨论，主要通过抢占锁和关中断实现。但是在多核环境下，抢占锁和关中断已不能保持原有的语义。

首先，抢占锁的目的是保证任务间的互斥访问，但是其能保持语义的前提是同一时刻只执行一个任务。在多核环境下，除非只是用一个核运行，否则此前提已不复存在，因为同一时刻可能有多个任务运行在不同的核上，仅仅依靠禁止抢占，并不能阻止其余核上的任务进入临界区，如下面情况：

（1）CPU1 从一个特定的内存单元读出，并测试到其中的某一位（假定是 bit4）为 0。然后将这一位改成 1。

（2）在 CPU1 还没有来得及把修改后的结果写回该内存单元之前，CPU2 也从同一内存单元读出，也测试到 bit4 为 0，然后也将这位改成 1。

（3）然后，CPU1 把修改后的结果写回内存单元。由于该内存单元的 bit4 原来是 0，CPU1 便以为取得了该项临界资源，并且确信不会有其他任务前来打扰。在 CPU1 看来，它已经把这个内存单元的 bit4 改成了 1,如果其他任务再来对此项临界资源执行“测试并设置”操作，就会因此而被关在门外。

（4）然后，CPU2 也把修改后的结果写回内存单元，也以为取得了该项临界资源，也确信不会有其他任务可以前来打扰。

可是，实际上这两个处理器核之间的互相干扰已是十分可能的了。这个例子告诉我们，与单核处理器结构相比，SMP 结构对互斥操作的微观“分辨率”需更高，有些在单核处理器结构中的“原子操作”在 SMP 结构中不再是原子的了。

因此，抢占上锁已不能保证任务间的互斥访问。

同样，关中断的互斥机制在多核环境下也不能保持原有的语义。当一个任务访问临界区时，关闭本核的中断的实现方式虽然能保证本核上的任务不被抢占以及任务与本核上的 ISR 之间的互斥，但是其并没有阻止其余核上的任务或者 ISR 访问同一临界区，因为抢占锁已失效且其余核可能处于开中断状态。而关闭所有核的中断虽然能保证运行本核上的任务不被抢占以及与所有的 ISR 之间的互斥，但是由于其并没有停止核上任务的执行，其余任务还是可能访问同一临界区。不管使用哪种实现方式，都有可能出现同时访问同一临界区的情况，从而内核数据结构有可能遭受破坏。因此，关中断机制在多核环境下已不能保证其原有的语义。

因此必须提出新机制代替原系统机制保证多核环境下的任务间、任务与中断以及中断与中断间的互斥。

3.3.1 自旋锁原理

自旋锁很好的解决了多核环境下互斥问题[45]。自旋锁是为实现保护共享资源而提出一种锁机制，用于多个 CPU 系统中。自旋锁与互斥锁相类似，都是为了解决对某项资源的互斥使用。无论是互斥锁，还是自旋锁，在任何时刻，最多只能有一个保持者，也就说，在任何时刻最多只能有一个执行单元获得锁。但是两者在调度机制上略有不同。对于互斥锁，如果资源已经被占用，资源申请者只能进入睡眠状态。但是自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环在那里看是否该自旋锁的保持者已经释放了锁，由此可见，自旋锁比较适用于锁使用者保持锁时间比较短的情况。由于 RTEMS 系统对共享内存队列的操作比较简单，而且调用者不会进入睡眠状态，因此采用自旋锁机制可以提高效率。

在 RTEMS 多核系统中每个核同步使用自旋锁是基于核锁定数据总线的指令。当某个核锁住数据总线后，它读一个内存单元 lock 来判断这个自旋锁是否已经被别的核锁住，如果否，它写进一个特定值，表示锁定成功，然后返回。如果是，它会重复以上操作直到成功。对锁的操作主要包括以下两个过程。

加锁过程（Lock）：设置一个整形变量 lock 为上锁标志，当 lock 值为 0 时代表已上锁，lock 值为 1 时代表未上锁。在内核申请该资源时需要使用加锁函数 RTEMS_SPIN_LOCK()，该函数在加锁前首先判断该锁是否被占有，如果已经被占有，则进入自旋状态，即循环判断该锁是否一直被占有直到该锁被解锁；如果没有被占有，则对其加锁使 lock 值为 0。

解锁过程（Unlock）：解锁过程相对比较简单，调用解锁函数

RTEMS_SPIN_UNLOCK()，在内核使用完该锁后将 lock 值设置为 1 即可。

自旋锁原理如图 3.7 所示。

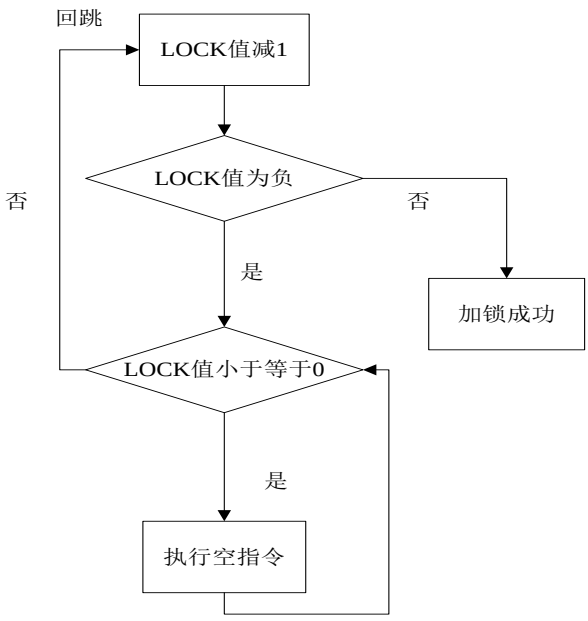

图 3.7　自旋锁原理

3.3.2　改进的互斥机制

虽然采用自旋锁机制很好的解决了多核之间的互斥问题，但是通过分析会发现在系统中简单的使用自旋锁会带来另外两个问题：

① 如果某一处理器核申请自旋锁，但此时自旋锁已被占用并且长时间不会被释放，那么该处理器将会一直在此锁上等待，降低了内核的利用率；

② 自旋锁保证了任务与任务之间、当前任务与其余核中断、当前中断与其余核中断的互斥，并未实现当前任务与本核中断以及当前中断与本核中断的互斥。

针对第一个问题有两种解决办法。一是系统设置两个变量 count 和 threshold，count 记录处理器判断 LOCK 值操作次数，threshold 表示系统设置的阈值。处理器每查询一次自旋锁的值，count 就要相应加 1，并且和 threshold 作比较，如果其值大于系统设置阈值，那么处理器核放弃对自旋锁的等待，对应进程由空等态进入睡眠态，否则将继续循环，判断自旋锁是否被占用。

另外一种办法是每个处理器核为申请自旋锁的任务设定锁占用时间。若在该时间段内任务仍未完成对临界资源的访问，就要自动放弃对锁的占用，执行解锁过程。对于该方法，如何设计锁占用时间是设计的关键。若采取为所有任务设置相同时间，则实现简单，系统开销小，但是由于其未考虑任务个性以及任务优先级高低，该方法的缺陷也是

显而易见的。首先每个任务对共享资源使用情况不同，如果将时间设置过低，则对于要求使用时间长的任务来说，不得不多次放弃对其的访问，频繁交替造成系统资源的浪费；如果设置过高，那么对共享资源访问时间较短的任务来说，又不得不长时间空等待，内核利用率也会降低。其次每个任务的优先级不同，则占有资源的时间分配也应不同，这样才能保证高优先级的任务优先执行。对此方法的改进是采用随机时间设置法，即所有任务占有自旋锁的时间不是固定的，可根据任务自身特性以及优先级大小进行设置，这样既避免了某个任务长时间占有，其余任务一直处于饥饿状态的想象，又使得高优先级的任务更易于获得资源的占有权。当然改进后的方法增加了系统开销，如果使用不当反而会降低系统的效率。

选择何种方法要根据具体的系统应用类型而定，由于本文对共享内存的访问是基于消息包的，通信量小，操作简单，因此选用第一种方法即可满足系统要求。

针对第二个问题，由于自旋锁未实现当前任务与本核中断以及当前中断与本核中断的互斥，而单核系统提供的关中断机制正是为解决该问题而提出的，因此系统可以采用自旋锁与关中断相结合的策略。这样系统提供中断自旋锁和任务自旋锁两种机制来保证多核下任务对资源的互斥访问。

RTEMS 多核下的同步互斥机制如图 3.8 所示。

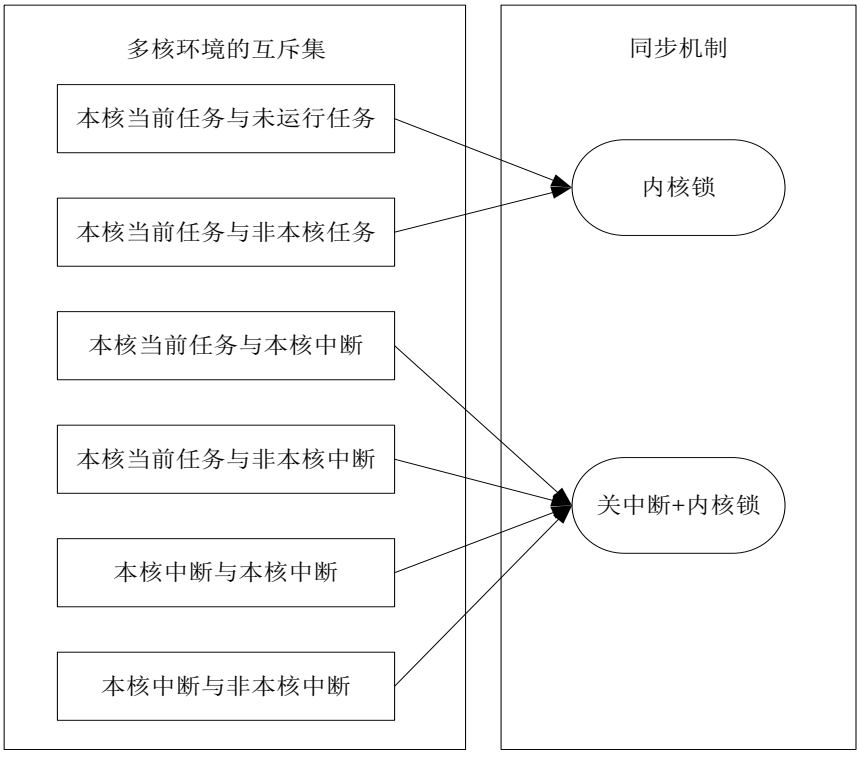

图 3.8　RTEMS 多核下的同步互斥机制

3.3.3　信号量机制实现

（1）中断自旋锁的实现

1）中断自旋锁的数据结构

要实现一个中断自旋锁，需要记录自旋锁归属字段，表示自旋锁是否在某个处理器核上，并且给出是哪一个处理器核取得了该自旋锁，以判断该锁是否正在被使用。另外处理器核在读锁过程中根据任务中断自旋锁的状态标记处理器核对该锁的获取情况。由于在读锁初始强制关闭中断，因此需要一个字段记录中断级别，在解锁或放弃等待时将中断打开。根据以上解析，本文定义的中断自旋锁的数据结构具体代码实现如下。

```
structure tSpinLockIsr
{
    int CpuOwner;
    tCpu cpu[CPU_COUNT];
    int key;
    TCB *pTcb;
}
structure tCpu
{
    int flag;
}
```

