嵌入式多核操作系统关键技术

姜春茂,倪蕴涛,黄春梅

哈尔滨师范大学计算机科学与信息工程学院,哈尔滨 150025

摘要　分析了目前多核操作系统的现状,多核芯片的出现导致了相应操作系统的改变,嵌入式操作系统的一些关键问题被提出。通过研究多核操作系统在实时性调度算法、负载平衡、同步机制中的现状及其存在的问题,提出未来的研究方向。并辅之以分析一个软实时多核操作系统 LITMUS,对其基于多核体系结构的嵌入式多核平台的应用给出了基本的研究设想。

关键词　多核操作系统;全局实时调度算法;最早截止时间优先调度

中图分类号　TP316.2　　**文献标识码**　A　　**doi**　10.3981/j.issn.1000-7857.2012.11.010

Key Technique for Embedded Multicore OS

JIANG Chunmao, NI Yuntao, HUANG Chunmei

College of Computer Science and Information Engineering, Harbin Normal University, Harbin 150025, China

Abstract　At present, the emergence of multicore chip bring about the change in the computer operating system, many multicore technologies are rapidly come out not only in the application software development but also in the operating system research. In order to provide overall review of researchers on multicore operating system, the common problems that the current mainstream multicore operating systems are facing are analyzed; these problems include real-time scheduling algorithm, load balance, synchronization mechanism, task model, and their applications in the embedded domain. Finally, application examples based on multicore operating system—LITMUS are given by means of the discussion on the system structure, design method, basic principles, and scheduling algorithm, etc. Through the analysis on some key issues in the solution method, the basic idea for transplantation and scalability in the field of multicore embedded problems is given, at the same time, the blueprint for the future direction of the research is proposed.

Keywords　multicore OS; global real-time scheduling; earliest deadline first

0　引言

目前无论是桌面系统还是嵌入式设备,多核都成为不可回避的主题。由于单核芯片固有的缺陷,多核芯片的使用越来越广泛,然而随着硬件的迅猛发展,与多核芯片相配套的软件的发展却严重滞后。多核芯片的普及给计算机产业带来了第三次危机,它对以往所掌握的软件知识产生了极大的冲击,首当其冲的就是操作系统的革新。

目前的多核操作系统一般产生于两类:一是扩展其伸缩性,改进传统的操作系统以适应多核平台;二是面向多核平台进行全新的操作系统设计。无论哪种方式,都面临着实时调度、同步机制等诸多问题。国内的诸多机构,如浙江大学的软硬件协同研究中心[1-2],对多核操作系统的实时性进行了研究,他们使用任务划分和功能划分,通过两个实例说明了实时性的改造,并为此申请了专利。在国外,北卡罗纳大学的实时工作团队提出了基于 Linux 内核的实时化多核平台的操作系统,很好地实现了 LITMUS(Linux Testbed for Multiproccessor Scheduling in Real-Time Systems)系统[3],以插件的方式实现了 EDF (Earliest Deadline First),G-EDF,G-NP-EDF 等典型的软实时调度算法,开发了部分用户 API,并且对负载平衡、同步等进行了很好的研究。

作为嵌入式多核操作系统,要保证系统的实时性、负载问题和核间同步机制等。针对异构多核,Tomiyama 等[4]开发了一个实时嵌入式多核操作系统 uITRON, 并在实时性等问题上进行了有益的研究。

对于嵌入式多核操作系统,实时性的调度算法、负载均衡、同步机制等问题越来越重要。基于特定的系统平台,进行多核操作系统嵌入式的移植改造,对实时调度算法、系统的伸缩性、负载均衡及其同步协议等进行研究,显得很有意义。

