分类号：_____ 密级：_____

编号：_____

桂林理工大学

硕 士 研 究 生 学 位 论 文

基于多核处理器的嵌入式微内核操作系统通信机制研究与设计

专　　业：　计算机应用技术

研究方向：　嵌入式系统应用技术

研 究 生：　许安明

指导教师：　程小辉 教授

论文起止日期：2012 年 4 月至 2013 年 4 月

The research and design of communication mechanism of embedded micro-kernel operating system based on multi-core processor

Major：Computer Application Technology

Direction of Study：Embedded System Applied Technology

Graduate Student：Xu An ming

Supervisor：Prof. Cheng XiaoHui

College of Information Science and Engineering
Guilin University of Technology
April,2012 to April,2013

研究生学位论文独创性声明和版权使用授权书

独 创 性 声 明

本人声明：所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含他人已经发表或撰写过的研究成果，也不包含为获得其它教育机构的学位或证书而使用过的材料。对论文的完成提供过帮助的有关人员已在论文中作了明确的说明并表示谢意。

学位论文作者（签字）：许安明

签字日期：2013年6月4日

学位论文版权使用授权书

本学位论文作者完全了解(学校)有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的印刷本和电子版本，允许论文被查阅和借阅。本人授权(学校)可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。同时授权中国科学技术信息研究所将本学位论文收录到《中国学位论文全文数据库》，并通过网络向社会公众提供信息服务。(保密的学位论文在解密后适用本授权书)

学位论文作者签名：许安明　　导师签字：程小舟

签字日期：2013年 6月 4日　　签字日期：2013年 6 月 4 日

摘　要

相对于传统的宏内核操作系统来说，微内核操作系统由于其内核体积小、灵活性高等众多优点，已经广泛用在航空航天、汽车等多个领域。越来越多的应用需要可靠性更高性能更好的微内核操作系统以及硬件环境支持。同时，单核处理器的功率消耗大、电路复杂等问题已经日益突出，使单核处理器性能提高的空间到了极限，所以具有并行处理能力的多核处理器取代单核处理器成为嵌入式微内核操作系统最佳选择平台。但是现有的支持异构多核的微内核操作系统性能并不高，而通信机制是影响其性能的关键因素。设计一种更高效的面向多核的微内核操作系统通信机制已经迫在眉睫。

本文首先对多核处理器和微内核操作系统进行了阐述，并介绍了总线共享、交叉开关互连、片上网络三种多核通信架构以及几种典型的支持多核的嵌入式实时微内核操作系统。然后，介绍了进程间通信原理及其两个阶段，描述多核处理器进程间通信特点和资源竞争特点，并深入分析六种典型通信方式和几种嵌入式微内核操作系统的进程间通信策略的优缺点。在此研究基础上，从核内和核间两个角度，设计了一种基于消息分类的线程代理通信机制，从如下几个方面进行了改进和创新：

（1）对于核内通信，设计了消息分类的通信策略，该策略充分利用寄存器和共享内存的地址映射通信方式的优点，将消息交由线程代理判定其长短，短消息采用寄存器方式进行通信，而长消息采用共享内存的地址映射方式进行通信。

（2）对于核间通信，利用轻量级线程，采用共享内存的地址映射方式，设计了基于代理线程的核间通信方案。在设计的核间通信机制中，通信由专门的主内核协助完成。

（3）为了提高通信时分配共享内存的效率，首次提出了可预测的共享内存分配算法。该算法利用马尔可夫链预测原理，通过转移概率矩阵评估下一次内存分配的概率大小，系统可以提前分配内存，减少分配等待时间。

（4）通过结合优先级和等待时间改进了通信优先级策略，并通过注册方法提升中断管理效率。同时，利用单核中信号量的同步原理，针对核间同步新问题，设计信号量管理策略。

文章最后通过 Simics 仿真器构建了多核实验平台，通过测试代码分别在基于消息分类的线程代理通信机制和邮箱通信机制的模拟系统上进行测试，实验结果表明本文设计的基于消息分类的线程代理通信机制更加高效，提高了面向多核的嵌入式实时微内核操作系统的性能。

关键词：多核处理器，微内核操作系统，通信机制，核内通信，核间通信

Abstract

Compared with traditional monolithic operating system, microkernel operating has been widely used in the aerospace, automobile and other fields system due to the small size of the kernel, high flexibility and many other advantages. More and more applications require higher reliability, better performance of the microkernel operating system and hardware environment support. At the same time, Single-core processor power consumption and circuit complexity has become increasingly serious, which has hindered the development of single-core processor, thus multi-core processors with parallel processing to replace the single-core processors becomes the best choice of embedded microkernel operating system. But the performance of the existing micro-kernel operating system which supports heterogeneous multi-core system is not so high, meanwhile the communication mechanism is the key factor that affects the performance.Therefore, the study of a more efficient micro-kernel operating system communication mechanism for multi-core is essential.

This paper firstly introduced multi-core processors, micro-kernel operating system, and three multi-core communication architecture—shared bus, crossbar interconnect, and network-on-chip, and several typical embedded real-time micro-kernel operating system which supports multi-core processor. The paper introduces the principle of inter-process communication and its two phrases, describes the characteristics of multi-core processors communication and the features of resource competition, and analyzes the advantages and disadvantages of six typical means of communication and several embedded micro-kernel operating system inter-process communication strategy. Based on the research, from the perspective of inner-core and core-to-core, a thread agent communication mechanism based on message classification is designed, and it is improved and innovated from the following aspects:

(1)In inter-core communication, the strategy of message classification is suggested, this strategy makes full use of registers and shared memory address mapping communication mode, a thread proxy decides the length of a message, the communication in the form of short message using register, for long messages using shared memory address mapping approach to communication.

(2) In the inter-core communication mechanisms, using lightweight threads, shared memory address mapping, to design inter-core communication scheme based on the

agent thread. In the inter-core communication mechanism design, the communication is completed by the assistance of specialized communication management kernel.

(3) In order to improve the efficiency of the allocation of shared memory communication, the predictable shared memory allocation algorithm is put forward for the first time. The algorithm uses the Markov chain theory, to assess the next memory allocation probability through transition probability matrices, the system can allocate memory in advance, reducing the allocated waiting time.

(4) Through a combination of priorities and waiting time priority strategy, improving communication and interrupt management efficiency through registration method. Meanwhile, based on the single-core semaphore synchronization principle, design a semaphore management strategy for synchronization problems in the nuclear.

Finally, a multi-core experimental platform is built based on the Simics emulator, through the test code in the thread of the message classification agent communication mechanism and mail communication mechanism-based simulation system for testing, experimental results show that the design of agent communication mechanism based on message classifications is more efficient, the performance of multi-core embedded real-time microkernel operating system is improved.

Key words: Multi-core processors, Microkernel operating system, Communication architecture, Communications mechanisms, communication in them of core

目 录

第 1 章 绪论

1.1 课题研究的背景及意义

1.1.1 课题研究的背景

操作系统自从二十世纪六十年代诞生以来，经历了手工操作系统、早期批处理阶段、执行系统阶段、多道程序系统阶段、分时系统、实时系统和通用操作系统等发展阶段[1]。它作为计算机系统的软硬件资源管理者，其性能的好坏会对计算机系统产生直接的影响。嵌入式微内核操作系统作为操作系统的一个重要分支，广泛被应用于军事、医疗、工业等诸多领域。从通信设备、高端仪器等新型电子产品，到电视、空调等传统电器，都可以看到它的身影。由于微内核操作系统普遍采用系统功能模块化的设计思路，把进程调度、进程间通信、内存管理等核心功能放在内核中运行，把对系统运行影响不大的功能放在外部服务器运行，从而压缩了操作系统的体积，达到以模块化的方式实现系统组成的目的。它具有以下优势：

（1）可移植性强。微内核操作系统和硬件之间起着桥梁作用的硬件抽象层为上层提供服务，是微内核操作系统中与硬件相关的部分。这种设计使得嵌入式操作系统在移植的时候就不必考虑的硬件相关部分。也就是说，微内核操作系统和硬件是无关的，微内核操作系统的系统功能已经完全与硬件分离开来，进而简化了微内核操作系统的设计结构。在对微内核操作系统进行系统移植的过程中，研究人员只需要修改硬件抽象层的相关代码，而不需要直接对内核进行大规模修改，减少了研究人员的工作量，同时也降低系统代码出错的风险。

（2）有较强的扩展性。嵌入式微内核操作系统把系统运行所必须的核心功能留在内核运行，把其他的功能放在服务器的设计思想也使得微内核操作系统在功能扩展时只需要修改服务器中的相关代码，不需要修改内核代码。在某种程度来说，这样设计思想降低扩展系统功能的复杂度，提高了修改和升级微内核操作系统的效率。

（3）安全性强。对于微内核操作系统来说，每个进程都有自己的内存空间，相互之间是隐藏的。内核与服务器之间都是通过进程间通信（IPC）实现系统功能的信息交互。即使服务器提供的服务出现错误时，系统的运行也不会受到很大影响，不会导致系统发生崩溃，从而实现系统的高可靠性和高安全性。

微内核操作系统的这些优点引起了系统开发者高度的兴趣，使得更多的研究人员投入到微内核操作系统的研发中。在嵌入式微内核系统飞速发展的今天，因单核主频的提高是以功耗及发热量几何倍数增长为代价的，这使得芯片运行的越来越不稳定[2]。毫无疑问，单核处理器的功率消耗、内存延迟和复杂的电路设计等问题已经日益突出，这些

成为进一步提高单核处理器性能的最大阻碍，使单核处理器发展空间达到了极限。因此，斯坦福大学在 1996 年率先提出多核处理器的思想后，即在单个芯片上集成两个以上处理器内核，研究者和厂商们将提高主频的重点转向了多核处理器。IBM 公司在 2001 年推出了自己的多核处理器 POWER4，AMD 和 Intel 公司也相继推出自己的多核处理器，如今多核处理器已经逐渐取代单核成为市场主流。

1.1.2 课题研究的意义

伴随电子技术的进步和制作工艺的成熟，多核处理器现在已不只服务于高级计算机、服务器等高端领域，它逐渐走入普通个人电脑配置中。虽然嵌入式微内核操作系统拥有许多优点，但微内核架构也不是完美的，性能不高是它的一个致命缺点。多核处理器的运用可以大大提高微内核操作系统的性能，促使微内核操作的发展达到了一个新台阶。然而，目前还没一个能很好适用于多核处理器的微内核操作系统，所以研究多核处理器平台的微内核操作系统，特别是研究多核处理器之间的通信等关键技术就显得更为重要。通信机制作为微内核操作系统不可分割的一部分，不但要完成多核处理器之间的信息交换基本功能，而且要有效保障系统的实时性和稳定性，为嵌入式微内核操作系统在多核处理器应用提供支持，使多核处理器下的微内核操作系统理论研究成果成为现实，并被得到实际应用。在单内核的微内核操作系统通信中，系统调用服务，仅需要两次转换用户态和内核态，每个进程都同时拥有可以存放信息的用户栈和内核栈。系统通过发送进程间通信消息给服务应用程序来调用服务，接着服务应用程序响应服务请求，并把结果返回给调用者需要通过另一个 IPC 消息[3]。这个过程需要切换进程的上下文等复杂操作，所以需要花费额外的拷贝时间来传送消息。由此可知，通信机制是提高多核处理器的嵌入式微内核操作系统性能的一个重要方面。

总的来说，多核处理器架构已经逐渐成为嵌入式操作系统的主流环境，但是支持多核处理器的嵌入式微内核操作系统尚未形成标准，在国际市场上也还没有相对成熟的产品，在国内对面向多核的微内核操作系统的研究更少，尤其是其通信机制还有待改善与提高。因此，需要研究并设计一套基于多核的微内核操作系统通信机制，发挥多核处理器的优势，减少通信延迟，提高系统的可靠性和实时性，为嵌入式微内核操作系统在多核环境下的研究奠定基础。这对于嵌入式微内核操作系统发展具有极其重要的意义。

1.2 国内外研究现状

为了充分利用嵌入式微内核系统和多核处理器的优势，研究支持多核的嵌入式微内核操作系统已经引起了学术界的广泛关注，国内外的一些研究者对其进行了研究。面向多核的嵌入式操作系统的设计思想起源于卡内基梅隆大学在 1974 年提出的 HYDRA 操

作系统。HYDRA 操作系统是在 C.mmp 硬件上运行的内核，也是最早应用于多核处理器的操作系统之一。卡内基梅隆大学在几年后又研发出了一款能支持多核的操作系统叫 Star OS。它继承了 HYDRA 操作系统的思想，又增加了一些新的概念。随后出现的 iMAX、类 UNIX、Choices、CHAOS、L4 等操作系统都支持多核环境，它们通过快速交互信息来执行进程。目前研究人员研究支持多核的微内核操作系统的思路可以分为三种：第一种是通过修改原有的的微内核操作系统成为支持多核的操作系统，然后使之适应嵌入式环境；第二种是开发新的支持多核的嵌入式微内核操作系统；第三种是通过改造现有的嵌入式微内核操作系统成为符合多核平台的操作系统。在通信机制具体设计中，许多研究者进行了一些方法改进和创新。

Mamidala Amith R 等针对 Intel Clovertown 多核操作系统中共享 L2 Cache 的特点，提出了两层的通信算法来优化 MPI_Bcast 和 MPI_Allgather，同时使用 NUMA 特点，优化共享内存的 MPI_Allreduce，并使用通信调度的方式充分利用 HT 总线的双向带宽，以优化 MPI_Alltoall 性能[4]。该研究重点关注了 Intel Clovertown 系统的硬件特点，而不是多核系统普遍的特点，以及多核环境对集合通信算法的影响。Graham Richard L 等人设计了共享内存来优化多核环境下的同步操作，研究通过小消息时树形通信算法的度的选择，以及大消息时并发访问度的问题[5]。该研究更多的是关注多核环境下集合通信算法实现的问题。张攀勇等人提出了层次化算法、限制并发、NUMA 感知的优化方法和 Cache 友好的优化算法[4] [5]，有效地利用多核结构特点，降低竞争带来的影响，提高了多核环境下集合通信的性能和可扩展性。Andrew Baumann 等人提出采用网络功能的本质特点，借鉴分布式系统的观点，把每一个核看作一组通过网络互连的独立的节点，通过核间通信显式进行、操作系统结构与底层硬件无关、以复制而不是共享的三个基本原则进行设计[7]。有些学者认为基于 NoC 的 MPSoC 与分布式系统有很多相似之处，NoC 网络对应着网络连接，通过 NoC 互连的多核操作系统可以看作是分布式系统的芯片化实现。Vincent Nollet 等提出了通过在操作系统中集成通信资源管理来合理地将并行应用中的任务映射到多个节点，在系统中实现了动态信息统计、动态注入率控制、操作系统控制自适应路由等功能。其独到之处在于将一个操作系统拆分成多个部分，其中操作系统内核运行于主节点上，而每个从节点则执行有限的操作系统功能。操作系统的某项功能实现就如同远程调用，主节点将参数通过消息发送到对应的从节点，从节点完成相应的功能后，通过消息机制将结果返回给主节点。Enea 的 LINX 操作系统使复杂的应用程序更容易分割及分布，其消息通信机制准确无误地连接上 OSE 和 Linux 操作系统，使系统具有良好的扩展性和软硬相对独立性。Wei Hu 等人在对集中式控制系统研究基础上，设计了一种全新的微内核操作系统，在实现过程中，每个节点上运行一个内核和其他的应用程序，这个内核为这些应用程序提供服务。当运行在系统上的应用程序需要服务时，微内核就需要调用系统的一些功能模块，如果这个功能模块在当前节点上，就直接调用；如果在其他节点上，就发送消息给相应的节点，那么这个相应的节点实现微内核想要的

功能。这样使得单核上的程序不要经过大幅度修改就可以适用于多核环境。

目前在嵌入式操作系统领域研究出一些支持多核处理器的操作系统产品，其中在国外研究中，比较著名的有 Rtems、L4、VxWorks 和 QNX 等。Rtems 操作系统采用消息传递通信的包驱动方式实现对多核环境的支持。Rtems 在进行核间通信时，先由系统的每个内核创建一个服务器线程，然后通信进程将消息交予这个服务器线程处理，对于进程双方来说，通信过程是透明的。从查阅的资料来看，L4 作为第二代微内核操作系统的代表，已经被应用在一些领域，并且也成为微内核操作系统的一个重要发展方向。由于其 IPC 机制引起的系统开销太大，严重影响了第一代微内核的性能，IBM Watson 研究中心的 Jochen Liedtke 研究设计了第二代微内核操作系统 L4。为了提高其 IPC 的性能，减少消息的复制次数，L4 采取通过通讯窗口机制，实现进程间的直接通信。Karlsruhe 科技大学的 Marcus Volp 提出并设计一款基于 SMP 架构的 L4 操作系统。在这个系统中设计了一个调度器，此调度器处于微内核的用户态中。它的功能是可以将一个核心上正在执行的线程转移到另个一个核心上运行。同时，也设计了一个在 L4 标准中进程通信机制没用的标志位，这个标志位的主要功能是标识某个进程通信操作是传给核内还是另外一个核的线程。VxWorks 6.8 在该操作系统的向下提供了一层虚拟层，并在该虚拟层实现了一套 MIPC Message Bus 机制。该套机制允许进程不必通过共同地址空间共享来进行彼此间的通信。提供了包括了 mipc_connect，mipc_sendto 等功能函数，其原理类似于 Windows Socket。在通信机制中屏蔽了底层硬件，但是通信效率也有所下降。QNX 是一个真正的微内核操作系统，它的驱动程序，应用程序，协议栈和文件系统都在内核外作为进程运行，从而几乎所有组件都可以失败再重启，而不影响内核或其他组件。QNX 使用消息传递作为进程间通信(IPC)的基本方式，因此其系统可扩展性强，模块化程度高和易于使用。

国内的科研院所也在不断的研究微内核操作系统，并且也做出了相应的微内核操作系统，如浙江大学的 SmartOSEK OS，其核间通信技术采用了核间符号表的形式，简单说来就是一个符号对地址的映射表[8]。两个核在通信时，一个核上的程序将符号信息和地址信息写入符号表内，然后另一个核上的程序便可以从表内将符号对应的地址取出来，从而实现系统通信。但是存在符号名字唯一和符号表容量有限等缺点，使用符号都需要从符号表中查找，所以写入符号表和查找符号表都会导致核间通信性能的下降，而且维护符号表需要额外开销。SmartOSEK OS 在单处理器上进程间直接利用消息对象进行数据交换，不需要通过底层网络和最后期限监控。多个消息接受者通过选择消息以队列或非队列的方式可以接收同一个消息发送者的消息。SmartOSEK OS 的进程间通信利用消息对象实现一种进程间以及中断通信机制，并提供了消息过滤、接受通知等功能。但是 SmartOSEK OS 的进程间通信的应用场景仅限于单处理器系统，对于多核的硬件环境却无能为力因此需要扩展核间的通信方式，使得处于不同核上的任务能够互相发送消息，实现核间的进程通信。在 SmartOSEK OS 基础上改进的 SmartosEKOS-M 为多核处

理器分配了同一块共享内存区用于存储核间通信的数据和控制信息，实现了多核之间的同步与通信。它拥有控制消息队列和数据消息队列，每一个消息队列控制头有分别指向消息队列的头部和尾部的两个指针，用来控制一个消息队列。在 2008 年，由国内外著名科研院所联合研发的 Corey 也是一款非常优秀的多核操作系统。它在核间利用应用程序来控制共享资源，从而使得多核系统中各个内核统一信息数据。国产 COSIX V2.0 是吸取 Mach3.0 的成功经验基础上，成功开发的微内核操作系统。它采用了基于微内核的多服务器结构，支持任务、进程通信、端口、线程（threads）、消息等概念，具有较高的结构化、模块化程序等优势，便于裁剪和移植。除此之外，还有兰州大学研发的 XtratuM 操作系统，这个操作系统采用了虚拟机的模式，通过共享内存和管道结合的方式进行进程间通信。

1.3 本论文研究的主要内容

支持多核处理器的嵌入式实时微内核操作系统作为未来操作系统发展的一种重要方向，受到许多研究者的关注，也取得了一定的研究成果。本文针对支持多核处理器的嵌入式微内核操作通信机制存在的待解决的问题，以现有的多核处理器核间通信和微内核操作系统通信策略研究成果为基础，通过研究异构多核处理器的特点及通信原理，结合当前嵌入式实时操作系统应用特点，从核间通信和核内进程通信两个角度，研究多核处理器的嵌入式微内核操作系统通信机制和方法，包括支持多核的嵌入式微内核操作系统通信机制架构、核内进程通信模型、核间通信机制、内存共享策略、中断和信号量管理等，从而构建一套能够适应多核的嵌入式微内核操作系统通信机制。研究主要内容具体如下：

（1）通过研究分析多核处理器和微内核操作系统相关技术，构建支持多核的嵌入式微内核操作系统通信机制架构。支持多核的嵌入式微内核操作系统通信机制架构是通信机制的布局，所以具有非常重要的地位。针对嵌入式系统的实时性、可靠性等特点，充分考虑核间通信和核内线程通信要求，对通信机制进行总体架构。

（2）研究并改进核内通信策略。由于核内通信相对于核间通信要更为频繁，因此从某种程度上来讲，嵌入式实时微内核操作系统的性能取决于核内进程间通信模型的性能。在新的架构设计中，原本的 IPC 机制可以用于处于同一个内核中的两个进程间的通信，但是运行的效率有待提高，所以需要改善现有的通信策略。

（3）研究核间通信机制。两个进程处于不同的内核中（也即处于不同的处理器核心上的情况），原本的 IPC 机制不能直接使用，需要设计新的 IPC 机制来应对这种情况。多核处理器系统，特别是异构多核处理器系统中，为了发挥各个核的性能特点，常常需要在核间传递一定规模的数据。因此，充分考虑到中断、进程优先级、同步等因素，设

计核间通信策略，成为多核处理器下嵌入式微内核操作系统通信机制的重点内容。

（4）研究共享内存的快速分配算法。在核内和核间通信都采用了共享内存的地址映射方式，因此，共享内存的快速分配对通信的性能有很大影响，所以研究高效地内存分配算法有助于提高通信效率。