包含的字段含义如下。

CpuOwner：自旋锁归属字段；

CPU_COUNT：系统中包含的可用处理器核的总数；

cpu[CPU_COUNT]：自旋锁针对每一个处理器核的独立字段；

cpu.flag：自旋锁在每个处理器核下的标识，表示该处理器对于指定自旋锁的使用情况。可设置为 SPIN_LOCK_INTERESTED（正在准备取锁），SPIN_LOCK_EMPTY（没有使用任何自旋锁），SPIN_LOCK_WAIT（等待其它处理器核正在使用的自旋锁）。

key：该自旋锁被取得后运行的中断级别。

pTcb：获得该锁的任务的任务控制块。

2）中断自旋锁的初始化

在定义了中断自旋锁的数据结构后，在使用中断自旋锁之前要对其进行初始化。在对中断自旋锁初始化时，先将中断自旋锁的归属字段设置为SPIN_LOCK_NOBODY (-1)，表示该自旋锁当前不被任何处理器核占有。再将属于每个处理器核的 flag 字段设置为 SPIN_LOCK_EMPTY（0），表示当前该处理器核没有使用任何的自旋锁。具体代码实现如下。

```
void spinLockIsrInit(tSpinLockIsr *pLock)
{
    int i;
    pLock->CpuOwner = SPIN_LOCK_NOBODY;
    for (i = 0; i < CPU_COUNT; i++)
     {
        pLock ->cpu[i].flag = SPIN_LOCK_EMPTY;
     }
}
```

3）中断自旋锁的加锁

在被初始化以后，中断自旋锁就可以被使用了。加锁过程要先判断该锁是否已经被其它的任务或中断取得，如果没有则取锁，改变锁当前状态。若该锁已被其它任务或中断取得，则任务循环等待，直到锁被释放或到达一定的循环次数。在这个过程中必须保证对锁中各个字段的修改操作是原子的。加锁过程的实现流程如图 3.9 所示。

4）中断自旋锁的解锁

在中断自旋锁的解锁过程中，应该将在获得该锁时关闭的中断打开。中断自旋锁解锁流程如图 3.10 所示。

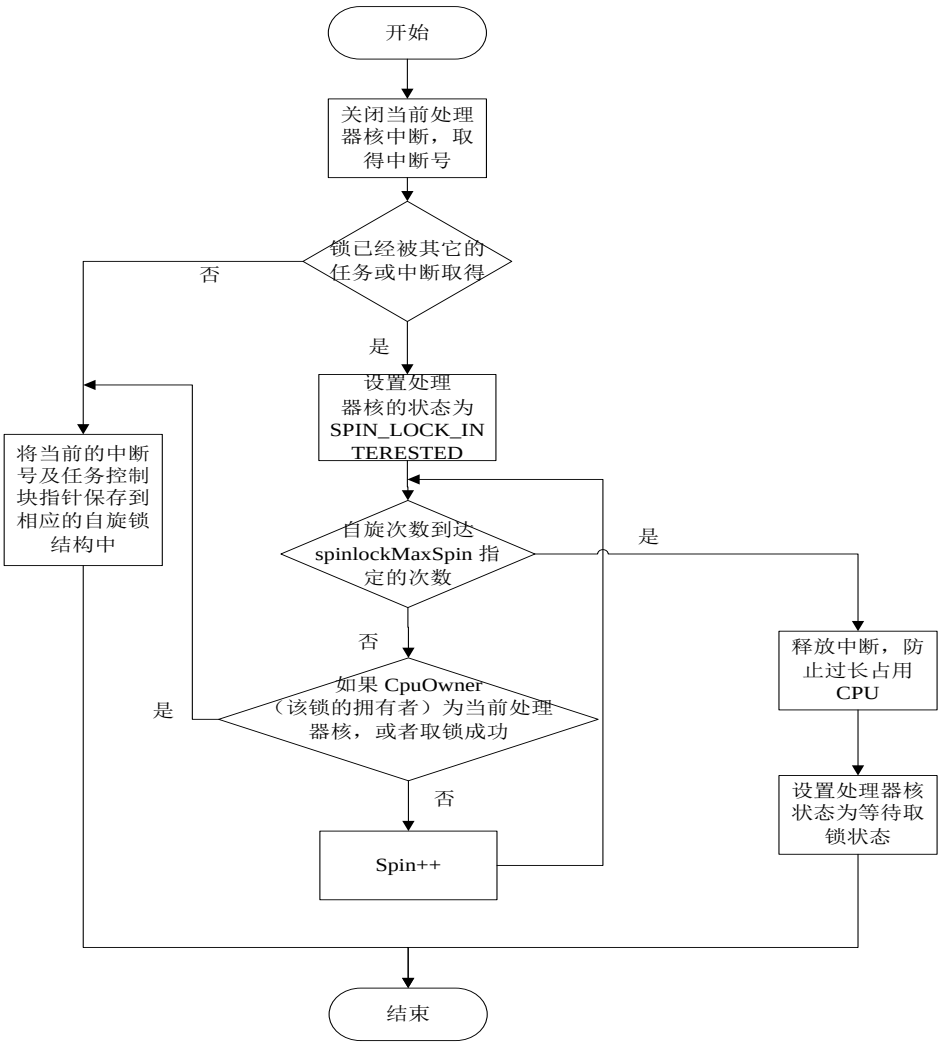

图 3.9　中断自旋锁获取流程

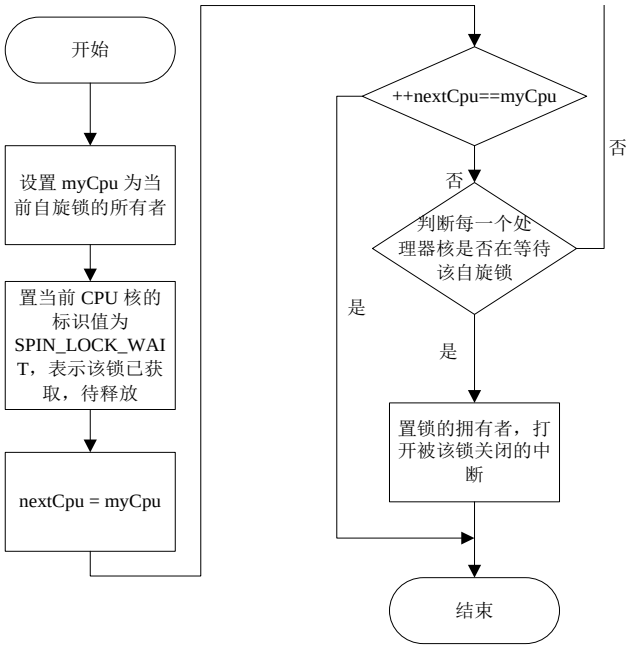

图 3.10　中断自旋锁释放流程

（2）任务自旋锁的实现

1）任务自旋锁的数据结构

为了实现一个任务自旋锁，需要定义字段记录任务对锁的拥有权的标识、下一个可获取自旋锁的任务的标识、已获取到自旋锁的标识以及相应的处理器核，因此任务自旋锁的数据结构具体代码实现如下：

```
structure tSpinLocktask
{
    myTicket;
    int nextTicket;
    int ticketInService;
    int cpuIndex;
}
```

包含的字段含义如下。

myTicket：调用该函数的任务对锁的拥有权的标识。

nextTicket：下一个可获取自旋锁的任务的标识。

ticketInService：已获取到自旋锁的标识。

cpuIndex：相应的处理器核。

2）任务自旋锁的初始化

对任务自旋锁的初始化，只要设置自旋锁中变量值为初始值即可。其中nextTicket 为 0；ticketInService 为 0；cpuIndex 为-1。具体代码实现如下。

```
void spinLockTaskInit(tSpinlocktask *pLock)
{
    pLock->nextTicket = 0;
    pLock->ticketInService = 0;
    pLock->cpuIndex = -1;
}
```