收稿日期:2012-02-06;修回日期:2012-04-06

基金项目:黑龙江省自然科学基金项目(F201139);黑龙江省省级教育厅智能检测工程研发中心项目;黑龙江省机器视觉智能检测校企业共建工程研发中心项目

作者简介:姜春茂,副教授,研究方向为嵌入式系统软件、移动 P2P、云计算,电子信箱:hsdrose@126.com

1　国内外研究现状

多核处理器的发展趋势由双核到四核，在未来 5 年很可能出现 10 个甚至上百个核。在多核环境下,应用程序频繁和操作系统交互,而操作系统服务使用共享数据结构,然后通过多核处理器进行处理,这样程序的性能就受制于操作系统。

从图 1 可以看出目前嵌入式多核操作系统研究的几种思路:一是通过改造原有的通用 OS 成为多核操作系统,再针对嵌入式平台进行修改;二是重新开发多核操作系统,再改造成嵌入式多核操作系统;三是针对目前的嵌入式操作系统进行修改使之符合多核平台的要求。

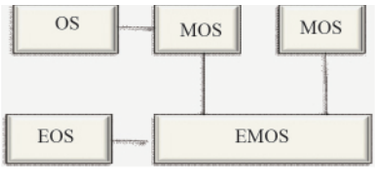

图 1　嵌入式多核操作系统研究路线

Fig. 1　Research roadmap for multicore embedded operating system

Linux 的开源特质使得基于 Linux 的嵌入式多核操作系统产生很多变种[5],通过改造原有操作系统增加多核支持来达到目的,比较著名的有 RTLinux、RTAI、Xenomai 等。本文将要分析的 LITMUS 也是基于 Linux 改造的多核实验平台。

RTLinux 是著名的研究机构 FSMLab 研发的一款实时 Linux,它使用双内核结构,即把 Linux 内核作为新实现的子内核的普通核,子内核位于 Linux 内核和硬件抽象层之间,实时任务运行于子内核之上，只有当没有实时任务需要运行时,Linux 内核才有机会运行。对中断的管理而言,它采用软件的方式处理 Linux 内的中断关闭。当 Linux 内核关闭中断后,并不是真正地屏蔽了硬件中断,相反,它使用了一个变量保存 Linux 内核的中断标志位,Linux 内核的开关中断只是影响了该变量,硬件的中断由子内核来接管。当 Linux 内核关闭了中断,子内核仍然可以响应任何中断,只有子内核不需要处理的中断才交给 Linux 内核处理。在 RTLinux 里,每一个实时任务都是内核线程,运行在内核空间,RTLinux 提供了一套专门的机制在实时任务和普通的 Linux 任务之间进行进程间通信。从调度的角度来说,RTLinux 是一个基于固定优先级的抢占式调度算法,缺少核间通信的机制[6]。为了解决 RTLinux 在各版本 Linux 之间的一致性问题,RTAI 的调度算法相对于 RTLinux 有了很大变化,针对单核和对称多核(SMP)形成了不同的调度策略,且针对核间的通信提供了一组 APIs,满足诸如 FIFO、mailbox、message queue 等消息传递机制。

多核体系结构的不同,比如核间异构、核间拓扑结构甚至采用网络拓扑形式,以及高速缓存的一致性等问题都为多核操作系统的开发提出了很大的难题。从资源的观点出发,考虑异构或者共享 Cache 的改造[7]等算法可提高多核系统的性能。从任务调度机制的角度出发,细化访问的粒度和多级队列,以及调度域机制的改进都极大提高了原有操作系统对于多核平台的适应性。

Hurricane 是多伦多大学设计的基于分集簇集的内核的操作系统。Hive 是基于内核划分的面向共享内存的操作系统,它将硬件资源划分为元包(cell),提高了可伸缩性。Disco 是美国斯坦福大学为缓存一致的非一致性内存寻址（cc-NUMA）处理器 flash 而设计的操作系统，采用虚拟化技术。Corey 是针对目前市场上主流的多核处理器的架构，即缓存一致、内存共享而研究的新操作系统抽象结构。FOS 是 MIT(麻省理工学院)研究的一个多核操作系统,设计思想是利用空间复用取代时间复用。Barrelfish(黑舵鱼操作系统)设想从网络的角度考虑每个核,把每个核单独看成一个机器节点而开发出全新的分布式操作系统[8]。