1.4 论文组织结构

本文首先对多核处理器和微内核操作系统相关技术进行研究分析，借鉴现有嵌入式系统通信机制，设计了一种支持多核处理器的嵌入式微内核操作系统的通信体系结构，并进行详细描述，最后通过仿真实验进行模拟并分析性能。本文由六个章节组成，具体如下：

第一章：本章介绍课题的研究背景和意义，介绍了微内核操作系统及其通信机制研究的发展历程及其发展的情况，并分析了研究支持多核处理器的微内核操作系统的意义。然后概括了微内核操作系统通信机制研究的主要内容，在本章的最后介绍了本文对各章节内容的安排。

第二章：对多核处理器的概念以及体系结构进行了简单阐述，介绍了微内核操作系统相关知识，简述其发展历程。然后介绍并分析了共享总线、交叉开关互连、片上网络这三种典型的多核通信架构，并对几种支持多核的嵌入式实时微内核操作系统进行了阐述和分析。

第三章：介绍了进程间通信原理及其两个阶段，分析了多核处理器多进程的通信和资源竞争特点。紧接着，对六种典型通信方式进行了介绍，并分析了其优缺点。最后分析了 L4、Minix3 、QNX、VxWorks 以及 SmartOSEKOS_M 等嵌入式实时微内核操作系统的进程间通信策略。

第四章：作为本文的重点章节，本章首先确立了研究的通信机制的目标，分析了多核环境下通信机制独有的特点，设计一种基于消息分类的线程代理通信模式。然后，从核内和核间两个角度分别对通信机制进行改进和创新。在核内设计了消息分类的通信方式，寄存器方式被用于短消息通信，共享内存的地址映射方式被用于长消息通信；在核间结合主内核和代理线程的共享内存方式进行通信。同时，本文对进程间通信优先级、共享内存分配方法、通信中断和信号量管理进行了改进与创新。

第五章：本章介绍了可用于多核环境下微内核操作系统模拟的 Simics 仿真平台以及搭建过程。让本文设计的通信机制和邮箱通信机制进行了对比实验，通过同样条件下执行同一个程序得出实验结果，并对结果数据进行了分析。

第六章：概括了本文的主要内容，总结了本文的主要工作和成果，并对以后的一些工作进行了展望。

第 2 章 支持多核的微内核操作系统通信机制相关技术概述

2.1 多核处理器体系结构介绍

2.1.1 多核处理器的概念

多核处理器也叫单芯片多处理器，或者片上多处理器(CMP) ，是指一个处理器上集成两个或多个内核，并通过并行总线将各个核连接起来[9]。多核处理器的产生和发展，是电子技术发展和市场需求的必然产物。自从片上多处理器思想和多核结构原型首次被提出，到现在仅仅经历了十几年的发展，多核处理器已经逐渐取代单核处理器成为了市场的主流。它解决了单核处理器出现的性能瓶颈，能很好的满足用户的需求。同时它在系统的安全性、虚拟技术等方面表现的非常出色，给用户提供了更快、更高效、更安全的体验，提高了计算机的工作效率。多核处理器在同一个芯片内集成多个核心，使所有的核心作为一个统一的整体对外提供服务，对用户来说具有透明性。多核处理器主要有以下几个显著优势：

（1）高主频。由于单芯片多处理器结构只包含极少的全局信号，所以控制逻辑相对比较简单[10]。故在同等工艺条件下，CMP 比超标量微处理器和超长指令字微处理器的主频高得多。

（2）低通信延迟。多个内核被集成在同一片内，极大地缩短了核间的通信距离，降低了核间通信延迟，提高了系统通信效率，而且采用共享 Cache 或者内存等手段，多线程的通信也会明显降低通信延迟，同时也提高了数据传输带宽。

（3）低功耗。多核处理器通过动态调节电压/频率、负载优化分布等方式有效降低功耗[11]。并且多核结构通过共享资源，提高了片上资源的利用率，减少了许多器件，也降低了功耗。

（4）控制逻辑简单。单芯片多处理器结构的控制逻辑比超标量微处理器结构和超长指令字结构简单。相应的单芯片多处理器的硬件实现必然要简单得多，从而可缩短设计周期和验证周期，节省研发成本。

（5）高并行性能。由于多核处理器集成了多个具有单线程或者多线程的处理内核，使得整个处理器可同时执行的线程数或进程数远超过单核处理器，极大地提高了处理器的并行计算性能。

这些优势促使多核处理器快速发展，但人类社会对计算性能需求的不断提高，迫使多核处理器也要不断提高性能。还有许多因素阻碍了多核处理器性能向更高水平前进的步伐，需要研究人员不断攻克和挑战，使多核处理器在现有技术基础上探索出新的思路和方法。

2.1.2 多核处理器体系结构

从体系结构角度看，多核处理器存在同构多核和异构多核这两种不同的架构。同构多核处理器是指计算机的内核相同，地位对等，没有主次之分。目前，大部分同构多核处理器是由通用的处理器核心构成，有单核处理器相似的结构，每个处理器核心可以独立地异步处理不同的任务。异构多核处理器的内核地位是不对等的，它们之间的功能和作用差异很大，通过 Cache 缓存进行互连，也可以通过共享内存进行互连，各个内核发挥自己最擅长的能力来提高整个系统的性能[12]。异构多核处理器一般采用“主从式”设计思想，使得多核的负载均衡和资源的管理遇到新的问题。其主要设计思想是当系统启动时，一个主内核首先启动，它负责整个系统的初始化以及管理其它内核。在系统运行后，只有主内核可以访问整个系统的资源，其它内核只能通过主内核调度来访问系统资源。异构多核处理器一般由如图 2.1 所示架构构成，它们有自己的高速缓存等私有存储空间，而且多个处理器间不共享前端，各个核只需要报交互的信息保存在共享存储器，减少多个处理器对内存中的地址和资源的访问冲突。

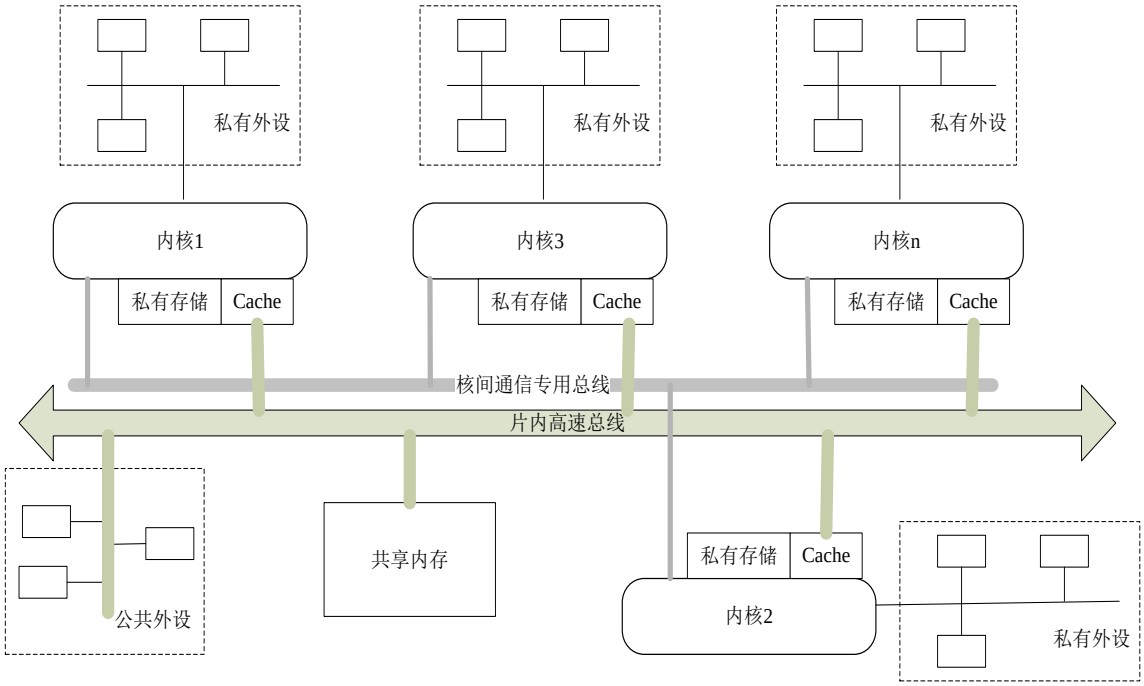

图 2.1 多核处理器架构

一种重要异构多核处理器的系统模型是每个处理器核心拥有一个内核，这样能更好地支持内核的异构特点。异构多核处理器能极大地提高系统的性能，是未来的多核处理器技术的发展趋势。因此，本课题也是以这种异构多核处理器为平台，研究微内核操作系统的通信机制。

2.2 微内核操作系统概述

2.2.1 微内核操作系统

所谓微内核其实是一种最小的计算机操作系统内核[13]，它本身不提供操作系统的相关服务，只提供实现这些服务的机制，诸如底层的进程间通信，地址空间管理以及线程调度等。也就是说它只对底层硬件进行很小的抽象，在内核中只保留那些内核正常运行所必须的核心功能模块，对于非核心的功能模块都被分离在核外以服务器的形式存在。这种设计使得系统的内核变得非常的精小，因此系统的移植性、扩展性、可靠性等特性得到了很大的提高。基于微内核的设计思想，微内核操作系统服务并不都运行在内核态中，而是把部分服务实现的机制运行于内核态，包括设备驱动、IPC 机制等。微内核的体系结构结构包括内核态和用户态[14]，如图 2.2 所示，在内核态下，微内核直接访问硬件。在用户态下，各个功能模块被分为独立的进程，每个进程提供单独的服务，这种服务器模型将多个进程联系在一起，相互协同工作，这样用户可以享用各种类型的服务。服务器采用相应的接口调用微内核，使各个相互独立的模块之间通过消息机制进行通信，从而保证了系统能对硬件的完整访问。

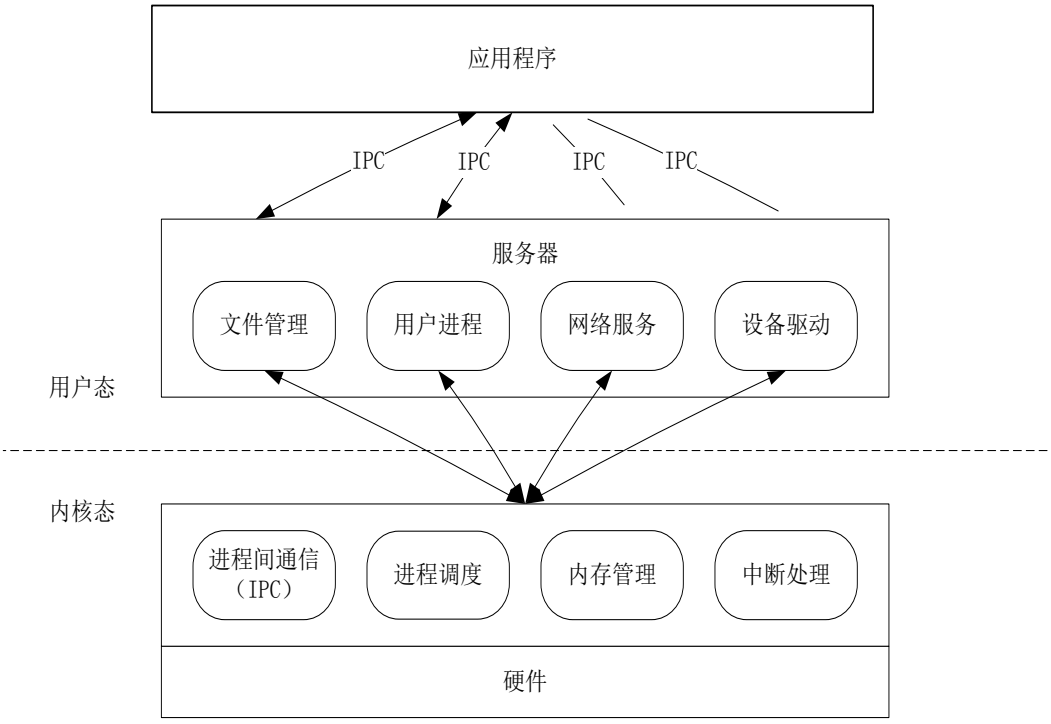

图 2.2 微内核操作系统的体系结构

由于微内核操作系统只是把最核心的功能放在内核态中，这就使得其内核相比于宏内核操作系统拥有一个更加小巧的体积，降低了系统内核的冗余程度，增加了系统的稳定性。同时，把其他系统功能放在外部服务器，采用模块化的设计思路[15]，每个系统功

能都被设计成为一个单独的系统模块，各个模块之间通过消息传递，实现信息的交互。这使得微内核操作系统在系统功能的删减上有了巨大的优势，增加了系统的灵活性。另一方面，系统的大多数功能都运行在服务器，也就是用户态中，所以当一个系统功能发生错误的时候，不会导致整个系统的崩溃，从而增加了系统的可靠性。

2.2.2 微内核操作系统的发展历程简述

在二十世纪七十年代，Richard Rashid 率先提出了微内核操作系统概念，并带领卡内基梅隆大学研究员利用消息传送机制，研发出第一代微内核操作系统---Mach。此后，许多研究人员相继投入到微内核操作系统的研发工作中来，从而产生了许多性能高效的微内核操作性系统。微内核操作系统的发展与其他所有事物的发展一样，通过四十多年的发展，经历了一个从无到有，从简单到复杂的过程。微内核操作系统的发展过程大致可以分为以下三个主要阶段：

（1）以 Mach 为代表第一代微内核操作系统，。

Mach 是卡耐基梅隆大学的 Richard Rashid 教授为了研究操作系统的分散与平行运算，对 BSD Unix 改进而设计出的操作系统[16]。像 Mach 这样的第一代微内核操作系统其实不是真正意义上按照微内核的架构思想来设计的，它们基本上是通过对宏内核进行适当地删减和改进而来的。因此，我们可以说第一代微内核操作系统实际上是介于传统宏内核操作系统和微内核操作系统之间的混合操作系统，所以它没继承宏内核操作系统的那种系统调度优势，又没有体现微内核操作系统的技术优势。加之，Mach 采用 port 机制实现进程间通信，频繁的上下文切换和内存空间的转化，导致了第一代微内核操作系统的效率低下，没有能够实现微内核技术思想设计的初衷。

（2）以 L4 为代表的第二代微内核操作系统。

二十世纪九十年代，IBM Watson 研究中心的约亨·李德克在研究 L3 的基础上研究设计了第二代微内核操作系统 L4。第二代微内核对第一代微内核进行了全面的改进，提高了效率，增强了系统的可靠性以及安全性。L4 在性能上得到了提高，这主要是得益于它采用了线程和地址空间这两种抽象以及映射机制和进程间通信策略[17]。应该说第二代微内核操作系统是第一次从实际应用的角度实现了微内核操作系统的设计思想，进而使得微内核架构的操作系统得到了广泛的应用。

（3）以 seL4 为代表的第三代微内核操作系统。

澳大利亚研究组织 NICTA 在 L4 操作系统上研发的第三代微内核操作系统 Secure Embedded L4，简写 seL4。seL4 作为第一个形式化机器证明（formal machine-checked proof）的通用操作系统，在学术界引起了极大的关注。seL4 拥有非常小的体积、高效的性能和很大的自由度。同时，它在微内核和应用程序级别内建了高性能的模型，提高安全性，促使微内核操作系统能够在对系统安全性要求较高的众多领域发挥其优势。虽

然这种设计思想不能够大规模应用于现有的微内核操作系统，还有待进一步研究，但它是将来微内核研究发展的一个方向。

2.3 多核通信架构

目前多核架构处理器已成为市场的主流，其多个核间相互协作和通信影响着处理器的功耗和性能。为了确保处理器的内核之间的数据同步与共享，提高处理速度就需要研究核间通讯结构。在多核通讯方式中，除了类似单核 SOC 中的总线共享结构，主要还有交叉开关互连（Crossbars witch）、片上网络（Network on-Chip ，NoC）等结构。

2.3.1 总线共享结构

总线共享结构是指片上多个内核、输入/输出端口、Cache 以及存储器之间的通信通过高速总线连接在一起[18]。总线结构由于较为简单，容易设计和实现，支持一些一致性协议，目前很多处理器都采用了此结构。比较典型的总线共享结构处理器有斯坦福大学研制的 Hydra、Intel 的 Core、IBM 的 Power4/5 以及 Cell 处理器等。但这种共享总线结构的局限性在于总线结构可扩展性较差，连接在片上的核心数较少，并且低延迟、高宽带、大尺寸之间相互制约。

2.3.2 交叉开关互连

当共享总线的多核处理器对总线的竞争，使处理器的并行性难以突破时。半导体制造技术的迅速发展，芯片的集成度越来越高，使得由交叉开关以及接口逻辑构成的交叉开关互连结构变为可能。由一些地址线、数据线构成的交叉开关和由一些加载队列构成的接口逻辑使连接在开关结构上的节点同时进行数据交换，最终让节点通过选择路径与其他节点进行点对点通信。与总线结构相比，交叉开关的优势具有多数据通道，更大的宽带。例如 AMD 公司的 Athlonx2 双核处理器、SUN 公司的 UltraSPARC T1 处理器等都是用交叉开关来控制核心与外部的通信。但交叉开关结构缺点是占用较大片上面积，并且随着核心数的增加也会导致性能下降。

2.3.3 片上网络

片上网络是借鉴了并行计算机的互联网络，使节点通过网络连接在一起，解决片上组件之间的通信问题[19]。目前，片上网络主要有树形、胖树结构、环、带弦环、星形、网格、环形网格等结构。在设计片上网络时，重点考虑多核互联通信效率与网络互连的开销[20]，以及片上网络的可重构性和可扩展性。与总线结构、交叉开关结构相比，片上

网络具有高可靠、强扩展性、低功耗以及连接更多 IP 组件优点，因此它被认为是更加理想的大规模多核处理器互连技术。麻省理工学院研究的 RAW 处理器、Tilera 公司的 Tile 系列处理器都采用了片上网络结构。

2.4 支持多核的微内核操作系统介绍

随着多核处理器技术的发展和对嵌入式实时微内核操作系统的进一步研究，在操作系统产品中出现了一些支持多核处理器的微内核操作系统，其中比较典型的有 QNX、VxWorks、RTEMS 以及被修改的支持多核的μC/OS。这些微内核操作系统各有优势，同时存在着许多的不足之处，本节简要介绍这些支持多核的嵌入式微内核操作系统。

2.4.1 QNX

在上世纪八十年代，Gordon Bell 和 Dan Dodge 创立的 Quantum Software Systems 公司研发了一款能够在 IBM PC 上运行的操作系统 QUNIX，后来改名为 QNX。QNX 是一个几乎支持所有多核处理器、遵守 POSIX 标准、支持多任务的、抢占式的微内核操作系统[21]。QNX 允许每个驱动程序，应用程序，文件系统和协议栈模块运行在自己的私有地址空间。一旦有某个组件发生错误，系统可以自己停止运行这些出错误模块并重新启动，不影响整个系统的运行[21]。QNX 的消息传递机制通过“软总线”机制很迅速地在系统上添加新的功能以及删除不需要的功能。QNX 系统具有与分布式系统中的其他系统进行通信的网络功能，并提供了监控核间通信工具，应用 profile 工具等开发多核的工具。

QNX 所采用的这些机制使得它的运行速度非常快，同时，QNX 和 Unix 系统一样，不会存在计算机病毒。QNX 的用户权限管理也非常严格，任何未正确登录的用户是无法访问系统的，这就不存在系统的文件资料被病毒破坏的危险，很大程度上提高了系统的安全性，而这些特点也是其他的操作系统所不具备的。

QNX 给用户带来了前所未有的稳定性、安全性以及良好的兼容性。QNX 优越的性能使得它被广泛应用在医用、航天、互联网、控制系统等领域，尤其在汽车行业市场份额达 75%以上，在手机通信行业，黑莓手机也开始采用 QNX 系统作为其手机系统。由此可见，QNX 是目前最成功的微内核操作系统。

2.4.2 VxWorks

1983 年，Wind River System 公司设计开发的 VxWorks 操作系统是一个支持各种硬件环境，具有微内核与可裁剪的组织结构，能快速切换多任务，并利用抢占策略进行任

务调度的实时嵌入式操作系统[22]。它可以支持多个内核上任务和进程进行通信以及同步操作，达到提高并行处理事务的能力。VxWorks 凭借高性能的内核、良好的可靠性和实时性以及友好的用户开发环境，被广泛地应用在军事设备、航空航天等领域中。VxWorks 的抢占式微内核结构，在保证实时性同时，也易于裁剪和扩展系统功能模块。但 VxWorks 也存在许多缺点，特别是在任务间通信策略上存在局限性，不能实现消息的暂存、异步传输等，在一定程度上，使得多任务间的同步受到限制。为此，研究人员也对 VxWorks 的通信机制进行了进一步改进，出现了比如说邮箱通信机制等。

2.4.3 RTEMS

RTEMS(Real Time Executive for Multiproeessor systems)是一个开源的无版税的实时多处理器系统。早期它被用于美国国防系统，支持导弹发射和防护系统，后来也被运用在实时的军用系统中。RTEMS 版本的维护以及升级是由 OAR 公司负责，这样将会促使 RTEMS 的性能不断得到改善。RTEMS 支持硬实时和软实、单调周期调度，具有稳定性高、运行速度快的特点。RTEMS 不但支持包括传统的紧祸合系统与松祸合系统的多处理器系统，还支持混合祸合系统、混合异构和异构系统。由于 RTMES 对于底层通信进行了屏蔽，从而使整个多处理器系统的软硬件在逻辑上表现为一个整体。目前，RTEMS 不仅被应用在军工、航空航天领域，也逐渐地被应用在民用领域。

2.4.4 改进的 μC/OS

μC/OS 是一个基本由 C 语言编写的开源实时操作系统，它主要是针对嵌入式应用设计的。μC/OS 的核心代码短小精悍，占用空间小，具有较高的执行效率。它的硬件相关部分采用汇编语言编写的，容易移植到其他的处理器上运行，其移植方法也是比较容易学会[23]。μC/OS 的核心部分包括系统初始化、中断进出的前导、时钟节拍等，其全部代码量也就是几千行。在 μC/OS 内核中，不同的任务使用自己的堆栈空间，堆栈的大小都是根据每个任务所需要的最大堆栈深度来分配的，比较灵活。然而，虽然传统的 μC/OS 只是一个不支持多处理器环境的实时操作系统，但是通过一些研究人员进行适当的改造，支持非对称多处理器环境的 μC/OS 就诞生了。改进的 μC/OS 可以移植到不同的内核上，采用简单的中断和共享内存通信机制，从而能在多核处理器系统平台上运行。