3）任务自旋锁的加锁

任务自旋锁在初始化后就可以被使用了，使用前要先获取任务自旋锁。任务自旋锁的加锁之前要等待该锁空闲，即使用该锁的任务将该锁解锁。实现流程如图3.11 所示。

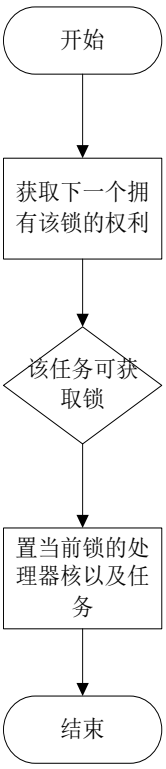

图 3.11 任务自旋锁加锁流程

4）任务自旋锁解锁

任务自旋锁的解锁流程如图 3.12 所示。

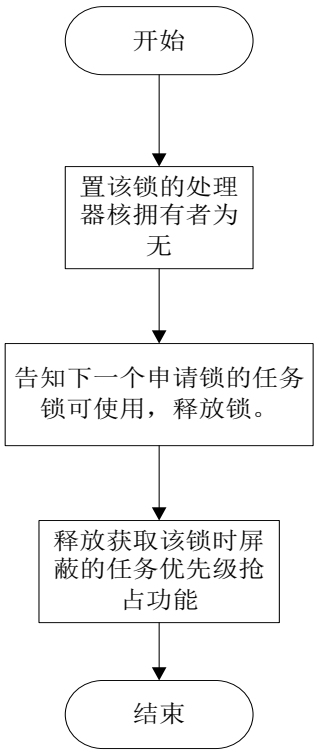

图 3.12 任务自旋锁解锁流程

3.4 本章小结

详细介绍了并行系统结构、存储结构模型、通信机制和 MPSL 层三个模块：MPCI 接口层，共享内存实现层，硬件相关实现层。探讨了 RTEMS 支持 SMP 设计过程中采用的节点间通信机制。并在原有自旋锁机制的基础上提出了一种任务自旋锁与中断自旋锁相结合的机制，解决了单核情况下的同步互斥机制在多核系统中不能保持原有的语义的问题。

第 4 章　RTEMS 多核任务调度机制

在实时系统中任务分配问题起着重要作用，直接决定着系统的实时性、高效性[46]。合理的分配任务将在保证系统实时性的前提下最大程度地提升系统的整体性能。RTEMS 内核对多核任务调度采用平均分配策略。由于任务具有不确定性，并且系统资源随着任务执行会不断发生变化，如果简单的采用平均策略，系统并行性不能得到很好的利用，严重影响系统整体性能。动态调度算法能够根据系统的状态信息动态决定任务分配，灵活性较高，更好的保证了系统负载平衡，提高了系统并行性。

4.1　任务分配问题

任务分配是指将任务分配到系统最合适的位置运行[47]，任务分配算法解决任务分配问题，一个好的任务分配算法可以缩小并行程序的执行时间。在设计任务分配算法之前需要确定任务就绪队列模型的选择。

在多核系统中，任务就绪队列模型主要分为两种：全局队列模型和局部队列模型[48]。

全局队列模型是指系统中所有节点共享一个全局任务就绪队列，全局任务就绪队列中记录系统中所有处于就绪状态的任务，由主控节点将就绪队列中任务调度到系统中可用节点上执行，一个任务在其生命周期中并不一定是在同一个节点上完成的，可能在不同时间运行在不同的节点上。相反，局部就绪队列模型是指每个节点都维护一个任务就绪队列，每个任务自始至终都在同一个节点上执行，系统中各节点的任务调度互不干扰。

全局队列模型由于所有节点共享一个就绪队列，因此实现了系统的负载均衡。但是，由于全局就绪队列是共享的，因此必须采取同步互斥机制实现多核的正确访问，这样多个节点不能同时进行任务调度，严重影响了系统的并行性。另外 cache 命中率也会受影响，因为系统要实现负载均衡，所以会出现同个任务在不同核之间迁移的情况，这样会增加 cache 不命中率，使得系统性能降低。

而局部就绪任务队列模型相对全局队列模型来说，更好的利用了系统资源，首先系统中各个节点都维护一个就绪任务队列，这样可以同时进行任务调度，提高了系统的并行性。其次，任务在其生命周期中很可能在同一个节点上执行，因此 cache 命中率会有很大增加，提高系统性能。

基于以上对比，选择将局部就绪任务队列模型作为 RTEMS 多核化任务分配模型。

如果采用局部就绪任务队列模型，则必须考虑负载平衡实现问题。与全局队列模型不同，局部队列模型的负载平衡实现需要更多的机制提供支持。

多核处理器的出现很大程度上提升了处理器性能。为了充分发挥多核处理器高并行性的优势，面临的一个基本问题是如何将工作均衡的分配到各个节点。对于多核系统，由于任务到达时间不确定性、任务在运行过程中产生的不确定变化，使得多核系统中的一些节点任务较多，负载过重，而另一些节点却是空闲的。

负载均衡是把任务根据所有节点的负载情况按照一定的任务调度策略和调度算法进行分配，以实现各节点统一开始工作，统一结束，不会因为部分节点任务轻处于空闲状态，而另一部分节点任务负载过重而一直处于忙碌状态[49]。

下面部分主要针对采取局部队列模型进行任务分配问题的研究，并设计一种高效的任务调度算法，以满足系统性能需求。

4.2 任务调度算法分类

任务调度算法根据任务调度时机可以分为静态任务调度算法和动态任务调度算法两种[50]。

（1）静态任务调度算法

在静态任务调度中，所有任务在编译阶段根据任务的通信开销、计算开销、任务之间的依赖关系等属性来确定各个节点的运行任务以及各任务的执行顺序，一旦任务确定执行处理器后，在运行过程中将不再调整，一直运行在该处理器内核上。

静态任务调度相对简单，运行时调度器开销较小，系统具有很好的可预测性。但是由于静态任务调度任务对应的执行处理器一经确认就不予更改，未考虑任务到达时间、任务在运行过程中产生的不确定以及任务运行过程中节点的负载状况，因此这种算法缺少灵活性，在某些情况下严重影响系统的整体性能。

（2）动态任务调度算法

动态任务调度是指在并行任务运行过程中，根据任务的运行情况以及各节点的负载状况决定每个任务的执行处理器节点。该算法具有更强的执行能力，能够根据系统的状态信息动态决定任务分配。

动态任务调度相对静态任务调度灵活性较高，根据任务的动态运行情况以及各处理器的负载情况进行调整，更好的保证了系统负载平衡，提高系统并行性。但是动态调度算法不可避免的加大了额外开销，如信息搜集、分析、存储以及任务的移动，因此动态任务调度算法应该尽可能的简单、高效，这样才能抵消这些额外开销，充分发挥动态任务调度算法的优势。

RTEMS 对多核处理器的支持如果采用动态调度策略，不但可以弥补平均分配策略带

来的不足，而且可以充分利用共享内存获取各节点的负载状态，进而对任务进行均衡分配。下面首先对现有动态任务调度策略进行分析，在此基础上提出一种基于系统整体效能和系统整体负载均衡的动态任务调度算法。

4.3　现有集中式动态任务调度策略

RandCentLB 调度策略，是一个随机调度策略，即当多核系统出现负载不均衡情况时随机选择一个处理器，实现任务重分配。该策略实现简单，当任务数量很大，并且任务之间通信非常小时，调度结果比较满意。但是该策略有非常大的局限和不确定性。

GreedyLB 调度策略，首先找到系统中负载最小的处理器，然后将当前未分配任务的最大负载分配给该处理器。该策略未考虑通信开销，对于任务之间通信频繁、通信代价大的系统并不一定能提高效率。

GreedyComm 调度策略在 GreedyLB 的基础上进行改进，在选择目标机的时候不但考虑节点当前负载情况，而且加进通信开销因素，一定程度上弥补 GreedyLB 调度策略的不足。

RefineLB 调度策略，是在系统运行过程中不断调整系统负载分布，系统每个节点都设有一个阈值，当存在任务负载超过该阈值的节点时，启动系统调度模块，进行任务重分配。该策略属于任务发起者调动策略。由于该策略只考虑了系统中重载节点，算法实现简单，并且通信开销较少。

RefineCommLB 调度策略，类似于 RefineLB，但是该策略在选择任务接收者时考虑到了任务之间的通信开销，以寻求一种更有益的选择[51]。

为了充分发挥多核处理器并行性能，必须找到一个合理的任务分配策略。任务策略的选择首先要考虑具体系统结构和任务类型，再在此基础上寻求最优任务调度策略。任务调度问题已被证明是 NP 完全问题，所以在大多数情况下，只能取得问题的近似最优解。并且一种具体的调度算法是在很多限制和先设条件下提出的，只能适应某些特定的情况和环境。

系统任务重新分配会带来两种开销[52]：任务在节点上的执行时间和节点之间的通信开销。为了提高系统性能，一个优秀的任务调度策略需要满足两个目标：减少节点的执行时间；减少节点之间的通信开销。这是两个相互冲突的目标，一方面要想减少节点的执行时间，达到缩短整个任务的执行时间目的，必须将任务均衡分配到各个节点上，但这势必会加大节点之间的通信开销。另一方面，如果减少节点之间的通信开销，则任务要尽量分配到一个节点中运行，这又会导致系统负载严重不均衡，影响任务执行结束时

间。

因此任务调度策略需要在这两个目标中达到一个平衡。基于对以上考虑提出一种更加全面调度策略，该策略不但考虑了系统各节点的负载情况和节点之间的通信开销，而且考虑了系统中每个任务优先级和系统整体负载状况，最大限度的利用系统的并行性，提高系统整体性能。

4.4 动态任务调度模型

动态任务调度模型可以形象描述多核系统任务调度过程，动态任务调度模型包括任务集、负载估计、调度策略和任务映射。对动态任务调度模型的求解过程就是在多核处理器系统环境下，寻求一个优秀的调度策略[53]，该策略以任务集和节点负载情况作为输入，将任务到节点的映射作为输出。动态任务调度模型如图 4.1 所示。

图 4.1　动态任务调度模型

4.4.1 任务集

多核处理器系统中的任务集可表示为一个四元组。

定义 1（任务集）：$T=(V,E,RT,CT)$，其中：

$V=\{V_1,V_2,\ldots,V_n\}$，任务集合，每个顶点表示一个任务，V_i 表示任务 i；

$E=\{E_1,E_2,\ldots,E_m\}$，任务关系集合，任务顶点和边的关系可表示为 $(V_i,V_j)\in E$，即任务 i 是任务 j 的前驱，任务 j 是任务 i 的后继，只有任务 i 完成之后，任务 j 才能开始；