2.5 本章小结

本章阐述了多核处理器的概念以及其体系结构，分析了总线共享结构、交叉开关互连、片上网络等三种典型的多核处理器通信架构。随后介绍了微内核操作系统的基本概

念以及其发展历程，分析其特点以及在可靠性、实时性和安全性方面的优势，并简单介绍了第一代操作系统 Mach、第二代微内核操作系统 L4 以及第三代微内核操作系统 seL4。在此基础之上，本章又介绍了支持多核的微内核操作系统 QNX、VxWorks、RTEMS 以及 μC/OS，分析了各自的优缺点，为后续章节研究支持多核的微内核操作系统通信机制提供了理论基础。

第 3 章 支持多核的嵌入式微内核操作系统通信机制的研究与分析

前一章大篇幅对多核处理器和微内核操作系统进行分析，由此得出多核处理器将逐渐变成微内核操作系统的应用平台。微内核操作系统从第一代发展到如今，影响系统性能提高的关键因素之一是通信机制。换句话来说，在面向多核的微内核操作系统中，通信效率的高低直接决定了微内核操作系统的性能。不管单核操作系统还是支持多核的操作系统，其通信的主体对象都是进程，只是单核操作系统的两个进程处在同一个核心上，而支持多核的操作系统的两个进程处于不同的核心上。因此，我们要设计一种既能实现微内核操作系统在多核处理器的架构上的进程间高效通信，同时要保证系统的可靠性。但是在设计这种支持多核的微内核操作系统的通信机制之前，我们必须要了解多核处理器下嵌入式实时微内核操作系统通信机制的基本原理。本章将系统的介绍多核处理器的进程间通信原理，阐述多核处理器环境中资源竞争特点，分析几种常用的通信方式，并对如何设计一种高效的支持多核的微内核操作系统通信机制进行了讨论。

3.1 进程间通信基本原理

无论是对于微内核操作系统来说，还是其他的操作系统，进程和信号几乎控制全部活动执行。进程是指一个正在执行的具有独立功能的程序，是构成操作系统的基础。进程具有自己的私有地址空间，它们既无法直接访问其他进程的地址空间和资源，也不允许其他进程访问自己的私有地址空间和资源。进程与进程之间的信息传递称为进程间的通信。在操作系统中，进程间常常通过数据交换进行通信。

操作系统的协同工作也是由 IPC 通信方式来实现的，并且在同一操作系统存在着多种通信方式，如消息队列、信号量和共享内存等。为了提高系统的整体性能，我们需要对多核处理器的微内核操作系统的进程基本原理、通信过程做深入地研究。

3.1.1 进程原理

与程序相比较，进程是发生在程序执行过程中，是一个动态的过程，而程序是一段静止的指令集合[24]。程序是一个没有生命的实体，只有处理器处理程序时，它才能成为一个活动的具有私有地址空间的实体。进程是人们抽象出来的一个概念，它主要由程序、数据及进程控制块（PCB）组成。一个进程有唯一标识进程的进程控制块，这样操作系统就可以根据这个标识知道进程的存在，然后可以对其进行管理和控制。正文段是用于共享的重入码，而数据段是进程运行时所使用的数据。如图 3.1 所示，进程有三种基本状态，即就绪状态、运行状态和阻塞状态。

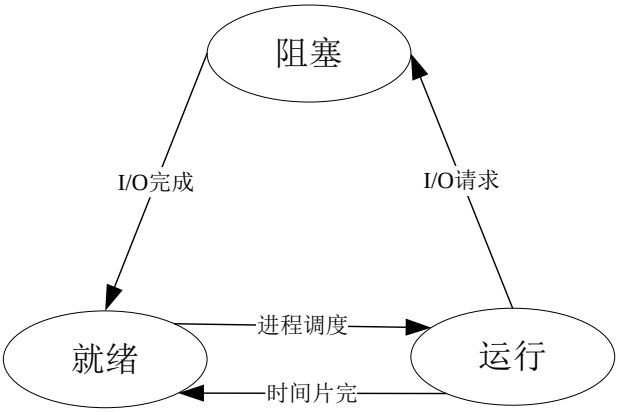

图 3.1 进程基本状态转换示意图

当进程已获得 I/O 等资源，已经进入就绪状态，此时只等待分配 CPU 资源。就绪状态有优先级的任务调度，就必须使用优先级队列来进行划分，只要获得 CPU 资源，进程就可以执行。进入运行状态的进程数目必定不大于处理器的数目，如果没有其他进程执行时，系统会自动执行空闲进程。当运行的进程需要等待某种条件，系统会阻塞进程，使其进入阻塞状态。系统通常在执行多进程时，子进程退出后要清除它的相关信息，如果一个进程在退出后未清除进程表中相应的信息，那么其父进程在查找子进程时，调用不存在的子进程，造成进程僵死。每个僵死的进程都会占用一定的资源，如果服务器不断运行，僵死进程会越来越多，耗费了大量的资源。当资源被耗尽时，系统就无法继续运行了。

每个进程都有被创建、执行和消亡的一个过程。当进程被创建时，操作系统为进程分配唯一的标识符和控制块，它记录了进程的进程标识符、运行状态、现场区及优先级等属性。当进程获得所有需要的资源后，就会执行。如果系统发生要求终止进程的事件后，系统根据被终止进程的标识符，从进程控制块集合中检索出此进程的 PCB，从中读取进程状态，结束被终止进程的资源占有权，归还给其父进程或系统，并从所在的队列中移出。

从前文可知，多核进程间通信是指在多核架构上两个以上的进程之间进行通信交互。进程间通信机制按照接收状态的不同可以分为同步和异步两种[26]。同步通信是指某一个进程在发送消息之后，另一个相应的进程立即执行并接收消息。如果通信的双方进程中任何一个进程没有获得运行的必须资源，都会使这个通信不能继续，进而导致系统的实时性降低。异步通信是指发送消息的进程不用考虑接收消息的进程是否立即接收消息。简单来说，就是发送消息的进程只需发送消息到内存或已经建立好的缓冲区中，直到接收消息的进程获得 CPU 资源之后，就会从相应的消息存放内存或者缓冲区中获得消息，从而完成整个通信过程。考虑到同步与异步通信的优势和缺点，通常许多操作系统会结合这两种方式进行通信[27]。

3.1.2 进程通信的两个阶段

不管是微内核操作系统还是其他操作系统，进程间的通信通常都分为发送和接收这两个阶段。

3.1.2.1 消息发送阶段

在系统进程进行通信时，发送进程一端在应用层发起通信需求，于是系统调用消息传递层的库函数，通过库函数提供通信原语来完成一次消息发送[25]。在这个阶段，为了做好消息发送的准备工作，系统先将应用层中需要发送的数据按照消息传递层提供的消息格式进行格式组装，形成待发送消息，然后将这种格式的待发送消息从消息层可见存储区域发送到硬件抽象层的可见区域，实现两个层的待发送消息控制权转移。接着硬件抽象层将消息通过网络接口转移到通信网络上，最后由网络接口遵循网络通信协议将消息封装成数据包发送到通信网络中。消息层可见存储区属于用户空间，而硬件抽象层属于系统空间，这意味着实现这两个层的转移，也就实现了数据从用户空间到系统空间的转移。在这过程中一般存在着大量的消息拷贝延迟问题需要去解决，为了减少这些拷贝延迟，有些研究者提出取消用户空间和系统空间区别，用户级直接访问网络接口等方案，这些方案虽然很大程度上减少了通信延迟，但是也存在不少问题，比如说移植性问题。

3.1.2.2 消息接收阶段

在系统通信过程中，消息被发送后，进程通信的接收进程一端开始准备接收消息。在消息接收阶段，接收端的目的节点网络接口接收等待接收的数据包，然后硬件抽象层将数据包从网络接口转移到系统的本地存储器，其中网络接口扮演子系统的外设角色，它和处理器的通信方式直接控制着接收端的传输模式。在接收进程端，可能存在有很多条消息同时在等待接收，系统要根据消息的优先级进行接收，这将引起通信的延迟和对资源的竞争，从而影响通信效率。

3.2 多核处理器环境中资源竞争

在单核的微内核操作系统中，对进程间通信效率产生巨大影响的不仅是嵌入式实时微内核操作系统的进程间通信机制本身的机制，还有嵌入式实时微内核操作系统的进程调度策略。进程间进行通信过程中，通常在消息的发送和接收阶段需要经过系统的两次调用以及两次消息复制过程，这都影响着嵌入式实时微内核操作系统的进程间通信效率。因此，许多研究人员从这两个方面进行改进，以减少系统调用次数，同时提高消息复制的效率。在多核的嵌入式微内核操作系统中，核间通信每个内核都有自己的 CPU 等资源，进程间调用都由不同内核控制，所以进程间的调用对通信的影响不大，而在同

一内核的进程通信和单核一样，也会有发送和接收消息时对系统资源进行竞争的情况，进程调用也会对系统性能有所影响。

在操作系统中，在执行多道程序时，需要共享系统资源，导致出现竞争资源的情况。多核处理器的微内核操作系统中资源的分配和共享策略是进程通信设计中的一个关键技术，直接影响通信的效率和复杂性。如果为每个进程分配独立的私有资源，相互之间不共享。虽然这样能以简单的方式实现资源管理，避免饿死和死锁现象，但是会导致资源利用率不高，不能满足进程资源动态变化的需求[28]。比较理想的方法是动态共享所有系统资源，并根据进程在不同环境、不同阶段中对资源的不同需求进行资源分配。该方法的优势是有比较高的效率，适应进程动态变化的需求，但是存在管理复杂、性能降低等问题。在实际设计中，通常为每个进程分配了私有资源和共享资源，使系统在相互矛盾的各项性能之间折中。在多核系统中，每个处理节点一般都有自己的操作系统模型结构。对于共享资源，由于没有一个统一的操作系统模型对其进行访问控制，容易会出现竞争现象[29]。许多操作系统为了解决资源访问控制，采用 PV 操作机制。当进程申请访问共享资源时，系统就审核申请节点能否拥有访问权限，如果获得准入访问的回复时，进程开始启动共享资源的准备工作，并禁止后续访问申请进程进入。如果没有更高的优先级后续访问进程，那么这个进程直到不再使用共享资源，就自动释放其控制权。

3.3 典型通信方式分析

3.3.1 管道和命名管道

管道是具有亲缘关系的进程间常用的通信方式之一[30]，它使用单向的数据流进行通信，消息发送进程将数据写入管道，那么接收消息的进程就按先进先出的顺序读取来自管道的数据。管道是一种类似打开的文件，在系统中无法看到其具体的表现形式，比较适合生产者/消费者关系的交互模式。当进程被创建的时候，系统分配一个指向管道缓冲区的数据页面的索引节点，内核将管道缓冲区当做环形缓冲区使用。每个管道保留了一个特定的管道缓冲区，用来存储两个进程之间传递的数据。当需要创建新的管道时，系统分配两个文件描述符给管道，用 pipe()系统调用来实现，它会返回一些文件描述符，进程再通过 fork()系统调用将文件描述符传给它的子进程，那么它的子进程也可以使用这个共享管道。这时进程使用 read()调用可以从管道中读取信息，同时也可以用 write()调用向管道中写入数据。命名管道具有管道的所有功能，并且允许无亲缘关系的进程间进行信息交互，攻克了管道没有标识的难题[31]。

虽然管道通信机制具有方便、简单、灵活等特点，但是其缺点也十分明显，它要求通信进程双方协商好交互的数据格式，并且要求系统保证进程通信中使用的管道必须存在。如果打开不存在的管道，那么任意两个进程就没法共享一个管道。

3.3.2 信号

信号常被用来处理系统响应某些事件，是一种快捷的异步通信的软件中断[32]，它一般被一个进程用来通知另一个进程已经发生了一个事件，也可以把信号发送给进程本身。异步性是信号通信的一个重要特点，这使得进程在运行过程中随时都可能收到信号，不需其他操作来等待接收信号。系统中的信号可以同时发给一个或多个通信进程，每个信号都有一个唯一的标识，它通常都是一个常数。信号的主要目的是通知特定的事件要发生，而不是传送数据，当进程不需要此信息，便可以忽略这个信号。一旦用户进程接收到信号后，就必须执行特定的信号处理程序。当用户不希望程序马上执行时，信号可以延迟传送给进程，那么这时可以先阻塞信号。然而信号通信过程是比较复杂的，交换的信息是比较短的，不适合长度过大的消息传递。

3.3.3 消息队列

消息队列是一种用链接表结构实现消息传递的通信方式[33]，适应于任何进程间通信，克服了信号承载信息量少等缺陷。消息队列初始化时，系统在内核地址上开辟一个空间，这个空间用来存储消息队列，可以让有添加消息权限的进程向队列中增加消息，让具有读权限的进程可以读取队列中的信息。并建立一个进程列表，这个进程列表中包含所有等待接收消息的进程，当有进程需要接收消息的时候，这个进程就会被添加到这个进程列表中。有消息进入到消息队列时，在这个进程列表中，优先级最高的进程会获得这个消息，或是这个进程列表中最先等待的进程将会得到这个消息。当一个进程需要向另一个进程发送消息时，发送消息的进程就会向接收消息的进程发送一个变量，这个变量实际上是一个指针，这个指针指向的内容就是需要被传递的数据。由于被传递的消息的内容不同，所以指针变量所指向的内容也是不同的数据结构。

3.3.4 共享内存

共享内存是一种通过访问同一个逻辑内存区进行通信的方式。这种通信方式可以传递数据量很大的消息，通过信号量等传递一个令牌告知其他进程共享内存的数据已经可以使用。系统使用共享内存进行通信，需要经历创建、连接、访问和释放几个阶段[34]。系统调用创建函数并通过传入申请共享内存空间大小等接口参数，创建共享内存。系统在访问共享内存前锁定信号量，访问后进行解锁信号量操作。通过调用连接函数实现将共享内存连接到一个通信进程上，有些系统将共享内存作为虚拟内存区来实现，这些虚拟存储区会被映射到物理内存区域。像访问虚拟地址空间里的其他任何内存一样，进程访问共享内存。最后，系统通过调用释放函数实现释放共享内存。

在多个相互关联的进程通信中，它们通信的数据通常包括配置数据、进程管理数据

和工作交换数据，并通过信号量 ID 来控制访问，其共享内存数据存储的结构可以用图 3.2 来表示。

图 3.2 共享内存数据存储结构示意图

除了工作交换数据存储区的访问比较频繁并且数据量比较大之外，其他的数据存储区的访问和数据量一般不是很频繁。共享内存区通常只允许进行数据的读写，在共享内存区之外对数据进行处理，从而减少系统对其操作所占用的时间。共享内存允许多个进程访问，并可以指定共享区域数据空间大小及自定义数据区的结构。但是相对其信号等通信方式，共享内存是比较慢的一种通信方式。

3.3.5 信号量

信号量是用于解决进程对共享资源的抢夺问题的一种特殊的数据结构[35]，它是荷兰人 Edsger Dijkstra 在 1965 年提出来的。信号量使一个系统资源在同一时间内只能被一个进程所占有、利用，其他的进程想要获得这个资源的进程必须等待现有进程运行完毕或者放弃使用这种资源之后，才能够有机会获得这个资源的使用权。信号量主要被用于处理两种情形，一种是互斥情况，其目的是为了防止多个进程在访问同一共享资源时发生数据不一致性的，通常采用 PV 操作原语来实现的。另一种是同步情况，又称生产者与消费者问题，当消费者进程开始执行时，它会执行一个 P 操作，表示此进程要进入临界区域。当生产者进程生发送数据后，将执行 V 操作，此时，消费进程可以处理这些数据，这使得请求共享资源的进程所获得数据信息就是正确无误的。

3.3.6 套接字

套接字（Socket），又称套接口，是一种非常有用的机制，它和前面的几种通信方式不一样，前面的都是在单台计算机上通信，而套接字方式可以在多台计算机之间进行通信。它是一种抽象的数据结构[36]，当它被创建后，就被映射到一个本地地址上，并监听客户端的请求。套接字又可以分为流套接字和数据文报套接字这两种类型[37]，其中流套接字是以字节流传输信息，而数据文报套接字是在数据文报的独立单元中传输信息。进

程利用套接字进行通信时，发送信息进程将一段信息写入套接字中，此套接字将这段信息发送到另一个进程的套接字中，从而完成进程间的通信。套接字通信很好地解决了多个计算机及多个进程之间的通信问题，为计算机之间的信息交互和资源共享提供了有利的条件。

3.4 嵌入式微内核操作系统通信机制分析

3.4.1 L4 的进程间通信机制

为了提高进程间通信的性能，减少消息的复制次数，L4 通过通讯窗口机制，实现进程间的直接通信。通讯窗口是指存在于内核地址空间的一段内存区域，内核通过把要接收消息的进程的地址映射到通讯窗口，完成消息的传递。L4 的消息传递过程如图 3.3 所示。

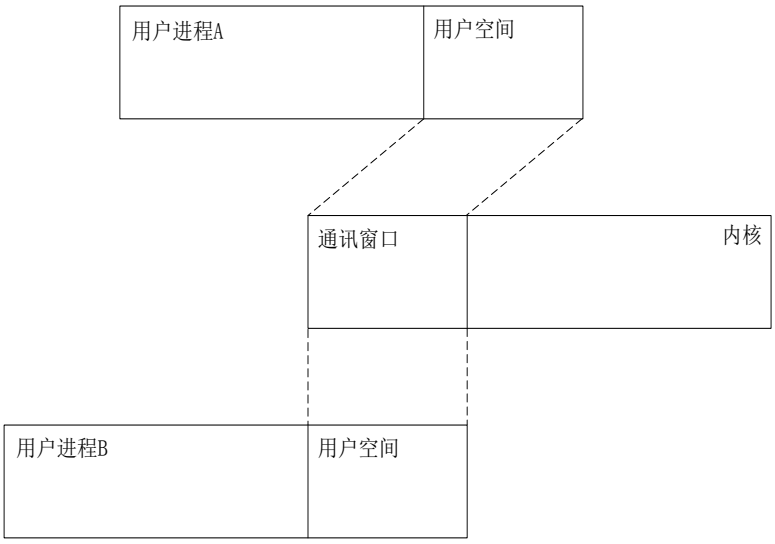

图 3.3　L4 的消息复制过程

当进程 A 向进程 B 发送消息时，进程 B 的地址被映射到存在于内核空间的通讯窗口中。然后，进程 A 把要发送的数据复制到通讯窗口，这样，进程 B 就可以直接使用这些数据，同时这些数据只能够被进程 B 访问、使用。

L4 的这种机制使进程间进行通信时仅仅需要通过一次的复制，这大大减少了进程间通信对系统的消耗，提高了系统的效率。但是，这种机制使得内核可以进入到任何用户进程的用户空间，影响了系统的安全性[38-39]。

3.3.2 Minix3 的进程间通信机制

在进程调度方面，Minix3 把每个进程都在物理内存中进行定位，把它们放在地址空间的对应位置[40]。同时，把内核编译到物理内存的最低端，整个系统的物理内存对内核来说都是可见的[41]。由于所有进程都被定位在物理内存，所以可以说所有进程的内存空间对内核来说都是可见的。当进程间进行通信时，内核可以通过把发送进程的消息复制到接收进程的地址空间来完成整个通信过程。如图 3.4 所示。

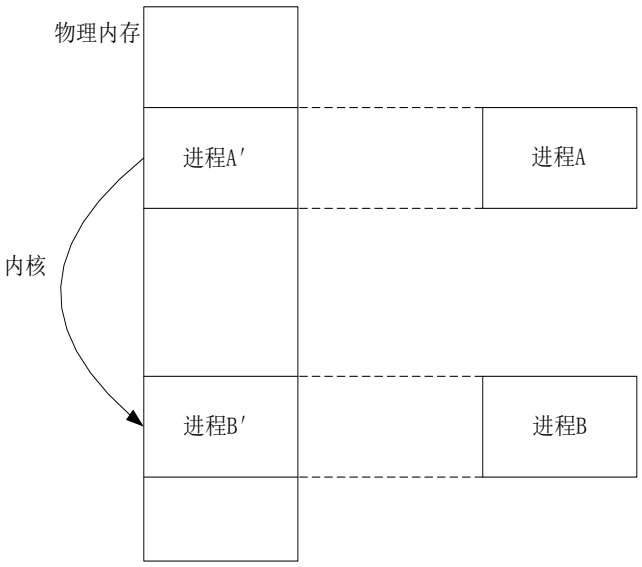

图 3.4　Minix3 的进程间通信

进程 A 和进程 B 分别在物理内存中重定位到物理内存中的进程 A′ 和进程 B′ 。当进程 A 发消息给进程 B 时，内核只需要把消息从进程 A′ 复制到进程 B′ 即可。

但是，这种方法是基于 Minix3 的分段机制的，如果采用分页机制作为内存的管理策略，那么这种通过直接复制实现进程间通信的方法是无法实现的[42]。

3.4.3 QNX 的进程间通信机制

QNX 的消息传递机制支持多核的核间消息传输。发送和接收消息的进程都拥有一个 MX 表，MX 表可以指示消息在内存中的位置。这就允许消息拥有一个从数据块中分离出来的数据头，发送的时候可认为不是一个连续的消息，这样就在复制过程中不会消耗系统资源。如果底层数据结构是缓冲区，一个分为头和两个连接块的三个部门的消息就可以被作为一个单一的“原子”消息被发送，这种 MX 映射机制不需要发送者和接收者的同步。QNX 微内核操作系统之所以能具有这么高的系统性能，主要是得益于它所采用的这种多部消息传递机制，它把一个消息分割成多个不同的部分进行传递。此机制的优点在于：

（1）多部消息传递机制会把一个长消息分割成多个短消息进行传递，不需要在内存中建立一个连续的地址空间用来存放需要被传递的消息。消息的发送进程和消息的接收进程都只需要通过消息指示表就能够获得消息。

（2）在一个进程发送消息或者接收消息的时候，这个进程可以直接到缓冲区中读出已经存在的消息，同时只需要读写消息的头部，这种机制避免了在读写消息的时候对于缓冲区的分配问题。

3.4.4 VxWorks 的进程间通信机制

在 VxWorks 操作系统中，常用管道、信号量、消息队列、共享内存、套接字、软信号、远程过程调用等多种方式进行通信。其中软信号被用于处理应用软件的异常，套接字和远程过程调用被用于处理网络通信，信号量被用来实现通信过程中同步和互斥情况，进程间的信息交换就由剩下的方式来实现。在多核处理器环境下，进程间的通信采用了共享内存对象来实现，所有的进程都存在单一的线性地址空间中，不同进程间共享数据，用运行在不同上下文的代码指向全局变量、环形队列、线性队列、指针和链表。VxWorks 操作系统中二进制信号量和计数信号量都可以用来完成进程间同步与互斥，但是计算信号量可被用来记录信号量释放的次数以及监视某一资源的使用情况，其状态图如图 3.5 所示。

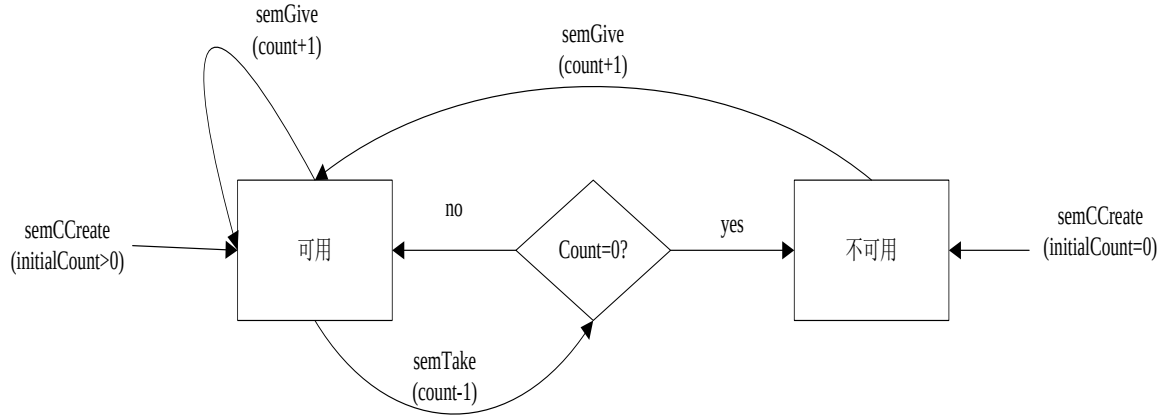

图 3.5 计数信号量状态图

3.4.5 SmartOSEKOS_M 的核间通信机制

从前文可知，SmartOSEKOS_M 是浙江大学开发的一款支持异构多核的嵌入式实时微内核操作系统。像其他面向异构多核处理器的嵌入式领域操作系统一样，也采用共享内存进行通信。但是不同于单核任务间通信和非紧耦合结构的多核处理器系统通信，它通过核间符号表来实现共享内存中数据的相互引用，但是核间符号表不是为了任务间的大量数据进行通信而创建的。SmartOSEKOS_M 参照 OSEK COM 标准的外部通信机制，

其核间通信任务模块采用了两个队列以及两种非队列核间通信服务。其实非队列核间通信服务是特殊实现形式的队列通信服务，是指队列中只包含一个节点。

SmartOSEKOS_M 的核间任务通信模块是根据支持异构多核的共享内存模型来实现的，它为多核处理器的多个核分配了用来控制和存储核间通信数据的共享内存区。如图 3.6 所示，消息队列的控制头分别控制一个消息队列，并且有分别指向消息队列的队头和队尾，当然消息队列没有消息时，那么这两个指针都为 Null 值。

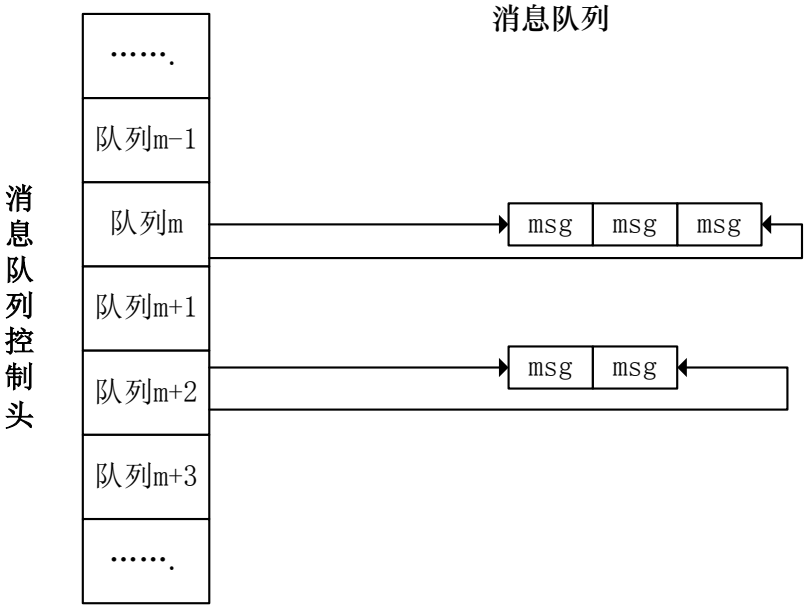

图 3.6 核间消息队列通信结构示意图

3.5 本章小结

首先本章对多核处理器进程间通信基本原理进行了详细阐述，介绍了进程的基本原理，分析了进程通信的两个阶段，并对多核处理器多进程的通信特点进行探讨。然后，就影响通信效率的资源竞争进行描述，并分别对管道、信号、信号量、消息队列、套接字和共享内存的优缺点进行分析。最后，对几种典型的微内核操作系统的通信机制进行了进一步分析和探讨，为下一章的通信设计提供研究基础。

第 4 章 支持多核的嵌入式实时微内核操作系统通信机制的设计

前一章介绍并分析了一些典型的通信方式以及多个具有代表性的嵌入式实时微内核操作系统通信机制，使我们对微内核操作系统的通信机制有了初步的认识。同时，也看到了现有的微内核通信机制难以满足人们的需求，尤其是在多核环境下的通信效率。为了提高支持多核处理器的微内核操作系统通信效率，本章在已有的微内核操作系统通信研究基础上，提出并设计了基于消息分类的线程代理通信总体架构，并从核内通信和核间通信两个角度进行创新和改善，降低通信导致的延迟，进而提高支持多核的微内核操作系统运行性能。

4.1 多核通信机制的设计目标

在多核处理器快速发展的今天，微内核操作系统已经跟不上多核硬件的前进步伐，需要不断地研究和革新，以提高微内核的性能。为了在实际工程应用中发挥出异构多核处理器的最大性能，满足多核环境的需要，本通信机制研究的主要目标是设计出一个高性能的微内核操作系统通信机制，其具体达到的目标如下：

（1）继承微内核操作系统可靠特点。本章在对基于消息分类的线程代理的微内核操作系统通信机制进行设计时，最重要的目是充分利用微内核和多核处理器的高性能，使支持多核处理器的微内核操作更加高效和可靠[43]。也就是说，在嵌入式微内核操作系统增加了对多核处理器的支持之后应该比原来的性能更加高效，降低中断响应和进程通信的时间，并且使得微内核操作系统增加了多核的高可靠性。

（2）可扩展性强。多核处理器中有些是双核，有些是三核，甚至更多核[44]。为了使设计的微内核操作系统适用于不同核心数目的处理器，不仅研究出易于扩展的微内核整体架构，也要设计出支持扩展的通信机制。

（3）较好的兼容性。本设计的目的是将微内核操作系统应用在多核处理器上，但是现在单核上的微内核操作系统研究相对比较成熟，所以在设计过程中，考虑到单核上的系统不需要做很多的修改就可以适应多核处理器架构上。

（4）低复杂度。嵌入式微内核操作系统最显著的一个特点是它拥有体积容量小的内核。在设计支持多核的操作系统通信机制中不可避免地会增加其复杂度和代码量。为了秉承微内核的设计理念，微内核操作系统的内核仅提供实现操作系统的相关服务，并在兼顾其他功能的基础上，尽量降低通信机制的复杂度。

（5）实现核间的快速通信。多核与单核环境下通信最大的区别是多核环境下通信双方进程可能处于不同的核心，它们之间的通信需要进行核间通信，由于它们不像核内

通信那样可以共享寄存器，所以通信机制也不同也没有那么快速，所以在设计时必须尽量地提高其核间通信效率。

4.2 多核通信的特点

与单核的通信机制不同，由于在异构多核架构这种特殊的硬件体系结构下，面向多核的微内核操作系统通信机制具有自己的特点。它也不像分布式操作系统那样，通过消息传递网络或远程过程调用等方式进行通信。面向异构多核处理器的微内核操作系统，每个微内核控制一个处理器的核心，各个内核实现着不同的功能，它们之间存在着主从关系，又相对独立，拥有私有的资源。因此，在设计这种支持异构多核的微内核操作系统通信机制时，就需要充分考虑如下的多核环境通信的特点。

（1）在单核中进程间通信的双方都位于同一个内核中，它们共享同一个内核中的 CPU 资源，不可避免会对 CPU 资源进行争夺。而多核间的进程通信双方位于不同的内核中，因为它们分别拥有不同的资源，不会对 CPU 资源进行抢夺。

（2）单核内部可以通过全局变量的方法实现进程间的通信过程，但是多核间无法利用这种方法。因为，一个内核具有的变量对另一个内核来说是不可见的。

（3）在单核中可以通过寄存器的方式实现，这种方法效率很高，但是在多核架构下，进程不允许直接访问不同内核的寄存器，内核之间的通信就无法通过寄存器的方式实现。

（4）虽然支持异构多核的微内核操作系统要求有高效的多核核间通信机制，但是涉及到两个及两个以上内核之间的进程通信的情况相对较少，大多数情况是在同一个内核内部进行通信[45]。那么为了实现这种小概率的多核间通信，而对现有的核内的进程通信机制进行大规模的修改，就会增大核内通信设计的复杂度，加大整个嵌入式微内核操作系统的研发周期。

（5）各内核之间具有相对的独立性。通过参照多核结构的良好扩展性的低级硬件抽象模型中的硬件抽象层[46]，设计的核对核式嵌入式微内核操作系统，即一个微内核控制一个处理器核心，并且每个内核实现不同功能，其结构如图 4.1 所示。此体系结构模型运用了单核操作系统的抽象层，只需提供较少的原语，精简了内核功能。本文是基于这样的异构多核的微内核操作系统体系研究基础上设计的通信机制，所以必须考虑到多核操作系统中各核功能的独立性特点对进程间通信的影响。

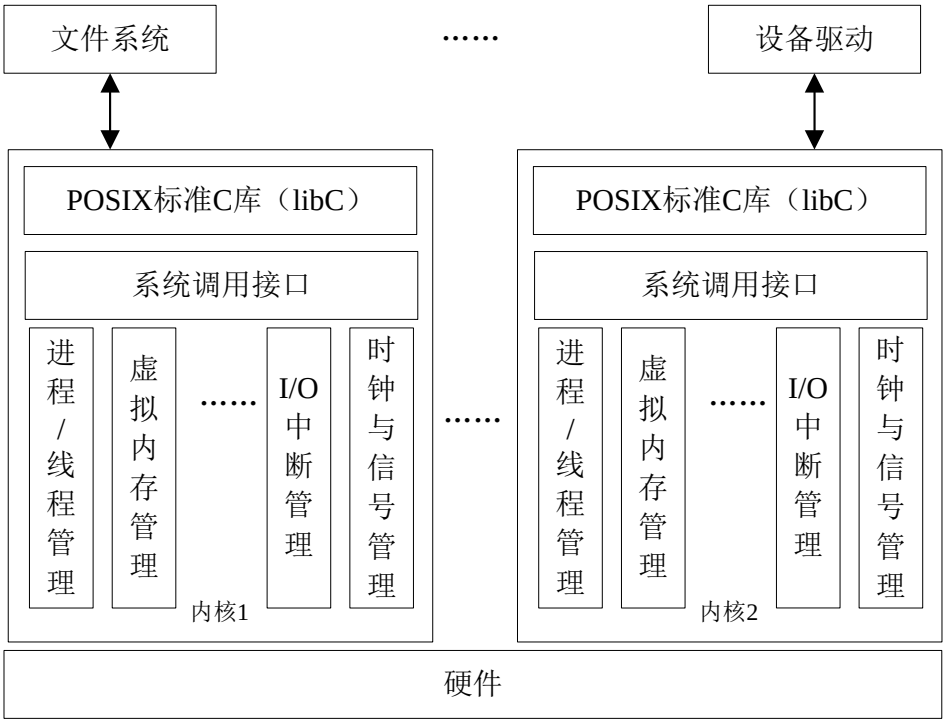

图 4.1 核对核式微内核嵌入式操作系统体系结构设计图

（6）在异构多核的微内核操作系统中要求在规定的条件和限定的时间内完成请求的相应功能，解决进程优先级、任务死锁等问题[47]。为此在微内核操作系统通信机制设计中，必须采取对应的策略，提高支持异构多核的微内核操作系统的可靠性，保障系统的稳定无误地运行。

4.3 基于消息分类的线程代理通信机制总体架构

本章研究的通信机制是针对于异构多核的微内核操作系统，在这个操作系统中每个微内核控制相应的一个核心，该微内核和核心组成相对独立的单元，实现不同的功能。前文提及过，无论是在异构多核环境还是单核环境中，微内核操作系统都是以进程为对象进行通信。如果在多核环境中使用总线结构进行通信，虽然总线可被多用户交互使用，但一条总线在某一时刻只能在两个用户间使用，其余的必须等待，等待机制导致了通信的瓶颈[48]。如果利用网络拓扑结构进行通信，虽然提供了良好的并行通信能力和可扩展性，从而使得通信带宽大幅度增加。但是这种片上通信消耗系统大量资源，庞大的时钟消耗与总线的功耗将占据芯片总功耗的绝大部分，再加上机制的复杂性和通信协议，所以不适合微内核操作系统[49]。如果由主内核直接为每个要通信的内核创建代理线程，那么核间消息可以通过主内核的寄存器可以进行高速通信，但是要通信的内核与主内核也是不同的内核，这样不可避免地又需要进行核间通信，增加了通信的时间。因此，本文

通过对目前国际上领先的嵌入式操作系统微内核设计理论方法进行详细的分析基础上，结合多核处理器架构共享 L2Cache 和内存的特点，以及微内核操作系统的基本原理，设计多核环境下基于消息分类的线程代理通信机制架构如图 4.2 所示。

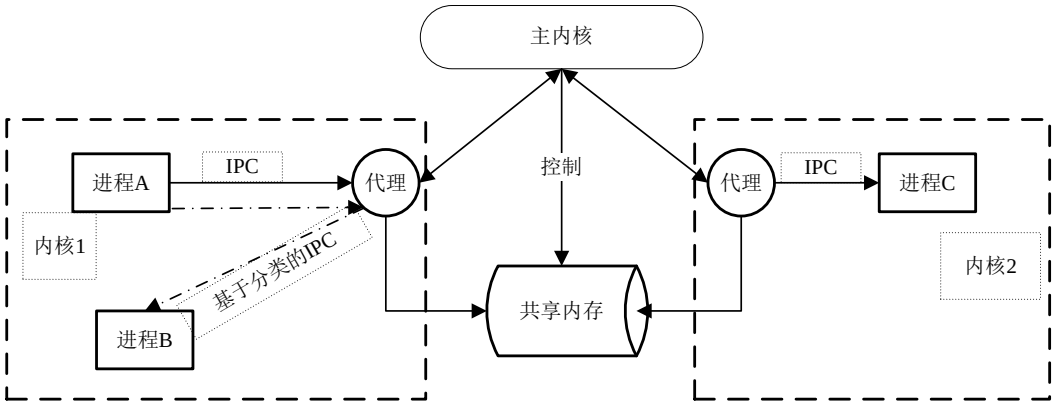

图 4.2 多核处理器架构下基于消息分类的线程代理通信架构

为了简化通信机制设计的复杂度，本文先从在主内核和通信的两个内核组成的模型进行研究，再以此扩展到具有更多内核的微内核操作系统中。在图 4.2 所示的通信架构中，系统采用了一个主内核，它被用来控制和协助核间通信，而且在除了主内核外，其他的内核都采用轻量级代理线程进行通信。在图中当核 1 中进程 A 需要与核 2 中的进程 C 进行通信时，进程 A 只需给同核中代理线程处理，代理线程把消息复制到共享内存并同时通知主内核，主内核根据维护的查找表，判断这个进程是哪个核上的进程，把信息的共享地址告知相应核的代理线程，代理线程把地址给进程 C，C 就可以去取消息了。如果是核 1 中进程 A 与同核进程 B 通信，进程 A 发信息给代理线程，代理线程判定属于同核，就会根据消息的长短不同，进行分类通信。接下来，本文从核内和核间通信这两个角度详细阐述具体的通信机制。

4.4 核内进程通信设计

4.4.1 基于共享内存的地址映射通信

在微内核操作系统中每个进程都有地址的页表信息[50]，这些信息存储在内核的地址空间中。在进程通信过程中，可以通过这些页表信息访问其他进程的地址空间，进而获取其保存的数据信息。由此可知，每个进程的都有存储在内核中的映射表，通过这个映射表可以间接地查看进程的地址空间全部信息。在基于共享内存的映射通信机制中，接收进程的消息缓存区通过地址映射表映射到内核空间的一段内存区域，发送进程要与这个接收进程进行通信，只需要把发送的消息复制到内核相对应的缓存中，接收进程就可以直接获得消息，从而完成通信过程。因为内核的缓存物理地址和接收进程的消息缓存

物理地址形成映射关系，发送进程把消息复制到内核缓存区相当于间接地复制到了接收进程的缓存区，减少通信的二次拷贝，降低了通信延迟。

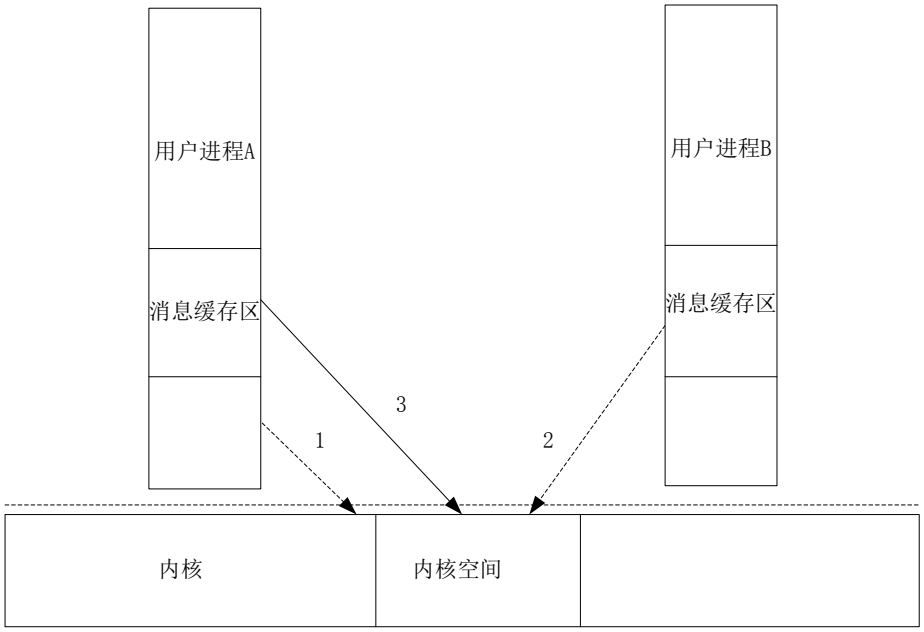

图 4.3 基于共享内存的地址映射机制示意图

如图 4.3 所示，基于内存共享的地址映射通信机制的消息传递过程包括如下三个主要步骤。

第一步：当进程 A 要和进程 B 进行通信时，进程 A 发送通信请求信息到内核的进程消息管理表中，等待内核的处理。

第二步：内核在接收到进程 A 的通信请求之后，根据进程表分析其发送的目的进程，得知进程 B 是需要接收消息的进程，就将进程 B 的缓存地址空间和存在于内核的消息缓冲区地址空间进行映射。

第三步：当进程 B 和内核空间已经建立好映射关系之后，进程 A 把要发送的消息内容复制到内核中的缓冲区中，实际上就是把进程 A 发送的消息内容直接传递给了进程 B，完成了整个进程间通信过程。

与此同时，基于共享内存的进程间地址映射通信机制采用了拷贝技术，这个技术的好处是页面通常以内存只读方式存在[51]，只有进程 B 需要写入时，才会复制消息，避免了设置消息缓冲，进而减少了系统的开销。

4.4.2 基于通用寄存器的进程通信

众所周知，寄存器是 CPU 的重要组成部分，它是高速的存储部件，但其存储容量极为有限。它是最顶端的内存阶层，通常被用来暂存指令、位址和数据。基于通用寄存器的进程通信机制利用 CPU 的这些寄存器来进行消息高速传递[52]。当发送消息的进程

把消息存储在寄存器中，同时产生一个中断通知系统，系统调度相应的接收进程。消息接收进行开始执行，把消息从寄存器中读取出来，这样就完成了一次消息传递。像这样把消息直接放入到 CPU 的寄存器中来实现进程间的消息传递，不仅使消息的传递过程简化而且大大地提高了进程间通信效率。

4.4.3 基于消息分类的核内通信策略设计

在现有的单核微内核通信研究基础上，设计并优化同一内核的不同进程的通信。为了充分利用基于共享内存的地址映射 IPC 与基于通用寄存器 IPC 的优点，本文对于不同长度的消息，分别采用不同的通信方式。如果需要传递信息是短消息，通信机制采用通用寄存器来存放发送的消息内容。通过寄存器传递短消息时，发送进程只需要简单的陷入微内核即可，然后微内核直接进行进程间切换，把消息缓存的寄存器切换到目标进程，目标进程就开始执行消息接收，这样就实现了消息的传递。如果需要传递信息是长消息，通信机制采用基于共享内存的地址映射方式进行通信。设计的基于消息分类的一次核内通信流程如图 4.4 所示。