$RT=\{RT(1),RT(2),...,RT(\mathrm{n})\}$，任务运行时间集合，$RT(\mathrm{i})$ 表示任务 i 运行时间；

$CT=\{CT_{\mathrm{i\text{-}j}}\}$，通信时间集合，其中 $(V_i,V_j)\in E$，$CT_{\mathrm{i\text{-}j}}$ 越大表示任务 i 与 j 之间的通信代价越大。$CT_{\mathrm{i\text{-}j}}$=0 表示两个任务之间无通信。

根据定义 1 描述，可将系统任务集表示为一个加权的有向无环路图 DAG。如图 4.2 所示。

该图中顶点表示系统所有任务，本例中共包括三个任务，其中每个任务又可以进行分解，T1，T2，T3，T4 属于任务 1，T5，T6，T7，T8 属于任务 2，T9，T10，T11，T12 属于任务 3。图中有向边表示任务之间的序列关系，边上权值表示任务之间的通信代价以及通信频率。由于属于同一任务的节点之间的通信频繁，开销较大，因此边上的

权值定义为无穷大。

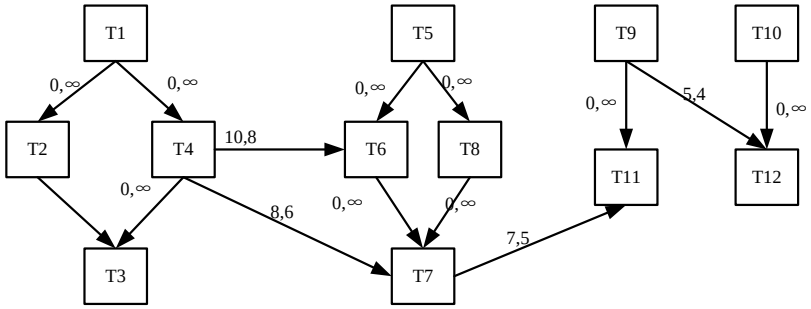

图 4.2　DAG 图例

4.4.2　负载估计

多核系统中的负载估计可表示为一个三元组。

定义 2（负载估计）：$L=(LP,LT,LS)$，其中；

$LP=\{LP_1,\ LP_2\ldots LP_p,\ t\}$，$p$ 表示多核系统中节点数目，LP_i=$p_i q_i$ 表示 i 节点在 t 时刻的负载。由于任务和系统类型差异，找到一个合适的负载向量来衡量各个节点的执行状态是非常困难的。目前常采用的负载参数可分为以下三类。

（1） 将资源利用率作为负载指标

资源利用率是衡量系统性能的重要部分[54]，通常选择 CPU、内存 I/O 等资源利用率作为负载指标。

由于 I/O 资源利用率不容易获取，影响准确度，因此一般选择 CPU 和内存利用率两方面作为负载指标。

CPU 利用率 p 由系统使用的时钟数比时钟总数得到。内存利用率 q 考虑绝对的剩余空间。

（2） 将任务队列长度作为负载指标

使用任务队列长度作为负载指标与利用资源利用率相比带有预测性质，资源利用率只表示节点在某个时刻的状态。但是由于各任务大小不同，单纯从个数方面衡量节点负载情况，会造成一定的误差。

（3） 使用综合策略作为负载指标

综合策略指同时考虑多种负载指标，通过对每项指标加权计算最后得出节点负载的综合值。

由于负载均衡的重要目的之一是充分利用并行系统的资源，因此资源的利用率是衡量一个节点执行状态的重要指标。并且有研究表明，如果衡量执行节点负载状态的指标

选择的非常复杂，反而会影响算法的效率，并不能取得很好的效果。因此在该调度模型中采用资源利用率作为负载指标，定义 LP_i=p$_i q_i$ 。

$LT=\{LT_1,LT_2,\ldots,LT_{\mathrm{p}}\}$，其中元素表示系统每个节点的负载阈值，p 表示系统中节点数目。由于系统采用同构多核处理器体系结构，因此各节点的负载阈值相等。假设各节点的处理能力为正整数 C，则节点的阈值定义为 $LT=\alpha C,\ (0<\alpha<1)$，节点的阈值步长定义为 t，节点的实际负载为 LP 。

C 值的大小直接反应节点的整体处理能力，这与节点的配置如 CPU 频率、内存读取速度、cache 读取速率、cache 层次、I/O 速率等因素有关。由于本文系统采用同构多核处理器，因此每个节点的处理能力 C 值大小相同。

$LS=\{NL,SL,HL\}$，节点负载状态集，其中 NL 表示空载状态，SL 表示适载状态，HL 表示超载状态。

节点的负载状态可由负载阈值确定，如果

$LP<LT$-t， 说明节点处于空载状态，此时节点没有任务执行或者执行任务量很少，可以作为任务接受者，接受来自其余节点的任务；

$LP>LT$+t，说明节点处于超载状态，此时节点中的任务量超过了自己的执行能力，需要将部分任务转移到空载状态的节点进行执行，否则将会导致任务执行时间加大，影响系统的整体性能；

$LP\in[LT-t,LT+t]$，说明节点处于适载状态。此时节点中的任务正好符合自己的处理能力，既不需要进行任务转移，也不会接受其余节点的任务。

节点负载状态判断如图 4.3 所示。

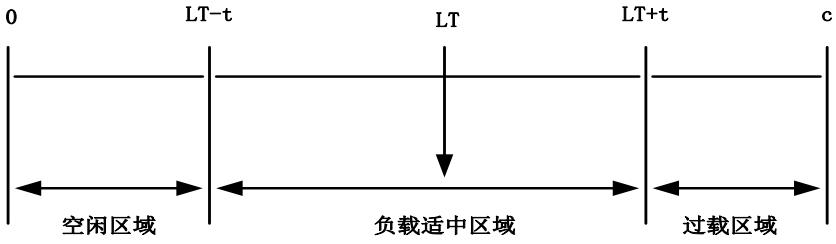

图 4.3　阈值和步长的基本原理

4.4.3　调度策略

调度策略可表示为一个四元组，四个因子组成系统调度过程中的整个生命周期。

定义 3（系统调度策略）： $SA=(\mathrm{TR,SE,PL,IN})$，其中：

（1）TR：转移策略

转移策略是指在任务调度的生命周期中，决定何时进行任务转移，判断是否有节点处于合适的状态参与转移工作。本文采用阈值策略实现任务转移。

每个节点设有两个阈值 y_1 和 y_2，若有任务到来，某一节点完成分配任务使其负载小于 y_1 时，则确定该节点为接收者；同理，若节点完成分配任务使其负载大于 y_2 时，则确定该节点为重载节点，并确认为发送者。阈值设置务必要谨慎，如果阈值设置过高，则会造成节点负载已经非常严重，但仍表现空闲的现象；如果阈值设置过低，任务迁移动作过于频繁，造成系统资源浪费，严重影响系统性能。为了实现简单本文只为每个节点设置一个阈值，根据定义 2，如果有节点进入过载区域即可进行任务转移工作。

（2）SE：选择策略

选择策略工作在转移策略工作之后，在确定发送者节点之后决定该节点之上的待转移任务。选择待转移任务的原则是额外开销尽量少，转移任务足够大，这样才能保证取得“利益”大于转移开销。

（3）PL：定位策略

在任务调度周期过程中，定位策略决定待转移任务的接收者，该过程必须考虑各节点的负载状态，任务转移带来的开销，以及转移后带来的通信开销，一个好的定位策略可以很大程度上优化系统的整体性能。

（4）IN：信息策略

系统根据各节点的信息决定何时进行任务调度，对哪些节点进行任务调度，需要迁移哪些任务，信息策略解决如何收集信息、信息存储位置、收集信息类型等问题。

由于 RTEMS 支持的多核系统结构采用共享内存的组织方式，因此信息策略采用在共享内存中建立资源池来存储各节点的系统信息，并且由 0 节点实现信息的统计、分析，执行任务的动态调度。

调度架构的四个方面包含了系统调度的各个生命周期，调度算法将围绕这四个方面进行分析，并在前人研究的基础上，提出一种两级动态任务调度（Two dynamic task scheduling algorithm-TDTS）算法。该算法的核心为在保证系统整体性能最大的前提下，使得系统整体负载达到最大均衡。

4.5　资源统计模块

0 节点资源统计模块完成了系统各节点资源信息监测和收集，供任务调度模块使用。该模块在 RTEMS 系统启动之后立即执行，并在系统运行的过程中循环执行，接收其余节点发送来的信息，并将结果以统一格式存储在 0 节点开辟的资源池空间中。任务调度

模块从资源池中获取全局信息，进行任务调度。

资源统计模块和任务调度模块分属于不同功能域，之间需要进行信息通讯以协助完成系统调度。RTEMS 系统首先读取共享内存中系统信息，得到系统中节点数目，然后根据系统资源配置文件中设置调查信息类型，为每一个节点分配固定大小内存，存储相应节点的系统信息。开辟的这段共享内存空间即为系统调度获取信息的资源池。

资源池建立之后，剩下的工作即解决 0 节点如何接收系统所有节点发过来的信息。

资源信息统计策略采取异步获取策略，即节点资源发生改变后，立即将信息发送到 0 节点，0 节点接收信息后对资源池进行更新。由于系统节点资源随着任务进行时刻都会发生变化，大多数情况下，节点资源波动很小，不会影响系统全局任务调度。如果每次变动都要求向 0 节点发送数据，更新资源信息，这样频繁发送反而会浪费系统资源，影响系统效率。可以在调度策略中采取为每个节点定义范围的方法解决这个问题，如果系统资源变动幅度超过这个范围再向系统发送消息。这样避免了无用信息的多次传递。