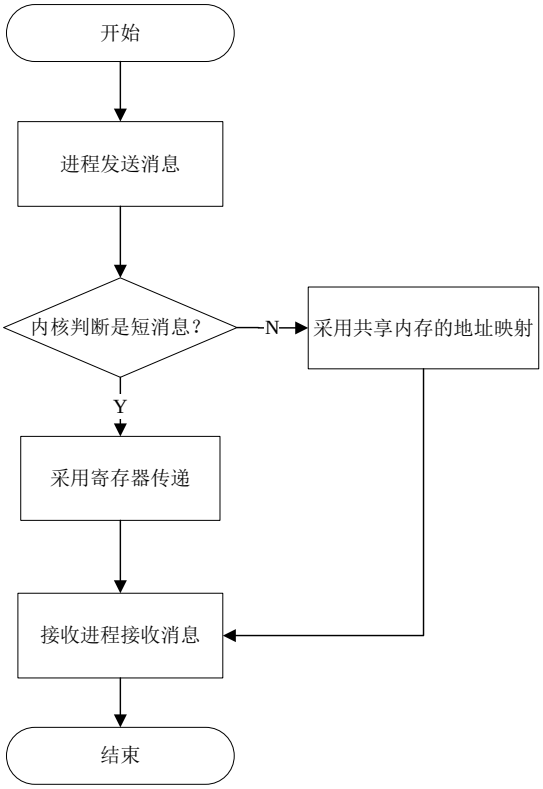

图 4.4 基于消息分类的核内进程通信流程图

在面向异构多核的微内核操作系统核内通信流程图中可以看出，在同一内核的通信双方，发送消息的进程首先发送请求消息给内核创建的通信线程代理，线程代理通过分析消息头的信息判定通信双方进程是否属于同一个内核。对于通信双方都是处于同一个

内核的，再进一步分析消息的长度。如果传递的消息属于短消息，就采用寄存器传递方式进行传递。如果传递的消息是长消息，就通过共享内存的地址映射方式进行通信。最后由接收进程进行消息接收，这样就完成了一次信息通信。

4.4.4 消息长度的划分

上述设计的核内通信机制涉及到长信息和短消息，那么怎么划分这种消息的长短呢？在微内核操作系统分页机制中页表头都有一定容量，本文研究的微内核操作系统设定的容量大小为 4 字节，则对应的页表大小为 1024 字节。如果通信系统采用基于共享内存的地址映射来传递超过页表头信息大小的信息，还不如直接传递消息内容，这样直接传递的通信效率会更快些。

在传统的二次拷贝进程通信过程中，发送消息进程把消息复制到缓存区中，然后相应的接收进程从缓存区中复制消息。整个过程所消耗的时间T_{2copy}为

$$T_{2copy} = t_L + t_L + \alpha \qquad \text{公式（4.1）}$$

其中t_L是消息大小为L的一次拷贝时间，α为等待中断等所消耗的时间。

本文设计的基于消息分类的微内核操作系统核内通信机制中，只有一次复制发生在发送进程发送消息到内核空间过程中，相当于传统二次拷贝的一次拷贝，在共享内存的地址映射机制第二步过程中还需要拷贝 4 个字节的消息头所花费的时间，所以基于消息分类的核内通信机制消耗的时间T为

$$T = t_L + t_4 + \alpha \qquad \text{公式（4.2）}$$

其中t_L是消息大小为L的一次拷贝时间，α为等待中断等所消耗的时间，t_4为拷贝 4 个字节的消息头所花费的时间。

如果找到一个值为L_0的消息大小，满足

$$\lim_{L \geq L_0}(t_4 / t_L) \approx 0$$

那么可以得到

$$T = \lim_{L \geq L_0}(t_L + t_4 + \alpha) = t_L + \alpha \qquad \text{公式（4.3）}$$

通过前期研究和不断实验得出L_0为 8 时，基本满足接近一次拷贝的条件。换句话说，在基于消息分类的核内通信机制中，当消息内容长度大于 8 个字节时，采用基于地址映射的消息 IPC 可以实现消息的一次拷贝传递；而对于消息内容小于 8 字节的时可以采用基于通用寄存器的消息传递。这种设计方案的好处是使系统在可变长消息机制中有很好的性能，同时又保证了系统的安全性。

4.5 多核核间通信机制设计

在核之间进行消息通信时，最重要的就是消息在最短的时间内到达目的地并完成信

息交互。考虑到不同内核之间不能共享寄存器，而单核的基于共享内存的地址映射机制已不完全适应多核核间通信。本文设计一种基于主内核控制的线程代理核间通信机制，在该机制中，代理线程把消息复制到共享内存之前通知主内核，主内核根据进程信息查找表，找到相应的接收进程，并把接收进程的缓存地址和共享内存的地址进行映射，然后发送信息进程把消息复制到共享内存中，其具体机制如图 4.5 所示。

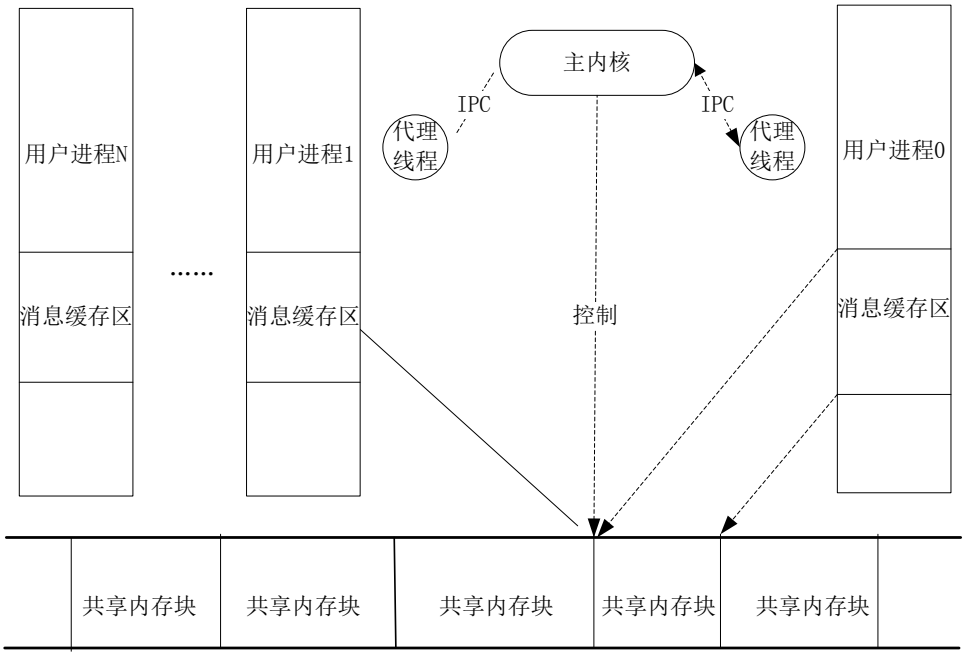

图 4.5 基于消息分类的核内进程通信流程图

在上图中，进程 1 要发送信息给进程 0，先由本核的代理线程进行处理，接着线程代理马上发送信息给主内核，经过主内核查到进程 0 的代理线程，然后由代理线程通知主内核开辟一块共享内存块，并建立进程 0 和共享内存块的映射关系。最后，进程 1 的代理线程把发送的消息复制到这块共享内存块中，完成了一次多核核间通信。完成进程 1 和进程 0 之间的通信之后，如果还有其他进程等待发送消息给进程 0，那么由主内核把进程 0 映射的这块共享内存地址通知下一进程，继续进行下一个通信；如果没有其他进程在等待，就释放进程 0 所占用的资源。在通信中，使用了主内核进行统一管理，协助分配资源，这样由主内核来管理通信，克服了现有的核间通信机制的缺点，提高通信安全性，减少了系统通信延迟。

4.6 通信优先级策略

有些情况下出现某个消息接收进程同时接收到多个进程的请求，就需要研究设计进程通信的优先级。进程通信优先级是指在执行通信过程中，通信的某一方存在有多个进程，这些进程中具有使用临界资源的优先等级[25]。本文研究的通信机制中可能存在多个进程同时发送消息给某一个接收进程，也可能存在某一个进程发送消息给多个接收进

程。对于一个进程发送消息给多个进程这种情况，不会产生对同一共享内存块竞争，但是对于多个进程发送消息到同一接收进程这种情况，就会产生对接收进程消息缓存区映射的共享内存块的竞争。为了更好地适应异构多核处理器中的微内核操作系统进程通信，每个通信进程都分配一个优先级，同时允许某个优先级对应多个通信进程，构建优先级队列，对于同一优先级队列中的通信进程的优先级是相同的。在进程通信时，优先级高的进程可以优先于低优先级的进程进行通信。在多个通信进程进行争夺通信资源时，可能出现优先级反转的情况，即某一低优先级的进程持有高优先级的进程所需要的资源，使得高优先级的进程需要等到低优先级的进程释放资源后才能被执行。为了解决这个问题，有些学者提出优先级继承等方法，但是存在不能避免死锁、阻塞链延迟等缺点。在核间和核内通信都涉及到基于共享内存的地址映射技术，所以设计了通用的微内核通信优先级策略。

支持异构多核的微内核操作系统通信优先级策略与单核系统有许多不同之处，多核处理器中有多个核心，出现进程间对 CPU 资源竞争的情况还是比较少的。因此，本优先级策略吸收了优先级继承的一些设计理念，设置 16 个优先等级，采用优先级和等待时间相结合的方式，避免通信进程死锁现象。具体算法如下：

（1）通信进程申请发送消息给接收进程，如果系统中没有其他进程发送给同一接收进程，直接转入步骤 5。如果系统中有其他进程在等待，就转到步骤 2；

（2）判断该进程与正在与其目的进程的通信进程，如果该进程优先级大，等待一个时钟后，阻塞这个正在执行的通信，转入步骤 3，否则，插入相应的优先级队列，转入步骤 4；

（3）被阻塞的进程赋予最大优先级，然后插入等待队列；

（4）在等待队列中的进程优先级在每隔一段时间，自动提高一个等级；

（5）发送进程获得主内核通信许可，在其协助下把消息复制到共享内存块中，进行通信。

4.7 共享内存管理

共享内存为基于消息分类的线程代理通信机制提供了通信支持。通常接收进程的消息缓存区所映射的共享内存块大小相对 PC 机等大型计算机系统要小得多[54]，而且由于运行的程序数量相对较少，使申请的内存块大小相对比较有规律，这为预测机制在微内核操作系统中成功运用提供条件基础。嵌入式内存主要可以分为动态内存分配和静态内存分配方式[55-57]，如果采用静态分配内存给每个进程，就不符合通信实际情况。而对于精简的微内核操作系统，传统的内存管理分配算法就显得复杂低效。为了提高通信过程中动态分配共享内存效率，设计了一种基于马尔科夫链的预测内存分配算法来配合通信

时系统分配共享内存。

4.7.1 马尔可夫链预测理论

定义 1　设有随机过程$\{X(n), n=0,1,2,\cdots\}$的离散状态空间为$E=\{i_1,i_2,\cdots\}$，若对于任意m个非负数$n_1,n_2,\cdots,n_m,n_{m+1}$ $(0\le n_1<n_2<\cdots<n_m<n_{m+1})$，以及任意满足：

$$P\{X(n_{m+1})=j \mid X(n_1)=i_1, X(n_2)=i_2,\cdots, X(n_m)=i_m\}$$
$$=P\{X(n_{m+1})=j \mid X(n_m)=i_m\} \qquad \text{公式（4.4）}$$

则称随机过程$\{X(n), n=0,1,2,\cdots\}$为马尔可夫链[58]。

如果n_m表示第m次所给随机数，n_{m+1}表示第$m+1$次所给随机数，$n_1,n_2,\cdots,n_{m-1}$分别表示第 1 次到$m-1$次所给随机数。那么从公式中可知，过程在将来的第$m+1$次所处状态j仅依赖第m次所处状态i_m，而与第 1 次到$m-1$次的所处状态$i_1,i_2,\cdots,i_m$无关，该特性称为无后效性[59]。

连续参数马尔可夫链$\{X(n), n=0,1,2,\cdots\}$在随机数s所处状态i的条件下，在$s+n$所处状态j的条件概率，称为该马尔可夫链随机过程中的转移概率。如果马尔可夫链$\{X(n), n=0,1,2,\cdots\}$的转移概率$p_{ij}(s,n)$，只与状态i和j有关，及n有关，而与s无关，即不依赖s的马尔可夫链，则称齐次马尔可夫链。其公式为：

$$p_{ij}(s,n)=P\{X(s+n)=j \mid X(s)=i\}=p_{ij}(n) \qquad \text{公式（4.5）}$$

式中$0\le p_{ij}(n)\le 1, \sum_{j\in E} p_{ij}(n)=1$

在马尔可夫链预测过程中，事物按照某一特点可划分为m个状态$\{i_1,i_2,\cdots,i_m\}$，用$p_{ij}(n)$表示经过n步转移的概率，用来描述相互转移的概率构成的称为概率矩阵。如果由某一时期的状态转移到下一状态的概率，叫做一步转移概率，记为p_{ij}。将所有的概率p_{ij}按顺序进行排列，构成的一步转移概率矩阵:

$$Q=\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ & & \cdots & \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix} \qquad \text{公式（4.6）}$$

满足条件$0\le p_{ij}\le 1, \sum_{j=1}^{m} p_{ij}=1, (i,j=1,2,\cdots,m)$

4.7.2 预测内存分配机制的建立

对于共享内存分配来说，所分配的内存块大小就是马尔可夫预测中的状态，内存大小的集合构成了状态空间$E=\{8b,16b,32b,\cdots\}$。本文在设计马尔可夫预测的内存状态时，采用了聚类分析的思想，所谓聚类分析是指将某事物的集合分组成为由某一特征类似的对象组成的多个子类的分析过程[60]。通过这种聚类的思想把申请的内存块大小按系统提

供的满足的最小内存。如图 4.6，虚线框内表示的是嵌入式内存申请内存大小的队列顺序，另一边表示的是连续两次的转移状态和转移的统计量。当程序申请 48b 大小的内存块时，嵌入式系统会分配一块大小为 64b 大小的内存块，也就是说把申请内存块大小大于 32b 并且小于等于 64b 的归为到 64b 这么一个状态。在图中，程序申请 48b 时所处 64b 状态，而申请到 29b 时所处 32b 状态，故在转移表中，相应的 64b 转移到 32b 状态的统计量增加一个单位。同理，程序申请 980b 大小的内存块获得分配后，32b 转移到 1k 状态的统计量增加一个单位。如果队列达到满队，每更新一次，转移量统计表中相对应的转移统计量都增加一个单位，从而增加转移的统计量，使系统预测下次申请的内存块大小更加精准。

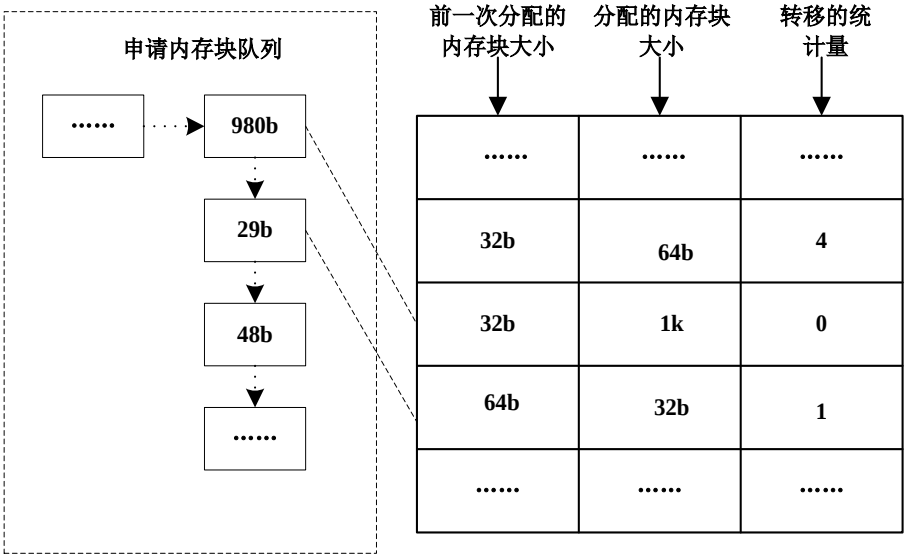

图 4.6 嵌入式内存分配转移量统计表原理图

基于马尔可夫链的嵌入式内存分配预测主要是依靠转移概率矩阵评估状态直接转移的概率大小。前面已经把嵌入式申请内存块大小问题成功转换成分配内存的状态问题，并有转移量统计表记录，以便为状态转移概率矩阵的建立提供数据支持。在嵌入式内存预测机制中，分配的内存块大小就对于马尔可夫链中的状态。假设总共有大小为 10 的状态集 $E=\{8b,16b,32b,64b,128b,256b,512b,1k,2k,16k\}$，则嵌入式内存状态转移概率矩阵为一个 10×10 矩阵，当然也可以设计成其他的状态集。嵌入式内存状态转移概率矩阵 Q 中对应的 i 行 j 列元素用 p_{ij} 表示前一次分配内存块大小 i 转移到本次分配 j 的概率，很明显 p_{ij} 可以通过转移量统计表统计量计算得出。

在内存管理系统中获取转移量统计表，在已经统计的统计数据记录了由前一次分配内存块大小 i 转到分配内存块大小 j 的次数，用 a_{ij} 表示。则

$$p_{ij}=\frac{a_{ij}}{\sum_{j=1}^{n}a_{ij}}\quad(i,j\in E)\qquad\text{公式（4.7）}$$

按照此方法就可以得到转移概率 p_{ij}，将所有的转移概率 p_{ij} 依次排列，就可以构成

嵌入式内存状态转移概率矩阵：

$$Q=\begin{bmatrix} p_{8b8b} & p_{8b16b} & \cdots & p_{8b16k} \\ p_{16b8b} & p_{16b16b} & \cdots & p_{16b16k} \\ & & \cdots & \\ p_{16k8b} & p_{16k16b} & \cdots & p_{16k16k} \end{bmatrix} \qquad \text{公式（4.8）}$$

在应用嵌入式内存状态转移概率矩阵时需要注意马尔可夫状态是指系统实际分配的内存块大小，而不是申请的内存块大小,并且某次所处的状态，下一次可能还会转移同一状态。

转移概率矩阵 Q 中的每一概率元素表示在内存由一种状态转移到另一状态的可能性。假设目前嵌入式分配的内存大小即内存所处的状态为 s_i， $p_{ij}(i,j\in E)$ 描述了从本次分配的内存大小 s_i 向各个状态转移的可能性，p_{i1} 表示下次申请内存的大小状态为 s_1 的可能性，p_{i2} 表示下次申请内存的大小状态为 s_2 的可能性，p_{im} 表示下次申请内存的大小状态为 s_m 的可能性。将 m 个内存状态转移概率按大小排列成不等式，概率最大的就是下一次申请内存块大小的预测结果，即可预测分配的内存块大小经一步转移最大可能达到的状态。

4.8 多核核间通信中断管理

核间中断管理是异构多核处理器的重要组成部分，主要是被用来控制多核之间的任务和消息的同步。核间中断管理机制往往与多核处理器的特点和类型密切相关[12]，通常的多核处理器系统中会有多个核间中断号，通过在系统核间中断寄存器的相应状态位上设为 1，触发相应中断。当核间中断发生时，系统中相应的核间中断状态位将自动变为 1。如果系统要清除此中断，就需要在中断服务处理程序中执行相应的中断清除寄存器位。在多核核间中断策略中，需要每一个内核的核间中断都可以用于核间通信同步，就必须为每一个内核产生的核间中断分配一个不同的核间中断号，并用一个等待队列记录该核核间进程的相关信息。本通信机制中的中断策略采用了注册的方式，每添加一个新的内核需要通知系统，系统就分配中断号，这样使因申请核间共享资源而挂起的进程自动向相应的核间等待队列添加一结点，中断处理流程如图 4.7 所示。

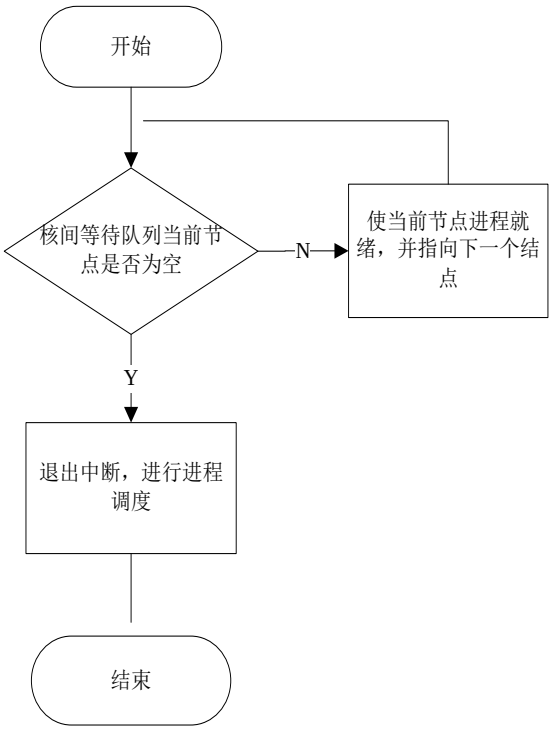

图 4.7 中断处理流程图

当发生核间中断时，系统判断核间等待队列当前结点是否为空，如果不为空就唤醒核间等待队列中当前结点对应的进程，并释放队列结点重新进行任务调度。如果核间等待队列当前结点为空，系统退出中断，进行进程调度。

4.9 核间信号量管理

信号量在单核操作系统中常常被用来处理互斥资源的访问和同步问题[35]。然而，在多核环境中，不仅存在核内同步，而且存在多核之间的同步。本文利用单核系统中的信号量设计思想，设计一种满足多核环境下核间同步需求的核间信号量技术。核间信号量可以进一步分为二元信号量和多元信号量两种，前者用来实现两个进程之间的互斥，后者用来实现多个进程的同步。为了使核间信号量具有更好的灵活性和性能，系统可配置内核中核间信号量的数目，并用静态存储器存储。核间信号量的功能结构如图 4.8 所示。

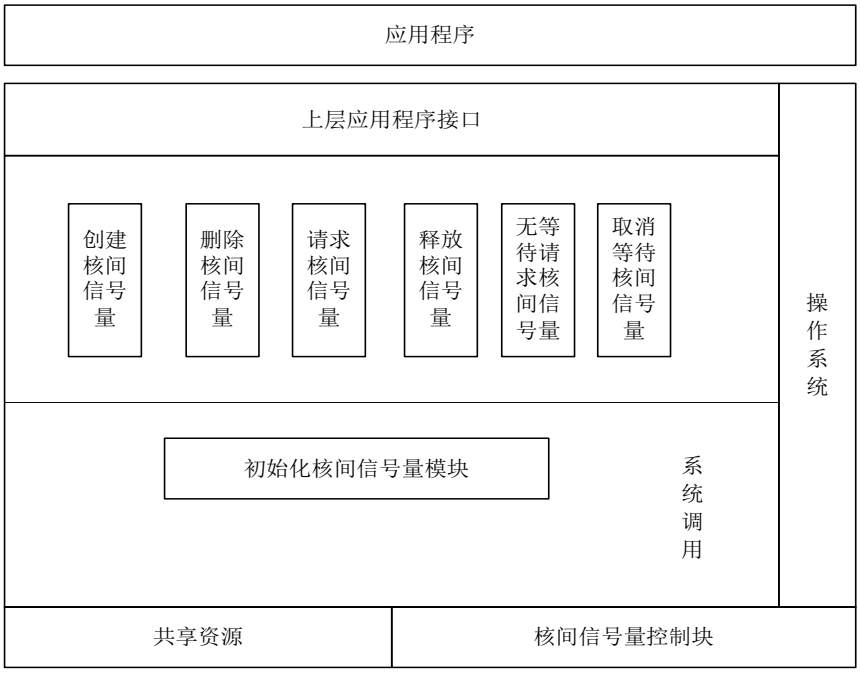

图 4.8 核间信号量功能结构示意图

系统将核间信号量资源封装成对象，再对其进行管理，并为每个创建的核间信号量分配一块控制块。核间信号量管理模块通过创建和删除核间信号量，请求核间号量，释放信号量，等待信号量等方式，实现多核核间进程的同步。在微内核操作系统初始化过程中，系统调用初始化核间信号量功能模块来实现核间信号量模块的初始化，然后通过核间信号量模块中的上层应用程序接口实现上层应用程序的进程同步功能。

4.9.1 模块初始化

初始化核间信号量模块是微内核操作系统初始化的时候必须执行的程序。整个初始化过程对于用户来说是透明的，其主要工作是根据配置信息初始化所有信号量状态和核间信号量控制块，以及把核间信号量控制块用注册的方式生成空闲核间信号量链表。核间信号量初始化流程如图 4.9 所示。

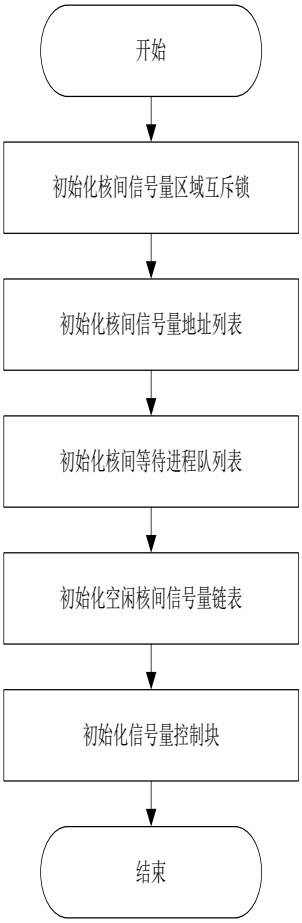

图 4.9 核间信号量初始化流程图

4.9.2 核间信号量的创建与删除

在核间信号量机制中，使用数字作为核间信号量的名字，那么用户可以通过使用这些数字作为核间信号量的句柄。用户创建一个核间信号量时，系统调用核间信号量函数原型，给用户分配一个信号量。当信号量被创建成功之后，就会返回一个信号，否则返回其他的失败信息。核间信号量的删除和创建是配对的系统服务，删除功能是用于回收一个不再使用的核间信号量。其实现相对比较简单，系统只需删除相应结点，并重新将该结点插入到空闲核间信号量链表，以进行重复利用。在本设计核间信号量管理中，一旦系统确定要删除某个信号量，它不管有无进程等待，均执行删除操作，其处理流程如图 4.10 所示。

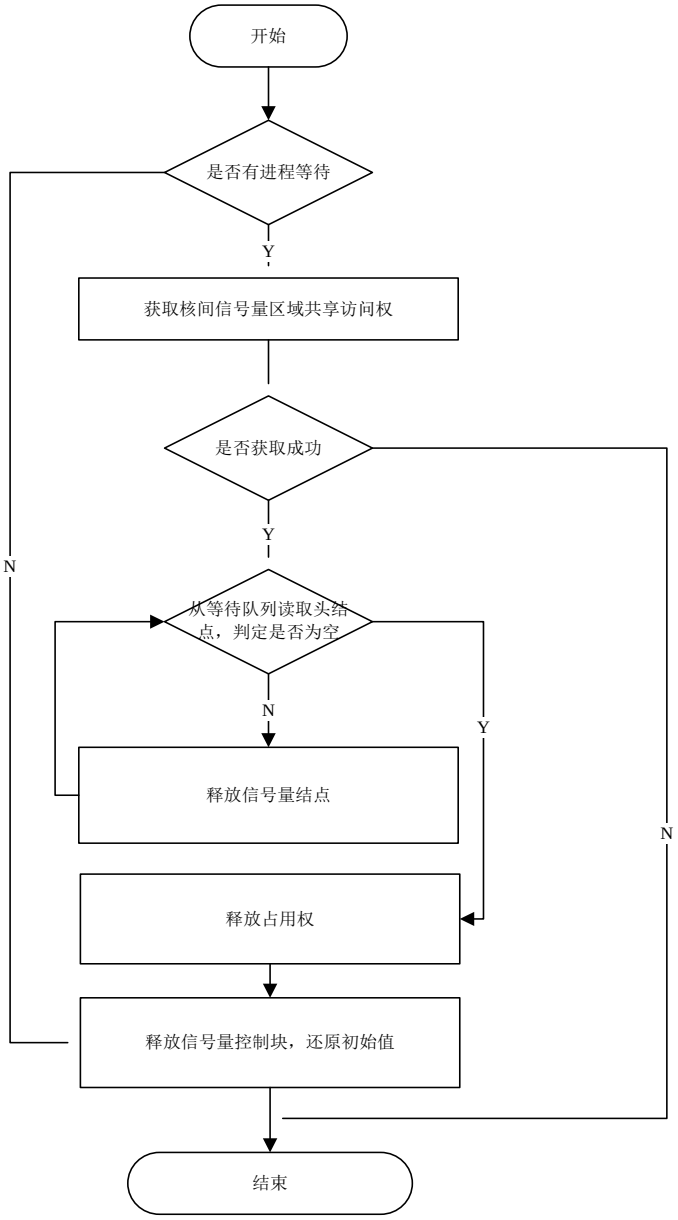

图 4.10 删除核间信号量流程图

在删除核间信号量过程中，先判断是否有进程在等待，如果无任何进程等待，就释放此信号量控制块所控制的资源访问权，并还原其初始值。如果有等待进程，成功获取核间信号量共享资源访问权之后，就释放核间信号量结点，一旦等待队列为空，马上释放资源占用权。最后，释放此信号量控制块所控制的资源访问权，并还原其初始值。

4.9.3 请求核间信号量

在支持多核的微内核操作系统中，独享某个资源时，可以通过获取相应的信号量来实现。系统调用相应的函数来获取信号量，并配置信号量的占用时间，以防止信号量使用过程中的死锁情况。首先，系统判断信号量是否有效，如果不是有效的或者没有成功获得共享内存的访问权，就会进入处理出错状态，返回相应的错误信息。如果系统成功

获取共享内存的访问权，才真正的进入信号量获取阶段。所获的信号量为二元信号量，则判定其是否被占用。如果被占用就将当前进程放入到该信号量的悬挂队列队尾；否则设置信号量被占用和时间戳，使用完信号量之后，释放共享内存的访问权。

4.9.4 释放核间信号量

在系统服务中，用户常常一起使用相配对的核间信号量请求与核间信号量释放。核间信号量的释放方式有系统调用和占用时间两种方式。系统调用释放核间信号量是指系统通过调用释放核间信号量的函数接口，运用核间中断技术，释放核间信号量。占用时间的信号量释放是在占用时间达到时，系统通过调用释放核间信号量的系统服务接口，完成释放核间信号量过程。为了提高微内核操作系统的核间信号释放性能，本文设计的释放核间信号量流程如图 4.11 所示。

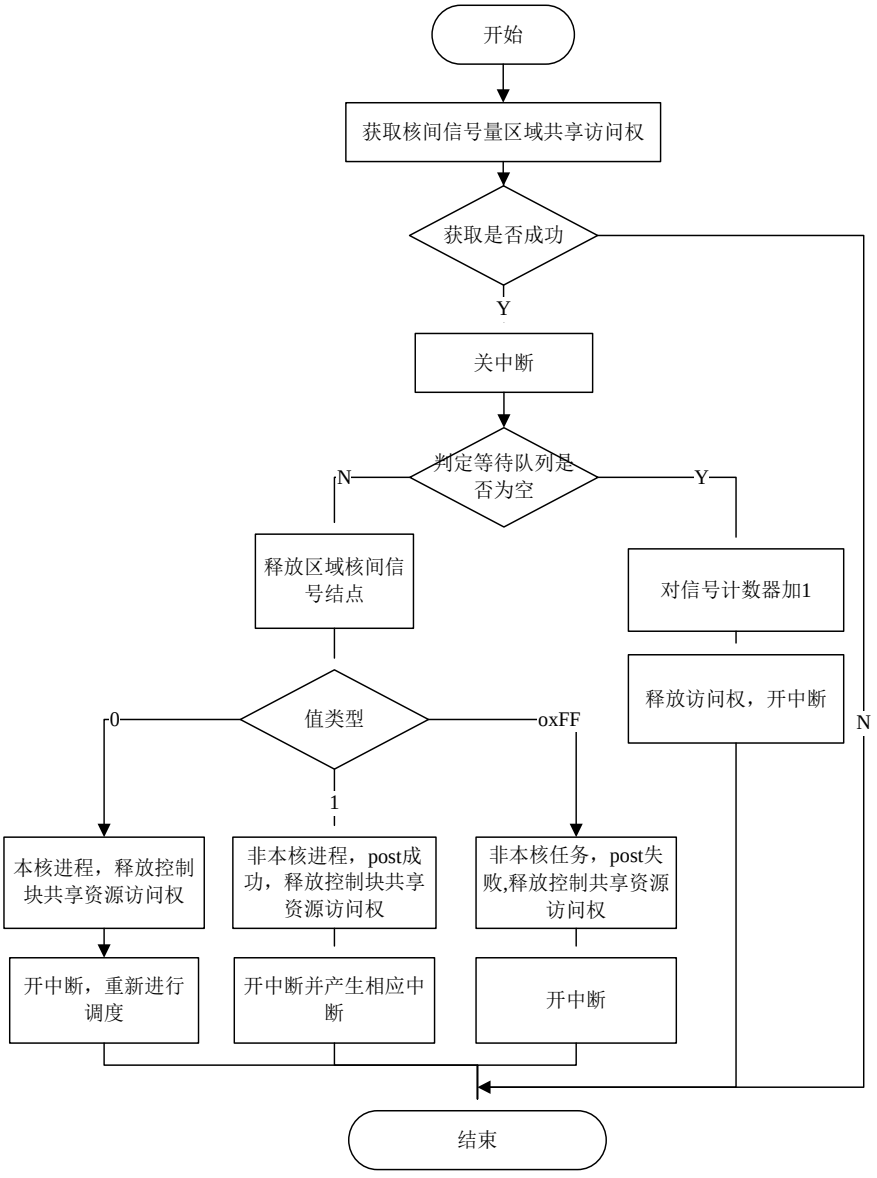

图 4.11 释放核间信号量流程图

在释放核间信号过程中，首先，系统成功获得核间信号量区域共享访问权后，判断等待进程队列是否为空。如果队列为空，核间信号量计数器就加 1，并开中断，释放对共享资源的访问权。否则系统调用释放区域核间信号量结点函数，根据值类型进行相应的操作，最后释放共享资源的访问权。

4.10 本章小结

本章是本文的重点章节，阐述了支持异构多核的微内核操作系统通信机制的具体设计。采用了“总体设计，各个击破”的设计理念进行研究和描述基于消息分类的线程代理通信机制。首先，介绍了面向多核的微内核操作系统通信机制设计目标和通信特点，在此基础上，设计通信机制的总体架构，并从核内和核间通信两个角度进行开展研究设计；接着，通过介绍共享内存的地址映射和寄存器两种通信方式的优势，阐述了设计的核内进程通信机制，并分析了消息长度的划分临界值。同时，描述了基于主内核的代理线程核间通信。然后，在分析马尔科夫链预测原理基础上，建立了可预测分配的共享内存管理策略，提高共享内存的分配效率。最后，对通信中断和核间信号量管理进行改善与设计。

第 5 章 实验测试与结果分析

5.1 实验平台

5.1.1 Simics 仿真器

Simics 最初是由瑞典计算机科学研究院（SICS）开发的一种高性能全系统仿真器。它在 1998 年被交与 Virtutech 公司进行商业化开发与运作，并在 2010 年 2 月被 Intel 收购，被集成到 Intel 全资子公司 Wind River 的产品线中。它提供了一个确定的、受控制的、高性能的虚拟实验平台。它可以模拟存储器、单核处理器和多核处理器平台等多种硬件平台，同时也支持多种虚拟机、硬件软件同步虚拟、驱动程序和操作系统等系统软件测试，具有强大的模拟仿真功能。从技术角度来讲，Simics 可以归类为一种事件驱动的模拟器，可以模拟一系列互斥的事件和 CPU 处理等。软件开发人员在这个模拟平台上可以执行和调试目标软件栈。

它提供了比较简便的操作平台，能高效地在多种模拟硬件上运行代码，可以达到每秒执行数十万条指令。在计算机研究领域中，由于它支持 SMP、CMP 和超线程等多核处理器技术，所以可以模拟计算机内核的源码调试。通过 Simics 的配置文件，我们可以配置内核及处理器的各种参数，比如一级缓存、二级缓存、系统频率和内核结构等。用户自定义一个系统时，可以通过配置系统的硬件参数，可以放内存、处理器、南桥与北桥芯片以及外接设备等，按照 Simics 规范配置好这个系统后，就可以在 Simics 仿真平台上运行和测试系统。因此，本文选择 Simics 作为支持多核的微内核操作系统通信机制的仿真平台。

5.1.2 Simics 多核仿真平台的搭建

Simics 平台是通过 Simics 脚本或 python 脚本来实现多核平台的。其搭建方式比较简单，只需要配置好相应的硬件及操作系统，然后就可以启动自己定义的多核配置，就可以执行相应的程序，具体步骤如下：

首先，下载 Virtutech Simics-3.0.29 的安装包 3ddown.com_setup.exe，点击安装 Virtutech Simics 软件程序，就会进入如图 5.1 的安装界面。

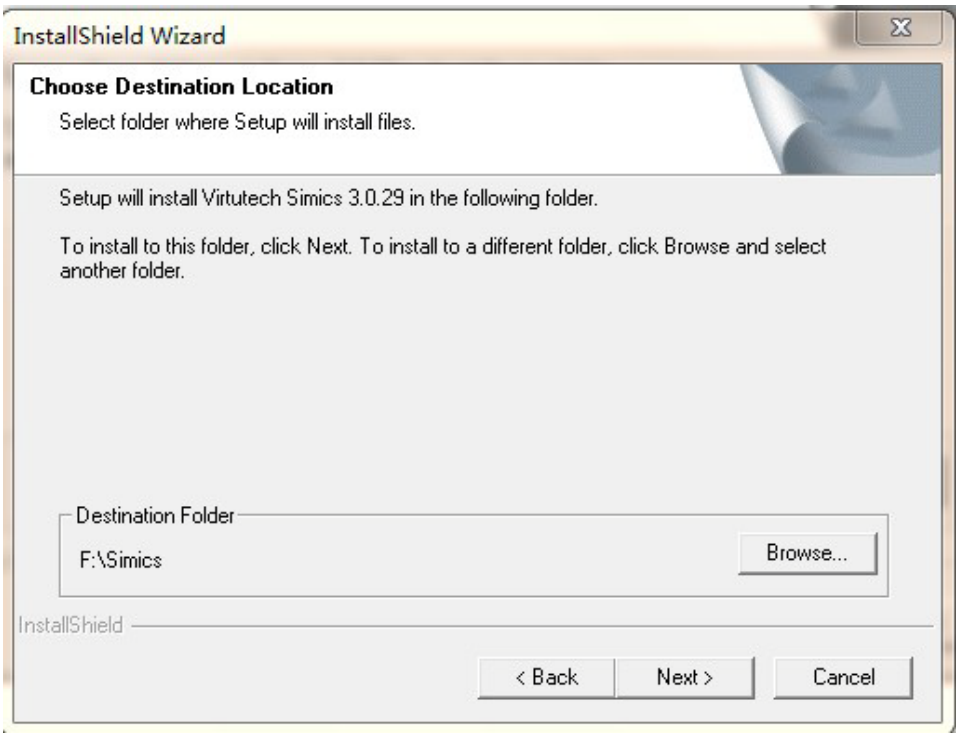

图 5.1 Virtutech Simics 安装界面

点击【Next】按钮，继续安装。当遇到如图 5.2 所示的设置 license 路径界面时，可不设置 license 文件的路径。

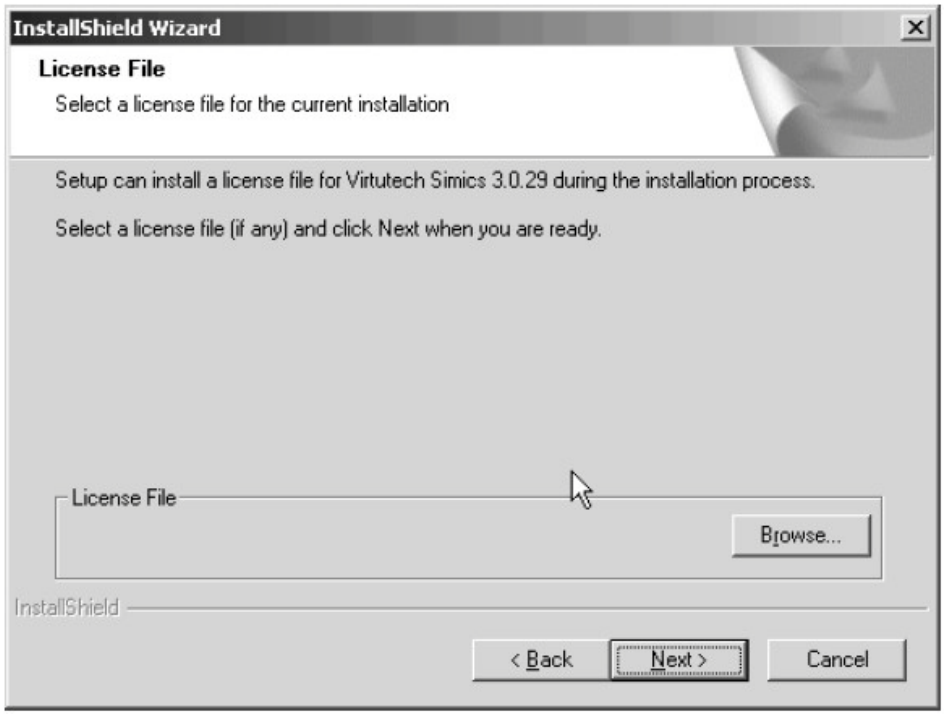

图 5.2 设置 license 路径的界面

点击【Next】按钮，继续安装，进入到如图 5.3 所示的安装 WinPCap 包的界面。可以勾上安装 WinPCap 包，对于调试 NetBSD 内核来说，可以不用选上，其他的设置选择默认即可。

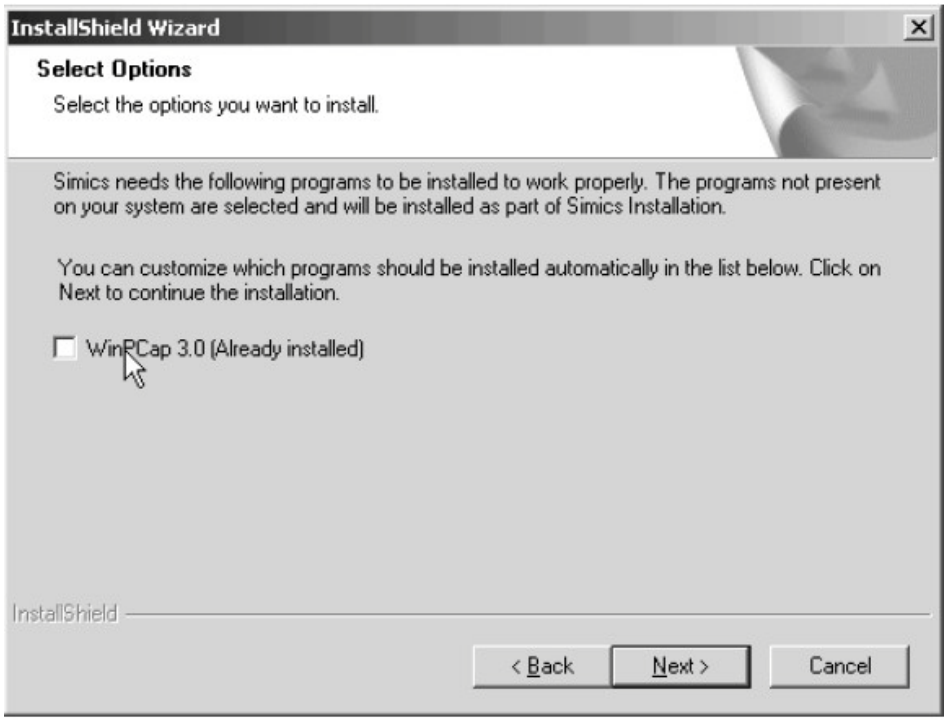

图 5.3 安装 WinPCap 包的界面

点击【Next】按钮，开始安装 Virtutech Simics 文件，待安装完成之后点击【Finish】就标志着安装 Virtutech Simics 完成。从主菜单中启动 Simics 程序，配置 workspace 路径为 D:\Virtutech\workspace，进入到 Simics 程序的主界面。在 Simics 程序的主界面的菜单栏点击【View】下的【Preferences. . . 】子菜单项，打开“Preferences”对话框，配置由 keygen.exe 生成的 license.lic 文件的路径，如图 5.4 所示。然后 Simics 程序可以正常工作了，可以开始进行配置内核的调试环境了。