资源统计模块执行过程如图 4.4 所示。

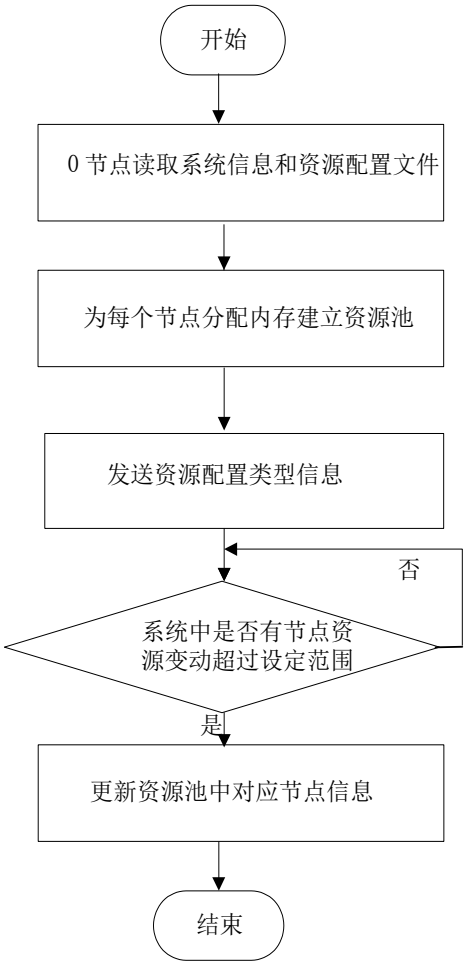

图 4.4　资源统计模块执行过程

4.6　系统调度策略实现

为了结合静态任务调度的高效性，系统任务调度主要分为两阶段：任务的静态初始分配和任务运行过程中系统动态调度。

4.6.1　任务初始分配

任务初始分配在系统调度过程中非常重要，初始分配的准确性直接影响后续系统的负载平衡[55]。任务分配应遵循的原则为：将通信量大、通信频繁的任务分配到一个节点上，以减少通信开销；应使得系统各节点负载均衡。

基于以上原则，任务的初始分配采用基于数据划分和任务复制的并行调度算法[56]。通过数据划分得到任务关系图，基于复制将任务分配到对应的处理器核上。

（1）数据划分

为了使得系统在最短的时间内完成所有任务，必须将任务均衡的分配到所有节点中，并且节点之间的通信应该降至最低，以节省系统通信带来的开销。基于以上来两点，数据划分应按照如下原则：在数据划分时尽量将同一工作的任务分配到同一个处理器核上执行；通信开销大的任务应分配到同一个处理器核上执行。以图 4.2 任务集合为例，按照该原则将任务集进行初始数据划分之后如图 4.5 所示。

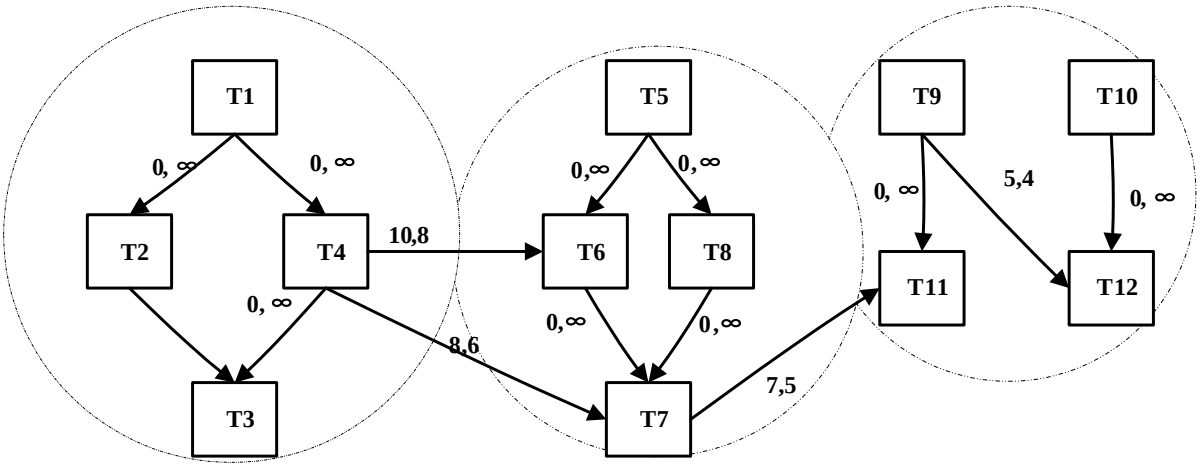

图 4.5　任务划分初始分配图

图 4.5 中虚框表示该框中的所有任务均被分配到同一节点。

初始任务分配必须保证：分配节点数目必须小于系统中所有节点数目；每个节点上的负载量必须小于 LT+t，即不能超过节点自身的处理能力。

按照以上原则进行数据划分，势必造成某些节点存在时间空闲块的情况。这样违背了系统负载平衡原则，未能充分利用多核优势，严重影响系统并行性。任务复制算法将

很好解决这个问题，该算法通过空间换时间，将通信频繁、通信代价大的任务进行复制，提高处理器利用率，进而提高系统的并行性。

（2）任务复制

降低任务之间通信开销的一个办法是采取任务复制方法。复制方法的关键问题在于找到复制任务，既可以选择复制所有的公共任务，也可以选择部分公共任务进行复制。由于采用任务复制方法需要考虑多方面因素，并且需要额外的空间进行存储，因此与其它算法相比有更高的时间和空间复杂度。但任务复制方法可以充分利用系统的并行性，具有更高的调度性能，所以在并行与分布式任务调度策略中仍得到了广泛的应用。

本文采用了 Fork-Join 任务图这种基本模型结构中的任务复制思想[57]。即，复制公共的父任务，并且把子任务适当地组合起来；让公共的子任务与具有最大计算和通讯时间的父结点处于同一处理核，并在该核上适当的吸纳其他父结点。

复制任务的重点是找到系统中的关键任务T_s。其中 s 为满足以下条件的最小下标值，

$$\sum_{i\in N}^{S}\left(RT_i+CT_{i-j}\right)\geq \max\left(RT,CT\right)$$

对图 4.5 中任务集进行初始数据划分之后进行复制算法，映射后的结果如图 4.6 所示。

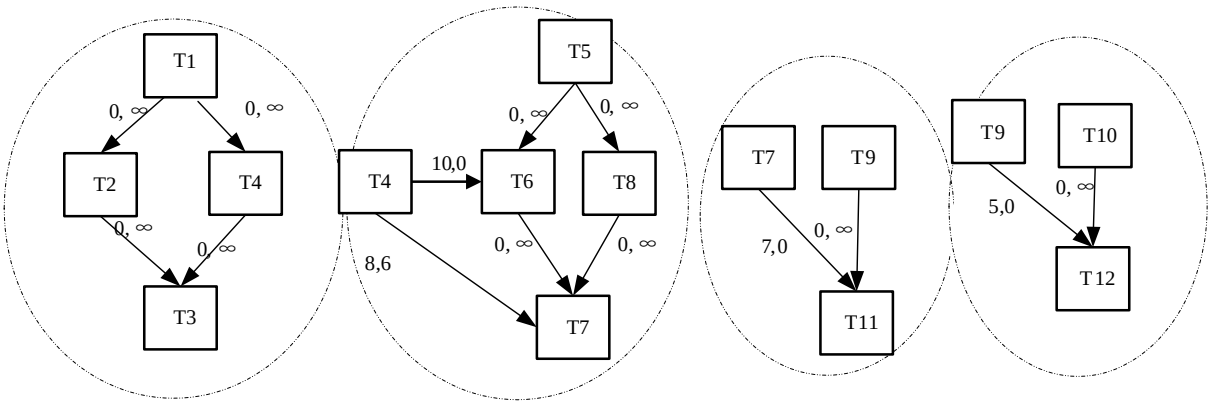

图 4.6　任务复制后的映射图

4.6.2　动态任务调度算法

动态任务调度算法需解决的主要问题是确定任务的发送者与接收者[58]。

系统在任务执行过程中如果发现有节点没有足够能力处理某些任务，即节点处于过载状态，则进行任务调度，将该节点设为发送点，在其余节点中寻找空闲节点。经过对比，主控节点选择使得系统整体效能变优节点集合 N。然后在该节点集合中继续选择，计算 N 中各个节点接收任务后系统整体负载情况，从中选择最优节点，使之成为任务的

接收者。

定义 4（节点效能）：$U=\{U_1,U_2,...,U_p\}$，U_i 表示系统中节点 i 执行任务 j 的效能，p 表示系统中节点数目。

节点执行效能定义为收益减去代价，其中收益包括任务的优先级、任务的 level 值、节点的执行能力，其中 level 值代表入口任务到该任务的路径长度，衡量系统中任务之间的先后关系，值越大表明该任务执行时间越晚，定义入口任务 level 值为 0。另外因为系统中节点执行能力一致，因此可以省略对该因素的考虑；代价包括与其余节点的通信开销和任务的执行时间。设任务 j 的优先级为 $\mathrm{Priority}(v_j)$，任务的 level 值为 $Level(v_j)$，结点 i 完成任务 j 的时间开销和通信开销分别为 $Time_i(v_j)$ 和 $Communicate_i(v_j)$，则结点 i 完成任务 j 的效能为：

$$U_i(j)=\alpha_1\mathrm{Priority}(v_j)+\alpha_2 Level(v_j)-\alpha_3 Time_i(v_j)-\alpha_4 Communicate_i(v_j)$$

定义 5（系统整体效能）：$U=\sum_{i=0}^{p}U_i$，U_i 表示 i 节点效能，p 表示系统中的节点数目。

定义 6（系统负载情况）：

$$V=\sum_{i\in N}(\mathrm{r}_i-\mathrm{r}_0)^2 \qquad (4\text{-}1)$$

式中：　$r_i=\dfrac{L_i}{C_i}$，　$\mathrm{r}_0=\dfrac{\sum_{i\in N}LP_i}{\sum_{i\in N}C_i}$

其中 r_i 表示节点的负载率，r_0 表示系统整体的负载率。很明显，节点负载率表示单个节点负载情况，r_i 越大表明节点负载越重，越小表明节点负载越轻；与此对应，r_0 表示系统整体负载情况，r_0 的大小直接反应了系统整体负载水平，越大系统整体负载越重，越小系统整体负载越轻。然而系统负载平衡的真正意义是指所有节点根据自己处理能力“均衡”分担系统负载，单独变量 r_i 和 r_0 都不能准确反应系统平衡状况，公式（4-1）中引入方差概念，以反应各节点负载率的离散程度，作为系统整体负载指标。由公式（4-1）我们可以知道，V 与系统负载平衡成反比，V 值小表明系统整体平衡状况良好，V 值大表明系统整体负载不均衡。

算法过程描述如下：

Step1：主控节点统计资源池中各节点负载信息，当系统中存在节点 i，该节点负载 $LP>LT+\mathrm{t}$ 时，说明系统有过载节点，定义该节点为任务发送者；

Step2：主控节点根据资源池中负载信息，选择处于空载状态的节点，形成集合 N，该集合中所有节点负载 $LP \prec LT$-t；

Step3：计算节点 i 除去任务 T_k 之后对自己效能影响，即

$$U_i^- (T_k) = U_i (S_i \setminus \{T_k\}) \tag{4-2}$$

Step4：计算节点集合 N 中，每个节点接收任务 T_k 后对自己效能的影响，即

$$U_j^+ (T_k)() = U_j (S_i \cup \{T_k\}) \tag{4-3}$$

Step5：计算各个节点接收任务后系统整体性能情况，即

$$U_{i,j} (T_k) = U_i^- (T_k) + U_j^+ (T_k) \tag{4-4}$$

Step6：如果不存在节点使得任务转移后系统整体性能有所改善，则表明接收节点不存在，算法结束，否则跳转到 Step7；

Step7：选择节点集合 N 中使得 $U_{i,j}(T_k) \succ \delta$（$\delta$ 为可设置的阈值）的节点构成集合 G，剩余节点则标注为不可接收节点；

Step8：针对节点集合 G，计算每个节点接收任务后的系统负载平衡水平，选择使得系统整体负载情况最佳的节点，即

$$Min \sum_{i \in N} \left(\frac{WS_i * Q + LP_i}{C_i} - r_0 \right)^2 \tag{4-5}$$

式中：$\sum WS_i = 1$，$WS_i * Q + LP_i \le C_i$，$Q = POSL$，$WS_i = 0,1$

Step9：动态任务调度算法至此结束。

4.7 本章小结

针对 RTEMS 内核在多核任务调度实现上采用统一分配带来不足，提出一种新的任务调度模型，该模型采用任务初始分配与动态任务调度相结合的策略，其中动态任务调度使用新提出的两级动态任务调度算法。该算法由主控节点 0 执行，根据资源池中采集信息，对系统任务进行重新分配。两级动态任务调度算法在有节点超载的情况下进行任务重映射，不但考虑任务迁移对系统整体效能影响，而且考虑不同选择下对系统整体负载情况影响。在降低任务调度长度前提下，更好的利用了系统并行性。

第 5 章　性能测试与分析

为了验证RTEMS多核操作系统的功能实现，本章设计了一些测试用例，通过在搭建的M5仿真平台上运行，验证多核下同步互斥机制与任务调度算法的可行性与高效性。测试用例采用TGFF工具生成，并对任务的运行结果从加速比和并行效率两个调度评价指标进行分析。

5.1　实验平台搭建

本章在M5多核仿真平台[59]上进行多核系统测试。M5模拟器是一个提供计算机系统级体系结构仿真的平台，采用面向对象的编程和使用机制，详细模拟了计算机系统的时序特性，并且可以模拟多处理器系统、多核系统等复杂结构。M5包含多种仿真模块，如CPU、cache、存储器、I/O器件等，这些模块包含了用于模块间通讯的通用接口。

M5提供了两种可相互替代的CPU模型：SimpleCPU，O3CPU，其具有相同的接口，允许在运行过程中根据不同的要求在各种CPU模型之间进行切换。M5的存储器系统主要包括两个部分，存储器器件和互连。存储器器件包括cache、物理存储器、I/O器件等，而互连包括总线和互连网络。I/O器件是存储器系统的一部分，用于模拟对于存储器系统中特定空间的I/O访问并向DMA 发出传输请求。

M5具有的多种特点为算法的实现提供了很大的帮助。M5提供的详细模拟计算机系统时序的仿真模型，为实现该算法提供了可能。通过使用M5提供的统计信息包可以很方便的获得测试程序运行的结果。

试验中多核系统主要参数设置如表5-1所示。

表 5-1　系统设置参数

CPU 频率（GHz）	总线频率（MHz）	存储器（Byte）	L1 Cache（Byte）	L2 Cache（Byte）
2.0	800	96M	128K	2M

5.2　实验设计

为了证明提出算法的高效性，本实验将多组同一规模的任务分别运行在单核、多核未采用动态调度算法、采用RefineCommLB调度策略的算法[60]和采用提出的TDTS算法的系统中，通过对实验结果对比，分析算法性能。

5.2.1　测试用例

为了得到公正、有效的测试结果，测试用例的设计非常重要。本文的测试用例描述文件由任务图自动生成工具 TGFF 产生。并调用测试用例生成模块将产生的任务描述文

件生成对应的并行程序，以此作为动态任务调度算法性能测试的输入。

本章测试用例满足以下条件：

（1）并行计算节点之间有偏序关系，需要互相通信共同完成任务；

（2）不同计算节点具有不同的计算量；

（3）计算节点之间的通信开销不同；

（4）同一计算节点在不同处理器节点上执行效率相同。

使用 TGFF 工具[61]生成 5 个任务集作为本章测试用例，分别命名为任务集 1-任务集 5，每个任务集设置参数情况如下：

（1）计算任务个数范围{30，38，48，60，74}；

（2）通讯边数{47，52，62，77，97}；

（3）任务平均通讯时间与计算时间比值{0.1，0.5，2.0，5.0，10.0}。

任务集设置参数情况如表 5-2 所示：

表 5-2　任务集设置参数

任务集名称	计算任务个数	通讯边数	任务平均通讯时间与计算时间比值
任务集 1	30	47	0.1
任务集 2	38	60	0.5
任务集 3	48	75	2.0
任务集 4	60	95	5.0
任务集 5	74	120	10.0

5.2.2　实验结果

利用上述测试方案得到实验结果数据如表 5-3 所示。

表 5-3　实验结果数据　　s

	任务集 1	任务集 2	任务集 3	任务集 4	任务集 5
单核	403	550	760	1060	1560
未采用动态调度算法	182	223	302	415	604
RefineCommLB 调度策略算法	171	209	273	378	549
TDTS 算法	188	231	271	368	534

实验结果对应的统计图如图 5.1 所示。

实验结果统计图表明，当任务规模相同时，单核系统完成任务所用时间明显大于多核系统。多核系统中采用动态分配策略情况的结果优于未采用动态分配略。而随着任务规模的不断扩大，与采用 RefineCommLB 调度策略算法相比，新提出的 TDTS 算法缩短了任务的执行时间。

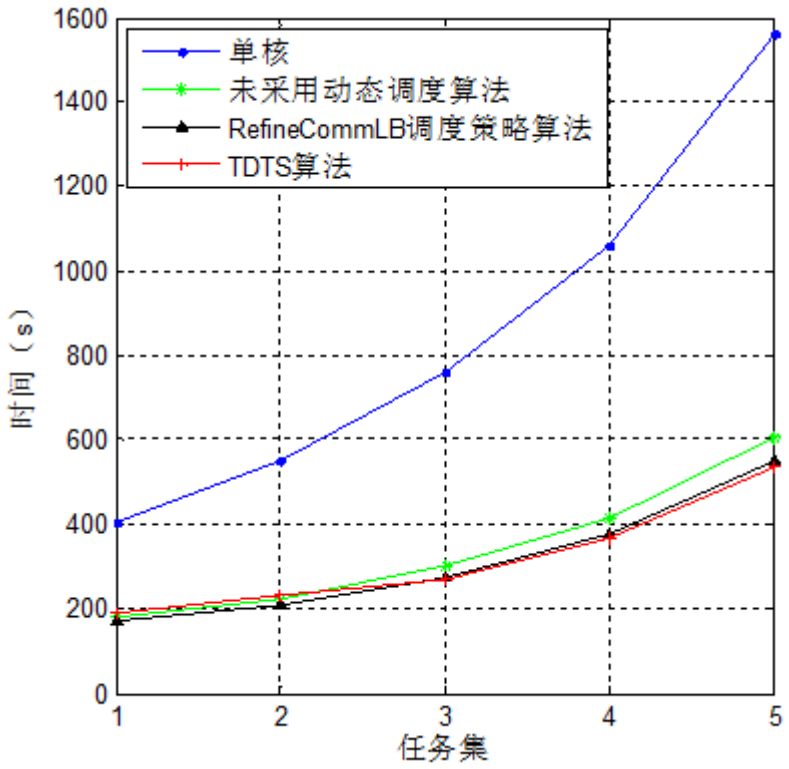

图 5.1　实验结果统计图

5.2.3　性能分析

动态任务调度策略的优劣需要评价指标进行衡量，由于任务调度的主要目标是通过充分利用系统资源以减少任务执行时间，因此本文选择加速比和负载平衡效率作为策略性能评价指标。分别定义如下：

S：加速比，定义为在单个处理器上完成任务所用时间T_a与在 n 个处理器上并行处理完成相同问题所用时间T_n比值。求解公式如下：

$$S=\frac{T_a}{T_n}$$

η：负载并行效率，定义为多核系统在未采用动态调度算法情况下完成所有任务使用时间T_1与采用动态调度算法求解相同问题使用时间T_2之差和T_1的比值，求解公式如下：

$$\eta=\frac{T_1-T_2}{T_1}$$

下面从这两方面对结果数据进行分析。

（1）　加速比S

加速比S统计表格如表 5-4 所示。

表 5-4　加速比 S 统计结果

	任务集 1	任务集 2	任务集 3	任务集 4	任务集 5
多核未采用动态算法	2.21	2.46	2.51	2.55	2.58
RefineCommLB 调度策略算法	2.35	2.63	2.78	2.80	2.84
TDTS 算法	2.14	2.38	2.8	2.85	2.92