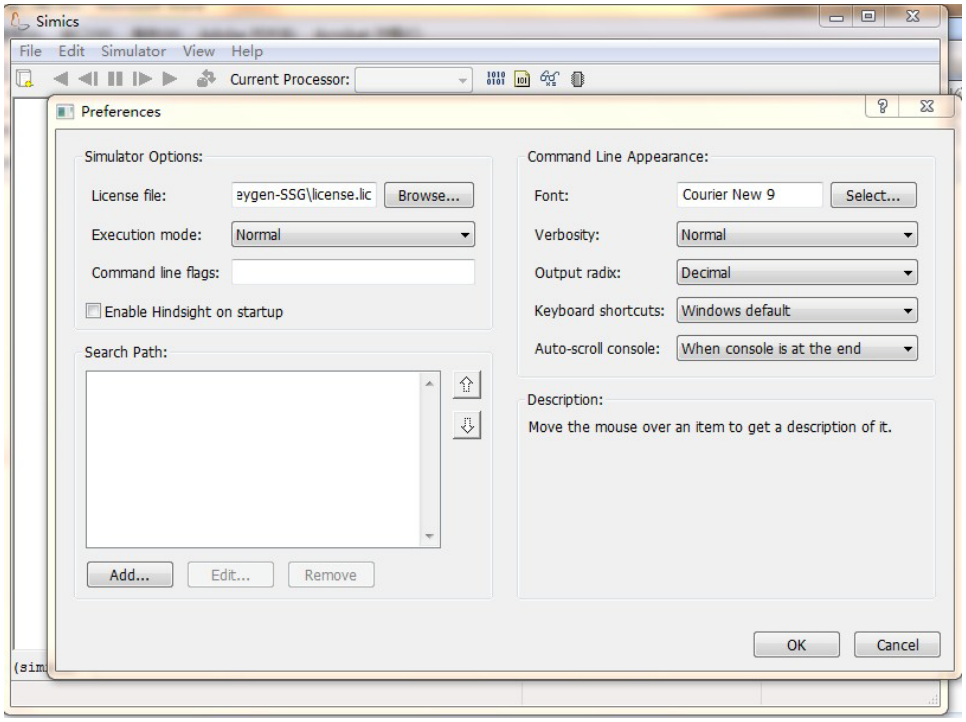

图 5.4 配置 license.lic 文件的界面

接下来，需要配置多核处理器的硬件环境，打开 x86-440bx-agp-system.include 文件，可以进行配置 cpu 数目、虚拟内存大小等参数，其中 num_cpus 为 cpu 的数目，在下面所列代码中设置的值为 3，即一个用来作为主核，另外两个用来进行通信。

```
if not defined freq_mhz     {$freq_mhz = 20}
if not defined cpi          {$cpi    = 1}
if not defined disk_size    {$disk_size    = 2049496236544}
if not defined disk_image   {}
if not defined rtc_time     {$rtc_time    = "2002-09-18 10:00:00 UTC"}
if not defined num_cpus     {$num_cpus    = 3}
if not defined memory_megs  {$memory_megs = 256}
if not defined cpu_class    {$cpu_class = "pentium-4"}
if not defined text_console {$text_console    = "no"}
if not defined use_acpi     {$use_acpi    = TRUE}
add-directory "%simics%/targets/x86-440bx/images"
import-isa-components
import-pci-components
import-std-components
import-x86-components
if ($num_cpus > 15) {
  $bios = "rombios-2.65.2.9-no-mp-tables"
  echo "The system has more than 15 cpus, so a special, less-tested BIOS will be used."
  echo "Only ACPI-aware OSes such as Linux 2.6 will boot correctly in SMP mode."
} else {
   $bios = "rombios-2.65.2.6"
}
$system = (create-x86-apic-system memory_megs = $memory_megs
                                  rtc_time = $rtc_time
                                  acpi = $use_acpi
                                  break_on_reboot = TRUE
                                  bios = $bios)
$count = 0
$create_command = ("create-" + $cpu_class + "-cpu")
while $count < $num_cpus {
    $cpu[$count] = ($create_command cpu_frequency = $freq_mhz cpi = $cpi)
    $system.connect ("cpu" + $count) $cpu[$count]
```

```
    $count += 1
}
……
```

然后找到 ebony-linux-multi.simics 进行配置机器的物理地址、网络 IP、主机名字等参数，其具体实现过程如下：

```
if not defined machine_count    {$machine_count = 3}
if not defined mac_address      {$mac_address = "00:04:ac:00:50:00"}
if not defined ip_address       {$ip_address = "10.10.0.50"}
if not defined host_name        {$host_name = "ebony"}
if not defined id               {$id = 0}
if not defined num_cpus         {$num_cpus  = 3}
@from components import mac_as_list, mac_from_list
@from components import ip_as_list, ip_from_list
@mac = mac_as_list(simenv.mac_address)
@ip = ip_as_list(simenv.ip_address)
$host = $host_name
$create_network = "yes"
if not defined eth_link {
    import-std-components
    $eth_link = (create-std-ethernet-link)
}
$count = 0
while $count < $machine_count {
    set-component-prefix "processor" + $id + "_"
    $mac_address = `mac_from_list(mac)`
    $ip_address = `ip_from_list(ip)`
    $host_name = $host + $id
    run-command-file "%script%/multiprocessor.simics"
    python "mac[5] += 1"
    python "ip[3] += 1"
    $id += 1
    $count += 1
}
sim->cpus_may_share_memory = 5
```

最后，搭建 Simics 多核架构的操作系统环境。由于 linux 是开源操作系统，本文的

实验系统选用 linux 操作系统环境， Simics 启动 linux 操作系统需要配置其镜像文件、启动环境和启动参数，其配置的 ebony-linux-setup.include 文件如下：

```
if not defined ebony_u_boot {$ebony_u_boot = "u-boot-1.1.2"}
if not defined do_login {$do_login = "yes"}
if not defined do_boot {$do_boot = "yes"}
if not defined kernel_image {$kernel_image = "linux-2.4.31.uimage"}
if not defined initrd_image {$initrd_image = "busybox-0.60.2.uimage"}
if not defined boot_command {$boot_command = "setenv bootargs rw; bootm ffc00000 ffd00000"}
if not defined ip_address {$ip_address = "10.10.0.50"}
if not defined host_name {$host_name = "ebony"}
if not defined service_node {}
if $memory_megs > 256 {
    echo "---"
    echo "You are currently running the Ebony configuration with"
    echo "more than 256MB of simulated RAM. Please note that this"
    echo "is not well supported in u-boot, even when compiling"
    echo "with the CONFIG_VERY_BIG_RAM option enabled."
    echo "---"
}
if $service_node {
    local $sn = ($service_node.get-component-object sn)
    ($sn.add-host name = $host_name
     ip = $ip_address domain = network.sim
     mac = $mac_address)
}
default-port-forward-target $ip_address
$plb = ($system.get-component-object plb)
$plb.load-binary $ebony_u_boot 0x100000000
$plb.load-file $kernel_image 0x1ffc00000
if $initrd_image != "" {
    $plb.load-file $initrd_image 0x1ffd00000
}
@from components import mac_as_list
@plb = sim.objects[simenv.system].object_list['plb']
```

```
@plb.memory[[0x1fffffe0c, 0x1fffffe11]] = mac_as_list(simenv.mac_address)
@plb.memory[[0x1fffffe18, 0x1fffffe1d]] = mac_as_list(simenv.mac_address1)
if $do_boot == "yes" {
    script-branch {
        local $con = ($console.get-component-object con)
        local $cmd = $boot_command
        # Wait for u-boot prompt, type boot
        $con.wait-for-string "=>"
        $con.input $cmd + "\n"
    }
}
if $do_login == "yes" {
    script-branch {
        local $con = ($console.get-component-object con)
        local $ip = $ip_address
        # Wait for login prompt, login as root
        $con.wait-for-string "ogin: "
        $con.input "root\n"
        # Wait for root prompt, do ifconfig
        $con.wait-for-string "# "
        $con.input "ifconfig eth0 " + $ip + " netmask 255.255.255.0 \n"
    }
}
```

经过以上的操作之后，就完成了 Simics 多核仿真平台的搭建。在菜单栏 File 中点击【New Session】选择 workspace 工作目录中选择 multiprocessor.simics 脚本文件，便可看到 Simics 平台初始化了 3 个处理器，processor0_cpu0，processor1_cpu0，processor2_cpu0，完成了 Simics 多核仿真平台的初始化，如图 5.5 所示。那么，就可以进入操作系统并运行应用测试程序。

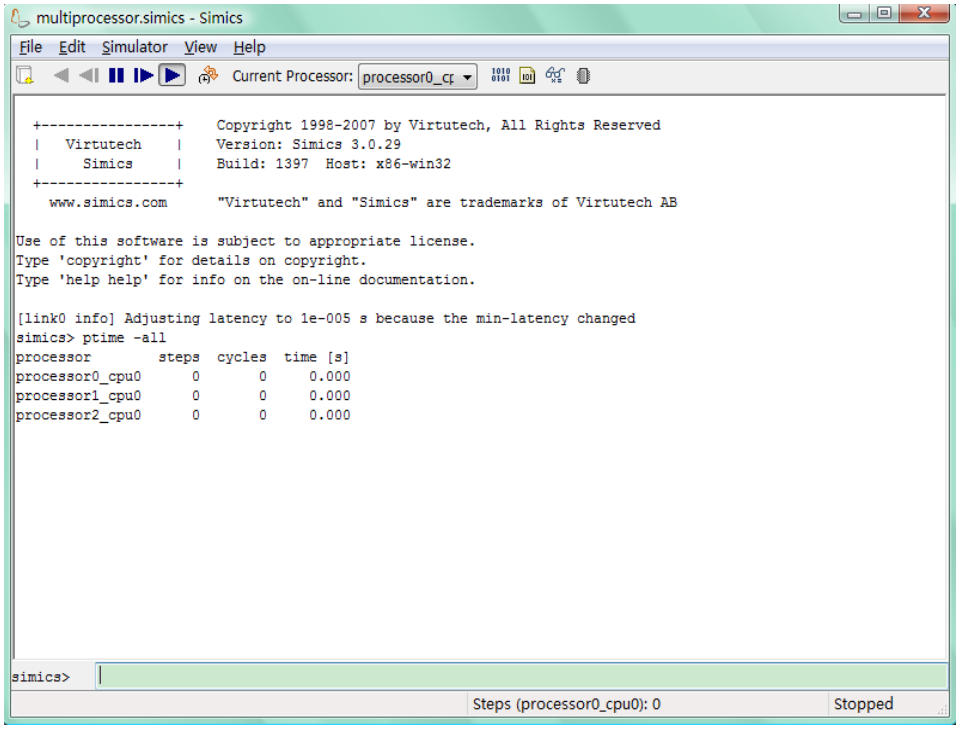

图 5.5 多核处理器初始化界面图

5.2 多核通信机制的实现

本文设计的基于消息分类的线程代理通信机制用 C 语言和脚本编写后，添加到系统中，用编译软件编译成 craff 类型文件，放入 Simics 仿真平台中，才能顺利运行。下面对基于消息分类的线程代理通信机制的部分核心代码进行描述。

5.2.1 进程间通信的实现

在对进程间通信机制进行实现之前，我们首先要声明一些函数和常量：

```
#define MSG_SHORT    1
#define MSG_LONG_SEND   2
#define MSG_MAP_SEND   4
#define MSG_MAP_RECEIVE    8
#define MSG_BOTHBUF          0xC
#define MSG_NOREPLY          0x20
#define MSG_LONG_RECEIVE 0x40
#define MAX_MSG_LONG  256
#define MSG_COPY_SHORT(to, from)    (*(to) = *(from))
extern error_t send_message(tid_t tid, message_t msg);
```

```
extern error_t     receive_message(tid_t tid, message_t msg);
extern int reply_message(tid_t , message_t );
```

在基于消息分类的线程代理通信机制中，进程间通信通过三个函数来实现消息的传递过程：transfer_msg （消息传递）、send_message（发送消息）、receive_message（接收消息）。

（1）消息传递函数的实现

```
static inline int transfer_msg(thread_t rcv, thread_t snd)
{
    u32 flags = snd->msg.flags;
    message_t msg = &snd->msg;
    vm_offset_t rcvptr = (vm_offset_t)msg->rcv_ptr,
                                sndptr = (vm_offset_t)msg->snd_ptr;
    int ret = 0;
    if(flags & MSG_SHORT)
       return ret;
    else if(flags & MSG_MAP_RECEIVE)
    {
         ret = vm_share_addr(rcv->proc->map,
                                    snd->proc->map,
                                    &rcvptr, msg->rcv_bufsize, PAGE_RW|PAGE_USER);
         msg->rcv_ptr = (void *)rcvptr;
    }
    else if(flags & MSG_MAP_SEND)
    {
        ret = vm_share_addr(rcv->proc->map,
                                snd->proc->map, &sndptr, msg->snd_bufsize, PAGE_USER);
        msg->snd_ptr = (void *)sndptr;
    }
    return ret;
}
```

（2）发送消息的函数实现

```
error_t send_message(tid_t tid, message_t mesg)
{
    thread_t r_thr;
    thread_t sender = current;
```

```
    int ret;
    r_thr = find_by_tid(tid);
    if(!r_thr)
        return -ESRCH;
    sender->msg = *mesg;
    sender->msg.tid = sender->tid;
    list_enqueue(&r_thr->rcvq, sender, sndq, thread_t);
    ret = transfer_msg(r_thr, sender);
    if(ret < 0)
    {
        kprintf("send_message: Transfer of message failed \n");
        return -ENOMEM;
    }
    if(r_thr->states & RECEIVE_BLOCKED)
        wakeup(r_thr);
    make_send_blocked(sender);
    *mesg = sender->msg;
    return sender->msg.w1;
}
```

（3）接收消息的函数实现

```
error_t  receive_message(tid_t tid, message_t msg)
{
    thread_t rcv = current;
    thread_t snd;
again:
    if(list_isempty(&rcv->rcvq))
    {
        make_receive_blocked(rcv);
        goto again;
    }
    else
    {
         list_dequeue(&rcv->rcvq, snd, sndq, thread_t);
         if(!snd)
         {
```

```
            kprintf("snder not found");
            return -1;
        }
        *msg = snd->msg;
        if(snd->msg.flags & MSG_NOREPLY)
        {
            wakeup(snd);
            return 0;
        }
        snd->states = REPLY_BLOCKED;
    }
    return snd->tid;
}
```

5.2.2 共享内存预测分配的实现

在通信机制中，共享内存的预测分配算法实现过程如下面的代码所示。当线程需要统计等信息时，就会寻找相应的区域取出所需要的统计数据。每当分配内存块之后，由预测线程进行预测接下来所需要的内存块大小，再由管理系统提前进行分割预测申请的内存块大小，以便未来系统申请时立即分配共享内存块。其算法如下：

```
#define statisticsmax 64 ;
int size;
typedef struct {
    QElemType *base;
    int *pre;
    int *rear;
}SqQueue
SqQueue Q;
if((Q->rear+1)% statisticsmax = =Q->pre){
  /*如果已满的情况*/
    Q->pre=Q->rear;
    Q->rear=Q->rear+1;
    Q->pre=address;
}
else{
```

```
            Q->pre=Q->pre+1;
            Q->pre+1;
        }
        then size=pre(SqQueue *Q);   /*预测算法进行预测，并把结果赋给 size;*/
        Malloc(size);   /*系统分割或合并大小为 size 的所需块进行分配*/
```

5.2.3 通信进程优先级的实现

在基于消息分类的线程代理通信机制中，通信进程的优先级取决于进程本身的优先级和等待的时间。本文设计了 16 个优先等级，其具体实现代码如下：

```
        ……
    Void SetTaskBit(INT8U Prio){
        if(Prio>8)OSRdyTbl[1]|=(1<<(15 Prio));
        else OSRdyTbl[0]|=(1<<7 Prio);
    }
    void ClrTaskRdyBit(INT8U Prio){
        if(Prio>8)OSRdyTbl[1]&=(1<<~(15 Prio));
        else OSRdyTbl[0]|&=~(1<<7 Prio);
    }
    INT8U FindHighestRdyTask(void){
        INT32U temp;
        INT8Uprio=0;
        if(OSRdyTbl[0]!=0){
        temp=OSRdyTbl[0];
    }else{
        temp=OSRdyTbl[1];
        prio+=8;
    }
    wihle(temp<0x80000000){
        temp<<=1;
    }
        ……
    }
        ……
```

5.3 性能测试及结果分析

5.3.1 邮箱机制

邮箱机制作为进程间通信的一种实现方法，在许多的多核操作系统中都得到了应用。邮箱机制是指发送进程会向接收进程发送一个指针类型的变量。邮箱发送的不是消息本身，而是指向消息的指针，指针指向的内容就是那则消息。

邮箱机制在初始化的时候，系统也会建立一个进程列表，这个进程列表中包含所有等待接收消息的进程，当有进程需要接收消息的时候，这个进程就会被添加到这个进程列表中。当有消息进入到队列中时，在这个进程列表中，优先级最高的进程会获得这个消息，或是在这个进程列表中最前面的进程获得这个消息。

邮箱是内存中的一块能被所有内核共享的区域，存在于邮箱中的数据信息可以被所有的内核读取和修改，同时在系统启动的时候，邮箱的地址就被告知给所有的内核，而这个地址也是固定不变的，方便内核的使用。并且，由于邮箱的地址将会被所有的内核经常的使用，所以会被放入到所有内核所共享的二级缓存之中，提高邮箱机制的效率。

如图 5.6 所示为邮箱机制下的一个消息的传递过程：进程 A 和进程 B 分别位于两个不同的内核 A 和内核 B。当进程 A 需要同进程 B 进行通信时，进程 A 首先把消息发送到内存中的邮箱之中，然后进程 A 所在的内核 A 会通过一个处理器间的中断把一些信息发送给要进行通信的进程 B 所在的内核 B，这些信息主要包含邮箱的索引信息以及接收进程 B 的一些信息，如进程号等。然后，内核 B 会通过这些信息找到接收进程 B 并获取邮箱中的消息。

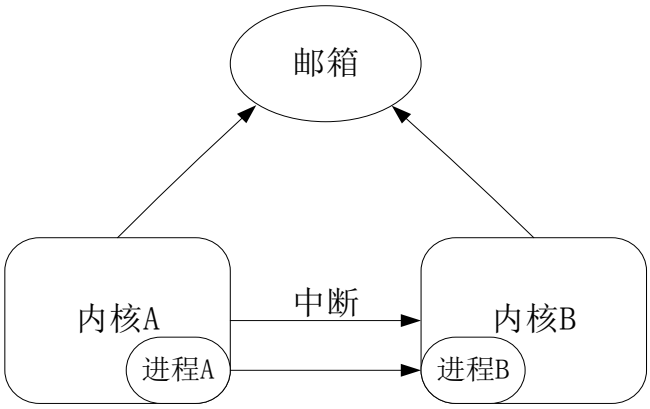

图 5.6 邮箱机制

5.3.2 性能测试程序设计

实现基于消息分类的线程代理通信机制后，需要对其性能进行测试。本文设计的测试程序的功能是三个线程同时进行打印。当一个线程完成打印之后会随机传递信息给另

一个线程，让另一个线程执行打印任务，在其完成之后，又会随机发送信息给下一个线程。可以通过改变打印次数的参数进行测试，具体代码如下：

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 3
void *myTask(void * p_thread_id) {
        int i;
        double sum=0.0;
        int thread_id = (int) p_thread_id;
        int RUN_TIMES = 2000;
        for(i=0; i<RUN_TIMES; i++) {
           printf("%d ", i);
              sum += i;
        }
        printf("\nthread%d run times = %d\n", thread_id, RUN_TIMES);
        fflush(NULL);
        pthread_exit(NULL);
}
int main() {
    pthread_t threads[NUM_THREADS];
    int t;
    //Create threads
    for(t=0; t<NUM_THREADS; t++) {
        int ret;
        ret = pthread_create(&threads[t], NULL, myTask, (void *) t);
        if(ret != 0) {
            printf("pthread_create: ERROR: t=%d return code=%d\n",t, ret);
            exit(1);
        }
        fflush(NULL);
    }
    //Wait for threads to complete
    for(t=0; t<NUM_THREADS; t++) {
        int ret, status;
        ret = pthread_join(threads[t], (void **)&status);
```