加速比 S 统计图如 5.2 所示：

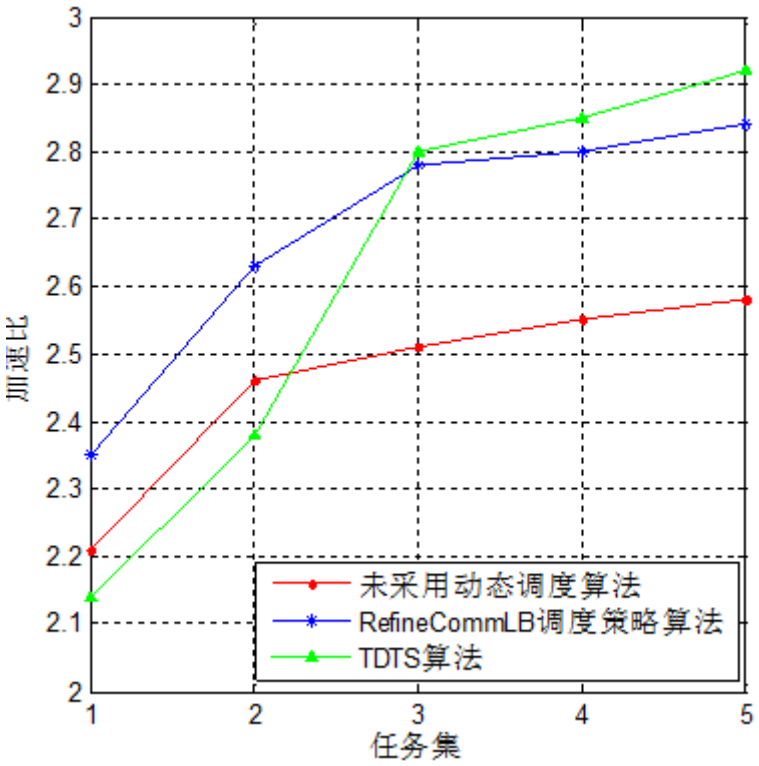

图 5.2　加速比 S 统计图

加速比 S 统计结果表明，当任务数量比较少，计算规模不大时，两级调度策略的加速比低于未采用动态调度策略和 RefineCommLB 策略。这是由于 TDTS 算法在执行过程中不但需要搜集节点负载信息，而且相对于 RefineCommLB 策略，除了考虑节点之间的通信开销还加入了任务的优先级和任务的 level 值，因此 TDTS 算法的复杂度较高，算法的优势在计算规模比较小时无法显示出来。但是随着任务规模不断增大，TDTS 算法的加速比增快速度超过未采用动态调度策略和 RefineCommLB 策略。当任务规模超过一定量时，TDTS 算法已开始优于 RefineCommLB 策略。.

（2）　并行效率

并行效率 η 统计表格如 5-5 所示。

表 5-5　并行效率η统计结果

任务数	任务集 1	任务集 2	任务集 3	任务集 4	任务集 5
RefineCommLB 调度策略	0.060	0.062	0.096	0.089	0.090
TDTS 算法	-0.030	-0.036	0.102	0.113	0.116

并行效率η统计图如 5.3 所示。

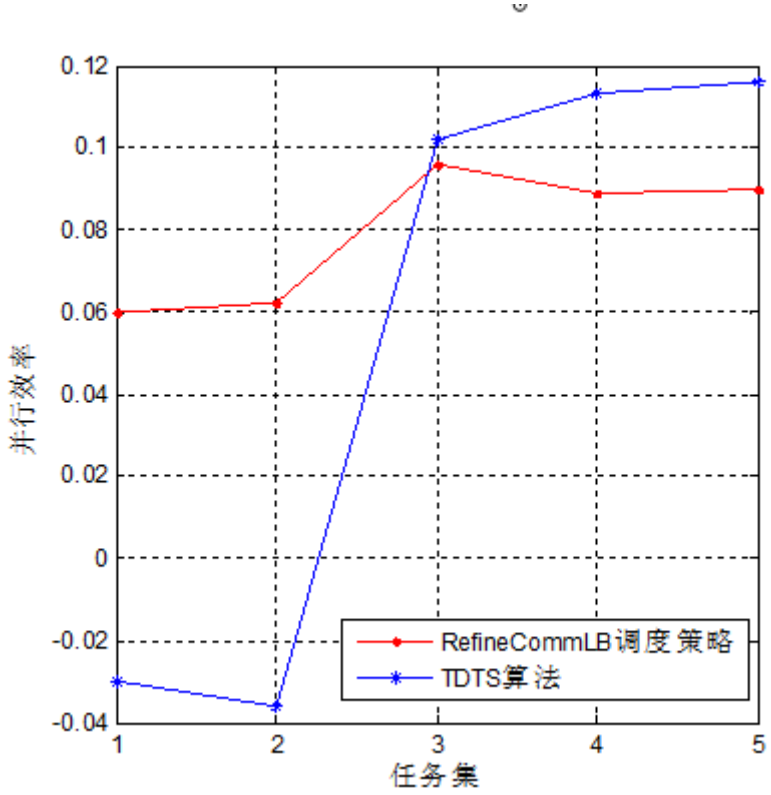

图 5.3　并行效率η统计图

并行效率η统计结果表明，随着任务规模、计算量的不断增大，两级动态调度策略的并行效率明显优于 RefineCommLB 策略。

综上所述，提出的动态调度策略减小了任务执行时间，尤其随着任务规模不断增大，执行效果更加明显。

5.3　本章小结

通过在搭建的 M5 仿真平台上设计实验验证了提出的 TDTS 算法的高效性。测试用例利用 TGFF 工具生成，并作为动态任务调度算法性能测试实验的输入。为了得出公正、合理的结论，实验中将分别在未采用动态调度算法、采用 RefineCommLB 调度策略算法与采用 TDTS 算法环境下执行的结果进行对比，并选择加速比和负载平衡效率作为策略性能评价指标。实验结果证明了提出的 TDTS 算法的高效性。

结　　论

随着嵌入式实时操作系统的不断发展，实时处理能力越来越强，嵌入式硬件平台的功耗和性能矛盾将会日益突出。多核技术为此提供了一条很好的解决思路，然而与之相配套的系统软件的发展却相对滞后。因此，研究支持多核的嵌入式实时操作系统具有重要意义。

RTEMS 嵌入式操作系统在实时性、安全性方面都表现优异，并且对多核提供了很好的支持，本文主要研究支持 SMP 架构的 RTAMS 操作系统多核化技术。完成的工作包括：

1）剖析 RTEMS 操作系统内核，主要分析了 RTEMS 体系结构和微内核设计思想，并对 RTEMS 内核中重要构件如任务管理、存储管理、信号管理等进行研究。

2）探讨 RTEMS 系统多核化实现机制，针对单核提供机制无法保证多核之间正常通信问题，提出改进的任务自旋锁和中断自旋锁相结合机制。

3）对 RTEMS 系统多核任务调度策略进行改进，提出两级动态任务调度算法，该算法充分利用了多核系统的并行性，提高了系统整体性能。

由于受硬件和时间等各方面的限制，论文还存在以下几个方面的问题有待进一步研究：

1）RTEMS 单核系统通过一个外部中断控制器实现外设与处理器之间的通信。在多核系统中如果采用相同的机制，由一个确定的核来实现外部中断控制的功能，会使得运行在该核上的任务被频繁的中断得到不公平待遇。

2）当对 TDTS 算法的性能进行测试时，采用 TGFF 工具生成测试用例与实际应用情况有一定差距。

参考文献

[1] 谢向辉，胡苏太，李宏亮．多核处理器及其对系统结构设计的影响．计算机科学与探索．2008: 233-235 页

[2] Amdahl G M. Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities. New York: ACM Press. 2004: 483-485P

[3] Varhol, P. Multicore Matters. Desktop Engineering. 2009:67-69P

[4] Kahle J A, Day M N, Hofstee H P, et al. Introduction to the cell multiprocessor. IBM Journal Research And Development. 2005:589-604P

[5] 胡威．嵌入式系统的发展．国际学术动态．2010:32-33 页

[6] 孙鲁毅．四种流行的嵌入式实时操作系统的比较研究-VxWorks,QNX,ucLinux,RTEMS．计算机应用与软件．2007，24(8): 196-197 页

[7] Agudo, Isaac,Fernandez-Gago, Carmen, Lopez, Javier. Concurrent access control for multi-user and multi-processor systems based on trust relationships. Concurrency and Computation: Practice & Experience. 2009: 34-38P

[8] 何军，王飙．多核处理器的结构设计研究.计算机工程．2007: 14-15 页

[9] Wind River Systems. VxWorks SMP Programmer's Guide. 2007

[10]王 伟，都思丹．基于 MPCore 与 Linux 的中断亲和性研究．南京大学学报．2009: 123-125 页

[11]Tim Jones M. Inside the Linux scheduler. 2008

[12]Scott Andrew Maxwell, Scott Maxwell. Linux Core Kernel Commentary, 2nd Editiontion. Coriolis Group Books. 2008: 45-48P

[13]RTEMS DeveloPment Environment Gtlid Edition 4.6.1. 2012

[14]许柯，陈跃新．基于 RTEMS 的多处理器任务调度机制的设计．计算机工程与科学．2005: 80-86 页

[15]刑群科，郝红卫，温天江．两种经典实时调度算法的研究与实现．计算机工程与设计．2006，1: 117-119 页

[16]Vakili S, M.Fakhraie S, S.Mohammadi. Evolvable multi-processor: a novel MPSoC architecture with evolvable task decomposition and scheduling. Computers and Digital Techniques, IET. 2010: 143-156P

[17]Barbosa J, Belmiro Moreira. Dynamic Job Scheduling on Heterogeneous Clusters.