```
            if(ret != 0) {
                printf("pthread_join: ERROR: t=%d return code=%d\n",t, ret);
                exit(1);
            }
            fflush(NULL);
        }
        pthread_exit(NULL);
        return 0;
    }
```

5.3.3 实验结果对比分析

本文设计的通信机制和邮箱通信机制环境下分别运行测试程序，并记录实验结果。如图 5.7 所示，在基于消息分类的通信机制下，测试程序循环执行了 2000 次，其中 load average 中的 1.04 是实际时间，0.02 是用户 CPU 时间，0.01 是系统 CPU 时间。其数据结果在每次运行中会有略微的变化，与当时的系统运行状况有关，但其产生的误差相对较小，几乎不影响结果。

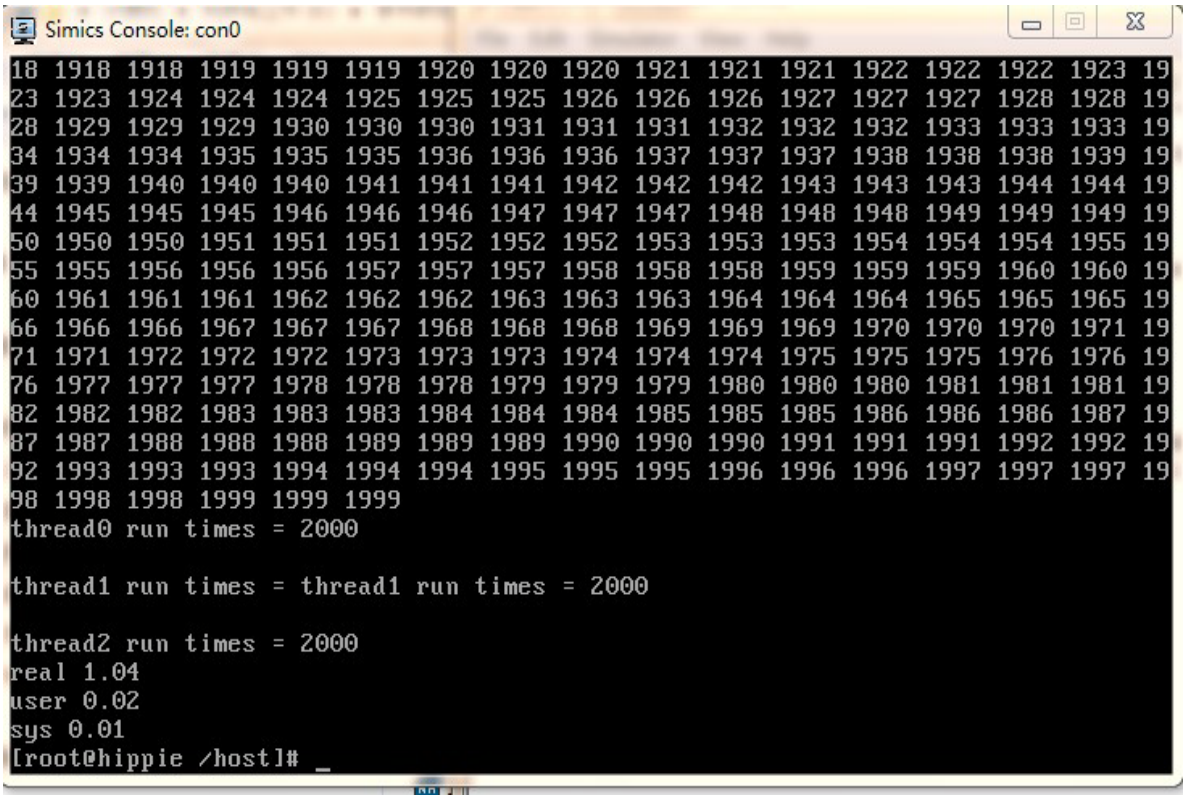

图 5.7 性能测试运行结果图

那么把循环打印次数修改为 1000、3000、6000、10000、25000，再进行测试，记录实验结果。与此对应的，在邮箱通信机制下，运行打印次数分别为 1000、2000、3000、

6000、10000、25000 的测试程序，最终得到如表 5.1 所示的实验结果。

表 5.1 性能测试数据对比

通信机制	1000 次	2000 次	3000 次	6000 次	10000 次	25000 次
邮箱机制	0.63	1.26	2.03	4.29	7.17	18.81
本文的通信机制	0.52	1.04	1.58	3.15	5.28	13.67

为更清楚看到消耗的时间随着循环次数的变化趋势，根据表 5.1 中的测试数据得到的趋势图如图 5.8 所示。

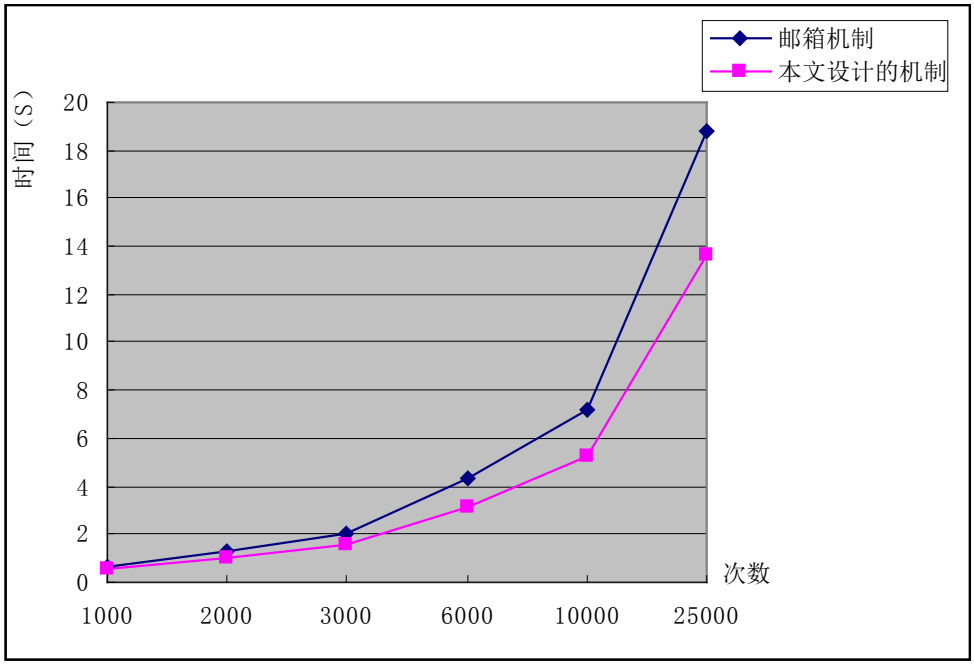

图 5.8 性能趋势图

从图 5.8 的趋势图中可以看出，基于消息分类的线程代理通信机制要比邮箱机制通信花费的时间少，并且随着进程间通信次数的增多，两种机制的差距越来越大，有进一步扩大的趋势。因此可以得出，本文设计的基于消息分类的线程代理通信机制提高了系统通信的效率，增强了系统的实时性。

5.4 本章小结

本章对基于消息分类的线程代理通信机制进行测试与分析。通过配置 Simics 文件和 include 文件，在 Simics 仿真平台上搭建了多核处理器硬件平台和微内核操作系统环境。然后对与通信机制相关的代码进行了修改和编写，实现了基于消息分类的线程代理通信机制。最后分别在本文设计的通信机制和邮箱通信机制下运行测试程序，并通过实验结果对比分析得出基于消息分类的线程代理通信机制的性能更加高效。

第 6 章 总结与展望

6.1 工作总结

伴随着半导体工艺进入纳米时代，多核处理器通过优化的核内部体系架构，能在较低的主频下拥有更高的处理器性能、更有效的电源利用率，并且占用的物理空间相对小等许多单核处理器无法具备的优势，不可避免地取代单核成为嵌入式微内核操作系统未来发展的主流环境。因此，本文针对异构多核处理器和微内核的特点，设计了一种基于消息分类的代理线程微内核操作系统通信机制。从介绍支持异构多核的微内核操作系统通信机制国内外研究现状入手，阐述其相关概念和技术。并把面向多核的微内核操作系统通信归根到系统进程间的通信，描述典型的微内核操作系统通信策略，分析了这些机制的优缺点。在此研究基础上，本文重点对多核环境中微内核操作系统进程间通信机制做了如下工作：

（1）分析了多核和单核环境区别，根据支持多核的微内核操作系统通信设计目标，设计了一个基于消息分类的代理线程通信架构。这个通信架构便于分解研究和实现，具有较好的扩展性和可靠性。

（2）通过分析寄存器和共享内存的地址映射通信方式的优缺点，本文将消息进行分类，发挥寄存器和共享内存的地址映射的各自优势，实现了核内进程间的通信，减少了核内进程间通信过程中的延迟，提高了核内进程间通信的效率。

（3）在主内核协助下，利用多核 CPU 共享二级缓存和内存的特点，采用共享内存的地址映射方式，设计了基于主内核的代理线程核间通信方案，为多核环境的嵌入式实时微内核操作系统提供部分的理论基础。

（4）研究了马尔可夫链预测原理，把共享内存中分配的内存块大小作为马尔可夫预测中的状态，通过转移概率矩阵评估状态直接转移的概率大小，设计了可预测的共享内存分配算法，提高了通信时分配共享内存的效率。

（5）设计了优先级和等待时间相结合的通信优先级策略，并改进了中断管理和信号量管理机制，提高了通信的相关支持技术性能。

最后，本文通过 Simics 仿真器构建的多核实验平台，模拟了设计的通信机制和邮箱通信机制下的系统，通过测试程序测试性能，并分析所得到的结果，得出本文的设计方案提高了进程间通信的效率，提升了支持多核的嵌入式实时微内核操作系统的性能。

6.2 研究展望

本文设计的基于消息分类的代理线程通信机制，能够较好的适应多核环境下的微内

核操作系统，并具有比较高的通信效率。但是，面向多核的微内核操作系统是复杂的，况且多核技术和微内核技术发展迅猛，本文研究的多核环境中的微内核操作系统通信技术不免存在一些不足。而且支持多核的微内核操作系统对能耗、安全性等性能有了更高的要求，这就要求我们进一步加深对通信机制的研究。本文认为今后可以从以下几个方面开展研究：

（1）当多核处理器的核心数量越来越多时，需要研究降低多核之间的通信功耗机制，解决现有的通信机制在能耗上存在的不足；

（2）研究微内核操作系统在多核架构上通信策略的通用性以及遵循的标准规范，为以后的研究和开发提供参考。

参考文献

[1] 邱霆. 基于微内核的地址空间架构的研究[硕士学位论文] .杭州：浙江工业大学，2008

[2] Xiao-hui Cheng， Ming-qiang Li， Xin-zheng Wang. Embedded real-time operating system micro kernel design. ICMIT'05: Information Systems and Signal Processing， Published by SPIE-The International Society for Optical Engineering (USA)，2006: 246～249

[3] 张荫芾，应忍冬，周玲玲. 支持多核架构的微内核操作系统设计. 计算机工程， 2009, 35(23): 249～251

[4] 张攀勇，孟丹，霍志刚. 多核环境下高效集合通信关键技术研究. 计算机学报，2010，33(2): 318～325

[5] Graham Richard L， Shipman Galen M. MPI support for multi-core architectures: Optimized shared memory collec-tives. Proceedings of the Recent Advances in Parallel Virtual Machine and Message Passing Interface. Dublin, Ireland,2008: 130～140

[6] Philipp Kupferschmied. Design and Implementation of a Microkernel-Based Operating System for NUMA Machines. Karlsruhe: University Karlsruhe， 2006

[7] 胡新安. 支持动态任务调度的多核分布式操作系统设计[硕士学位论文] .哈尔滨：哈尔滨工业大学，2011

[8] 俞建德. 支持异构多核的嵌入式实时操作系统 SmartOSEK OS-M [硕士学位论文] .杭州：浙江大学.2008

[9] 黄国睿，张平，魏广博. 多核处理器的关键技术及其发展趋势. 计算机工程与设计. 2009, 30(10):2414～2418

[10] 谢子光. 多核处理器核间通信技术研究[硕士学位论文]. 成都：电子科技大学，2009

[11] 郭广浩，沈绪榜. 多核微处理器核间高速互连技术. 计算机技术与发展. 2012, 20(06): 30～34

[12] 黄志钢，盛肖炜. 多核处理器结构与核间通信的 CMC 总线设计. 沈阳理工大学学报，2012, 31(06): 70～75

[13] 王红玲，吕强. 一个微内核操作系统的设计与实现. 微电子学与计算机，2008，25（4）:9～17

[14] 吴庆波，戴华东，吴泉源. 麒麟操作系统层次内核设计技术. 国防科技大学学报，2009，31（2）:76～80

[15] 李泽球. 嵌入式操作系统微内核体系结构的研究与设计[硕士学位论文]. 桂林：桂林理工大学，2011

[16] 郭杨，王新社. 嵌入式操作系统的硬实时微内核设计. 计算机工程与设计，2008，29（1）:115～118

[17] 方正书. 基于 L4 的 FIFO 通讯机制的研究和实现[硕士学位论文]. 兰州：兰州大学，2010

[18] 汪健，张磊，王少轩，赵忠惠，陈亚宁. 多核处理器核间高速通讯架构的研究. 电子与封装. 2011, 11(06): 41～48

[19] 袁立业. Noc 上的多核间通信策略研究[硕士学位论文]. 大连：大连理工大学，2009

[20] 王进祥. NoC_MPSim：基于片上网络通信架构多核仿真平台. 中国集成电路. 2011,6(145)，31～40

[21] Xiao-hui Cheng， Zeqiu Li， Fuyuan Xiao. An advanced inter-process communication model of micro-kernel. In ICCDA'11: Proceedings of IEEE 3rd International Conference on Computer Design and Applications. 2011

[22] 顾宝刚. 基于 VxWorks 的异构多核处理器软件系统的研究与设计[硕士学位论文]. 长沙：国防科学技术大学，2008

[23] 张荫芾. 基于多核处理器架构的嵌入式微内核操作系统的研究与设计[硕士学位论文]. 上海：上海交通大学，2009

[24] 唐伟杰. 微内核进程间通信的优化与实现[硕士学位论文]. 杭州：浙江工业大学，2008

[25] 邢向磊. 基于 ARM11MPCore 的多核间通信机制研究. 计算机应用与软件，2009，5(26): 108～110

[26] 孙帅帅. 基于嵌入式多核处理器的通信及中断问题的研究[硕士学位论文]. 成都：电子科技大学，2011

[27] 王宽卿. 微内核进程间通信的研究[硕士学位论文]. 杭州：浙江大学，2010

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

[29] 何军，王飙. 多线程处理器资源分配策略. 计算机工程， 2008， 34(15): 283～287

[30] Rami Matarneh. Multi Microkernel Operating Systems for Multi-Core Processors. Journal of Computer Science， 2009， 5 (7): 493～500

[31] 李彦冬，雷 航. 多核操作系统发展综述. 计算机应用研究. 2011, 28(09) : 3216～3219

[32] Wang Chengjun. Research on the Microkernel Technology. 2009 Second International Workshop on Computer Science and Engineering. 2009(J):199-202

[33] 蒋建春，汪同庆. 一种异构多核处理器嵌入式实时操作系统构架设计. 计算机科学，2011，38（6）:298～303

[34] 陈书明. 一种面向多核 DSP 的小容量紧耦合快速共享数据池. 计算机学报， 2008，31(10):1737～1743

[35] 肖学甲. 基于 AMP 架构的多核间任务同步与通信的设计与实现[硕士学位论文]. 西安：西安电子科技大学，2011

[36] 李静梅，王军锋，张 岐. 一种适应多核处理器核间通信机制的设计. 智能计算机与应用，2011，1（2）:26～30

[37] Yu-Hsien Lin， Chiaheng Tu， Chi-Sheng Shih， Shih-Hao Hung. Zero-Buffer Inter-Core Process Communication Protocol for Heterogeneous Multi-core Platforms. Proceedings of 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications，2009.69～78

[38] D. Kim， S. Lee， D. Shin. Design of the operating system virtualization on L4 microkernel. 2008 Fourth International Conference on Networked Computing and Advanced Information Management. School of Information and Communication Engineering Sungkyunkwan University， 2008.223～229

[39] Sang-Min Lee， Dong-Geun Kim， Dong-Ryeol Shin. Design of the L4 microkernel based lghtweight mobile middleware for mobile phone. 2008 IEEE

[40] 刘珊珊. Minix 操作系统的分析、改进与测评[硕士学位论文]. 成都：电子科技大学，2009

[41] 王怀武，田丽娜. MINIX 嵌入式操作系统进程调度的移植. 兰州大学学报，2008，44（3）:97～99

[42] 陆冠群，胡 光，涂时亮. 基于 Minix 的进程间通信系统的设计与实现. 计算机系统应用，2010，19（7）:1～5

[43] 谢铖. 多内核构件化嵌入式操作系统的研究：[硕士学位论文]. 杭州：浙江大学，2006

[44] Kumar Rahul， Mamidala Amith R， Panda Dhabaleswar K.Scaling alltoall collective on multi-core systems//Proceedingsof the IEEE International Symposium on Parallel and Distrib-uted Processing 2008(IPDPS 2008). Miami， USA， 2008:1-8

[45] Tu Bi-Bo， Fan Jian-Ping， Zhan Jian-Feng， et al. Accurate analytical models for message passing on multi-core clus-ters//Proceedings of the 2009 Parallel， Distributed and Net-work-Based Processing (PDP 2009). Weimar， Germany，2009: 133-139

[46] Harvey M. Deitel， Paul J. Deitel， David R. Choffnes. Operating Systems (Third Edition). 北京：清华大学出版社，2007

[47] Wang Guoqin， Xu Meihua. Research on Tightly Coupled Multi-Robot Architecture Using Microkernelbased， Real-Time， Distributed Operating System. 2008 International Symposium on Information Science and Engieering， 2008， 8～13

[48] Mamidala Amith R，Kumar Rahul，De Debraj，Panda Dhabaleswar K. MPI collectives on modern multicore clus-ters: Performance optimizations and communication charac-teristics//Proceedings of the 8th IEEE International Sympo-sium on Cluster Computing and the Grid (CCGRID'08).Lyon，France， 2008: 130～137

[49] Yan Liu， Ted Wong. Component architecture and modeling for microkernel-based embedded system development. National ICT Australia Ltd，Australia. 19th Australian Conference on Software Engineering， 2008: 1137～1143

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

[51] Rami Matarneh. Multi Microkernel Operating Systems for Multi-Core Processors. Journal of Computer Science, 2009, 5 (7): 493～500

[52] 张国杰，张毅. 多核多线程处理器 XLR732 的多核间通信. 重庆工学院学报，2008，22（10）:148～152

[53] Xiao-Hui Cheng， You-min Gong， Xin-zheng Wang. Study of Embedded Operating System Memory Management. ETCS'09: Computer Science， Published by the IEEE Computer Society，2009:962～965

[54] Xiao-hui Cheng， Liang Zhang， A research of inter-process communication based on shared memory and Address-mapping. In ICCSNT 2011: Proceedings of Int. Conference on Computer Science and Network Technology， Harbin， China，Dem 2011， 24～26

[55] Fang-Ming Yang. The Implementation of Virtual Memory in Embedded Micro-Kernel Oprating

System. 台南：国立成功大学，2008

[56] 顾胜元，杨丹，黄海伦. 嵌入式实时动态内存管理机制. 计算机工程.2009, 35(20):264～269

[57] 李志军，王铮，王帅. 嵌入式系统的自适应动态内存分配算法. 计算机工程.2007,33（20）:91～93

[58] 刘魁，李实英，李蕊. 基于隐马尔可夫模型的热路径预测算法研究.计算机应用研究.2010, 27(7):2468～2471

[59] 宁丹，刘鸿雁. 基于模糊聚类的马尔可夫方法在需求预测中的应用. 计算机应用与软件.2008,25（6）:150～152

[60] Xiao-hui Cheng，An-ming Xu. A new memory management mechanism based on Embedded system.The 3nd IEEE International Conference on Software Enginee-ring and Service Sciences, Beijing ,China, July 2012,348～402

个人简介、申请学位期间的研究成果及发表的学术论文

许安明，男，1987 年 2 月出生于湖南省永州市，2010 年 7 月毕业于湖南商学院信息管理与信息系统专业，获管理学学士学位。2010 年 9 月考入桂林理工大学攻读硕士学位，师从程小辉教授，研究方向为嵌入式系统应用技术。

一、申请学位期间参加的研究项目：

1、项目来源：国家自然科学基金项目
项目名称：《嵌入式实时操作系统微内核体系结构及 IP 核方法研究》（批准号：61063001）

2、项目来源：国家自然科学基金项目
项目名称：《物联网网络层信息安全体系结构与关键技术研究》（批准号：61262075）

3、项目来源：广西教育厅重大项目
项目名称：《物联网环境下信息传输关键技术研究》（编号:201201ZD012）

4、项目来源：农业部饲料质量监督检验测试中心（南宁）、广西壮族自治区兽药监察所、广西壮族自治区饲料检测所、广西壮族自治区畜牧产品质量监督中心
项目名称：《广西壮族自治区水产畜牧产品质量安全监管预警系统》

5、项目来源：桂林理工大学团委科技立项
项目名称：《物联网下设备间通信关键技术研究》

二、在攻读硕士研究生期间获奖情况

（1）2012 年荣获桂林理工大学“优秀研究生干部”
（2）2012 年荣获桂林理工大学大学生科技评审“一等奖”
（3）2012 年荣获广西区研究生数学建模“三等奖”
（4）2012 年荣获国家研究生奖学金

三、读研期间发表论文：

（1）Xiao-hui Cheng，An-ming Xu. A new memory management mechanism based on Embedded system.The 3nd IEEE International Conference on Software Enginee-ring and Service Sciences, Beijing ,China, 348-402, July 2012.(EI 检索号：20123915469166)

（2）程小辉，龚幼民，许安明. 基于马尔可夫链的嵌入式内存预测分配算法. 计算机工程与设计, 2013.09（录用）

致 谢

三年的研究生生活已经接近尾声，在这个阶段中，我开阔了视野，增长了知识，提高了科研能力和动手能力。回忆起研究生阶段的点点滴滴，有太多的人要感谢了。在此，我衷心的感谢所有帮助过我的老师、同学和亲人。

首先我要向我的导师程小辉教授表示诚挚的感谢，程老师从本论文开题到完成过程中，给予了很多宝贵的意见和悉心指导。程老师渊博的专业知识，严谨的治学态度，精益求精的工作作风，诲人不倦的高尚师德，严以律己、宽以待人的崇高风范，朴实无华、平易近人的人格魅力对我影响深远。不仅使我树立了远大的学术目标、掌握了基本的研究方法，还使我明白了许多待人接物、与人处事的道理。在我的印象中，这三年里程老师平均每隔一天就来实验室指导我们学习和科研，询问我们的学习情况和生活情况，并帮助我们解决一些遇到的困难。从他期待的眼神中，我们看到他迫切希望学生们能够学有所成，为社会做出更大的贡献。这么好的老师，怎么不让学生铭记在心，怎么不让学生以此为榜样呢！

我也感谢我的师兄邓昀、巫湘林、李泽球、张良、周茂杰、陈伟、史丹，师姐肖富元、张一哲、张茹等。感谢他们在我的研究生阶段给予的帮助、支持和关怀，帮助我解决了很多困难。同时感谢一起学习、一起科研的同门沈凡凡、窦萌萌、王小烨和吴铭欢，感谢他们给我留下了一段快乐时光。也要感谢何军权、魏力、梁启亮、康燕萍、郑雅莉、顾俊杰、阮忠海、何劲舟等师弟师妹给我的帮助和无限欢乐。还要感谢室友宋智辉、成阳等，感谢他们给予的帮助以及创造的和谐融洽的生活环境。

最后，我要感谢我的家人，特别是我的父母，是你们含辛茹苦地把我培养长大，给我莫大的物质和精神支持，为我创造良好的学习环境，让我不断的深造，使我慢慢学会坚强，感谢你们二十多年来的艰辛付出，没有你们就没有今天的我！