Parallel and Distributed Computing, ISPDC '09, Eighth International Symposium on. 2009: 3-10P

[18]uan Antonio Clemente, Javier Resano, Carlos Gonz á lez et al. A Hardware Implementation of a Run-Time Scheduler for Reconfigurable Systems. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. 2010:1-14P

[19]Zexin Pan and Earl Wells B. Hardware Supported Task Scheduling on Dynamically Reconfigurable SoC Architectures. Very Large Scale Integration(VLSI) Systems, IEEE Transactions on. 2008: 1465-1474P

[20]肖红，周朴雄．嵌入式多核系统软件设计和开发．现代计算机．2008: 55-58 页

[21]Todman T J, Constantinides G A, Wilton SJE, et al. Reconfigurable computing: Architectures and design methods. IEE ProcComput Digit Tech. 2005,152(2):193-207P

[22]陈渝，Ray，吴德新，邓祺．开放源码的嵌入式硬实时操作系统 RTEMS 内核分析与开发．2007: 46-48 页

[23]Plurality LTD. Announce its new hypercore architecture line of multicore processor. http://www.plurality.com, 2007

[24]McNairy C, Bhatia R. Montecito: A Dual-core, Dual-thread Itanium Processor. IEEE Micro. 2005: 10-20P

[25]李红卫．RTEMS 存储管理的研究与改进．微计算机应用．2008: 63-66 页

[26]谭琦，桂先洲．RTEMS 消息管理机制的剖析和验证．计算机仿真．2005: 252-256 页

[27]LUI SHA, Priority Inheritance Protocols:An Approach to Real-Time Synchronization IEEE transactions on computers. 2004: 1175-1185P

[28]JACK L. LO and SUSAN J. EGGERS, JOEL S. EMER, HENRY M. LEVY, REBECCA L. STAMM, DEAN M. TULLSEN. Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading. ACM. 1997: 322-354P

[29]Kongetira P. A 32-Way Multithreaded SPARC Processor. IEEE Micro. 2005: 21-29P

[30]王炜，汤志忠，乔林．片上多处理器互连技术综述．计算机科学．2008: 24-26 页．

[31]Kahle J A. Introduction to the Cell Multiprocessor. IBM Journal Res. & Dev. 2005，49(4/5): 589-604P

[32]Minaie, Afsaneh, Sanati-Mehrizy, Reza.Integrating multi-core techniques into undergraduate computer science curriculum. Utah Valley State College. 2009: 12-17P

[33]Microsoft. The manycore shift:Microsoft parallel computing initiative ushers computing

into the next era. Microsoft. 2007: 45-48P

[34]Qing Liand, Carolyn Yao. Real-Time concepts for Embedded systems. New York:Cmp Books. 2003: 13-16P

[35]Jing Chen, Jian-Hong Liu. Developing Embedded Kernel for System-on-a-Chip Platform of Heterogeneous Multiprocessor Architecture. IEEE RTCSA. 2006: 45-49P

[36]张昆峰，常进. RTEMS 操作系统在 SPARC-V8 处理器上的应用. 微计算机信息. 2009: 10-12 页

[37]Wayne Wolf. The Futue of Multiprocessor Systems-on-chips. DAC 2004, June 7-11. 2004: 681-684P

[38]阎淼，赵军锁，张文君．基于 RTEMS 的实时进程设计与实现．计算机工程与设计．2009， 30(17): 3928-3931 页

[39]周本海，乔建忠，林树宽．多核平台的并行实时调度与内存分配算法．东北大学学报．2012: 357-360 页

[40]Olukotun K, et al. The case for a single-chip multiprocessor. ACM SI GPLAN Notices. 1996(9): 2-11P

[41]李飞，王岩飞．实时操作系统在星载计算机中的应用．计算机应用．2004: 21-23 页

[42]许柯．嵌入式操作系统 RTEMS-for-SPARC 的研究与设计．国防科学技术大学硕士学位论文．2005: 22-29 页

[43]Agudo, Isaac,Fernandez-Gago, Carmen, Lopez, Javier. Concurrent access control for multiuser and multi-processor systems based on trust relationships. Concurrency and Computation: Practice & Experience. 2009: 67-69P

[44]诸利勇．支持多核处理器的星载嵌入式操作系统的研究与实现．国防科学技术大学硕士学位论文．2008: 34-38 页

[45]彭正文，徐新爱. 基于 SMP 的 Linux 内核自旋锁分析. 江西教育学院学报. 2005: 23-26 页

[46]Youngchul Cho, Sungjoo Yoo, Kiyoung Choi, etc. Scheduler Impementation in MP SoC Design. 2005: 151-156P

[47]Dreslinski, Ronald G; Blaauw, David; Mudge, Trevor; Sylvester, Dennis. Energy efficientnear-threshold chip multi-processing. International Symposium on Low Power Electronics and Design. 2007: 234-236P.

[48]杨际祥，谭国真，王荣生. 并行与分布式计算动态负载均衡策略综述. 电子学报. 2010:

1122-1129 页

[49]彭蔓蔓，黄亮. 多核处理器中任务调度与负载均衡的研究. 微电子学与计算机. 2011: 35-39 页

[50]Bini E, Buttazzo GC, Buttazzo G. Rate monotonic analysis:the hyperbolic bound.IEEE Trans. on Computers. 2003, 57(7): 59-66P

[51]Godfrey B, Lakshminarayanan K, Surana S, et a1. Load balancing in dynamic structured P2P systems[C]. In:IEEE INFOCOM, Hong Kong. 2004: 724-729P

[52]LIU YI, ZHANG XIN, LIHE, et al. Allocating tasks in multi-coreprocessorbased parallel system[C]//Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops Washington DC: IEEE Computer Society. 2007: 748-753P

[53]Static and dynamic temperature-aware scheduling for multiProeessor SoCs. IEEE transactions on very large scale integration(VLSI) systems. 2008: 1127-1140P

[54]Sanjoy Baruah, Nathan Fisher. The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems. IEEE Transactions on Computers. 2006: 918-923P

[55]Nijhuis M, Bos H, Bal H,et al. Mapping and synchronizing streaming applications on cell Processors//High Performanee Embedded Architectures and ComPilers. 2009: 216-230P

[56]马晓慧，陈娟. 一种基于数据划分和任务映射的并行调度算法. 现代计算机. 2012: 7-10 页

[57]Hill M, Marty M. Amdahl’s law in the multicore era. Computer. 2008: 33-38P

[58]王晶，樊晓桠，张盛兵，王海. 多核多线程结构线程调度策略研究. 西北工业科技. 2007: 56-57 页

[59]Nacul A, Regazzoni F, Laiolo M. Hardware scheduling support in SMP architectures. Design Automation & Test in Europe Conference & Exhibition, DATE’07. Nice Acropolis. 2007: 1-6P

[60]Park S, Hong D,and Chae S. A hardware operating system kernel for multi-processor systems. IEICE Electronics Express. 2008: 296-302P

[61]Xia B, Wu K, Xiang C,et al. Network interface design based on mutual interface definition. International Journal of High Performance Systems Architecture(in press). 2010: 66-67P

攻读硕士学位期间发表的论文和取得的科研成果

[1] 2010.09-2011.09：参与了信息安全评测服务支撑平台的需求分析和业务开发，该项目来源于黑龙江省电子信息产品监督检验院。

[2] 2011.09-2012.09：参与高速列车网络控制系统课题研究，该项目属于国家“十一五”重大科技支撑计划项目。

[3] Lu Li, Guoyin Zhang, Jinyuan Nie, Yingjiao Niu, Aihong Yao. The Application of Genetic Algorithm to Intrusion Detection in MP2P Network. Advances in Swarm Intelligence. 2012: 57-63 P

致　　谢

伴随着论文工作的结束，也要为我将近两年半的研究生生活画一个句号了。有机会在哈尔滨工程大学这座学术殿堂学习，感受资深教师的人格魅力，是我的荣幸。在学校受到的学术熏陶，是我人生的宝贵财富，她将继续影响着我以后的工作和生活。在论文即将结束的时候，衷心的感谢母校对我的培养，并向所有给予我帮助与支持的老师、同学和亲人表达最诚挚的谢意。

感谢我的导师张国印教授。在我的研究生生涯中，张老师给予了我很大的帮助和支持。同时，张老师深厚的学术功底、渊博的知识、严谨的治学态度使我受益匪浅。在张老师的悉心教导下，我的专业技能有了很大提高，而且我还学会如何做人、做学问。在论文编写整个过程中，张老师更是耐心指导，在每次遇到困难或迷茫的时候都会给出建议，为我指明方向。在以后的工作、生活中我会以您为榜样，努力做一个认真的人。也借此机会祝张老师工作愉快，身体健康，桃李满天下。

我也要感谢高伟副教授，高老师在我论文进行过程中给了很大的帮助，在论文思路、格式排版、言语措辞等各方面都耐心指导，在后期忙碌中也会抽时间查阅，指出不足，给出建议，生活上的关怀和鼓励也让我倍加感动。感谢实验室和抽时间参与我论文答辩的所有老师给予我的帮助，在以后的道路中，我会更加努力，不辜负你们对我的期望。祝所有老师健康快乐，万事如意。

其次，我还要感谢姚爱红老师。姚老师渊博的知识、严谨的工作态度和和蔼友善的待人态度，深深地影响和激励着我。正是因为有了她严格、无私和高水平的教导，我才能在这两年的学习过程中，不断地汲取知识和迅速提升能力。借此机会，向姚老师致以深深地谢意。

还要感谢实验室的兄弟姐妹们，谢谢你们的关怀和帮助。和你们相处非常快乐，使我感受到家一般的温暖。不管走多远，我们永远是亲人。祝你们学业有成，前程似锦。

最后感谢我一生都在操劳的父母，谢谢你们给我的无穷无尽的爱，有你们的支持和鼓励，我会更加勇敢！

支持多核处理器的RTEMS嵌入式操作系统的研究

作者：牛莹姣
学位授予单位：哈尔滨工程大学