Atom 处理器的功耗问题一直是困扰 Intel 的问题，为此与 Nokia 公司合作开发了 Moblin 系统，新名字是 Megoo[9]。Corey 还是一个不完整的原型，目前还不具备与功能完善的操作系统相比的能力。FOS 和 Barrelfish 目前还处在对多核 PC 或 HPC 的研究阶段,Barrelfish 团队于多核操作系统的调度、通信等技术有很好的设想[10],如图 2 所示。

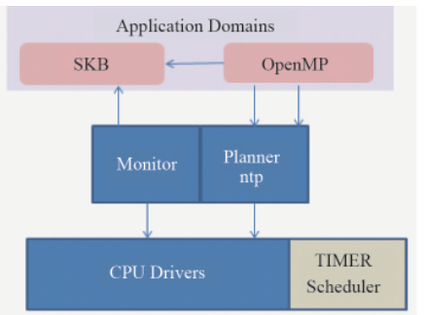

图 2　黑舵鱼系统结构图

Fig. 2　System structure for Barrelfish

针对多核处理器的嵌入式操作系统研究相对缓慢,支持多核处理器的嵌入式实时操作系统主要有 WindRiver 公司的 Vxworks[11]、QNX 软件系统有限公司的 Neutrino[12],以及由 Linux 发展而来的 MontaVista Linux[13]等少数操作系统。多核处理器的发展对操作系统提出了新的挑战,因此,支持多核的操作系统的研究成为热点。Kandemir 等[14]研究了多核情况下的任务调度问题,强调在多处理器片上系统(MPSocs)中,Cache 的利用对于系统性能的提高和功耗降低具有重要意义。它假设将一个循环拆成多个线程,计算拆分后的线程的共享程度，将有共享数据的线程先后调度在同一个核上,将不共享数据的线程调度到不同的核上。此方法不仅充分利用了 Cache 的性能,而且利用了多核的并行计算能力。在系统性能方面,Fedorova 等[15]提出,二级缓存的不命中要比一级缓存不命中和流水线的冲突造成更大的影响。通过合理的调度来减少线程对二级缓存的访问,可以提高系统的吞吐量。为了保证实时性和利用共享 Cache 带来的优势,Anderson 等[16]提出了用于多核实时操作系统平台的组调度策略。在降低功耗

方面,Gontmakher 等[17]提出了 Intheads 结构——一个极端细粒度的线程结构,通过在编译时显式的标记方式来实现并行执行。它认为某些程序利用 Inthreads 在顺序执行的处理器上执行比程序直接在乱序执行的处理器上执行消耗更多。

Enea[18]的特点是简便、灵活、透明和能方便地调试应用程序,是广泛应用于通讯领域的实时操作系统。OSE 采用了高级消息传送编程模型,使复杂的应用程序更易于概念化、模块化、分区和调试。它所具有的透明度可以将应用程序与基层硬件和物理拓扑结构的详细信息分离开来,使结果代码更具可移植性和可扩展性。Enea 的 LINX 进程间通信(IPC)服务将其消息传送优势延伸到了多个处理器和操作系统中的 OSE 应用程序上。LINX 可以在一个单 CUP、多核或者一个分布式系统的不同节点上准确无误地连接 OSE 和 Linux 操作系统。LINX 使复杂的应用程序更容易分割及分布。LINX 同样使得生成的代码更方便地扩展和维护,使系统开发者扩展了他们的系统,进行升级,并以对现有应用程序微弱的影响利用最新硬件来突显优势。Enea 具有很好的鲁棒性。一个内核对每一个实例相关的数据结构的核心进行单独调度,以维护确定性和实时特性,在用户自定义进程迁移和负载平衡的基础上来衡量每个内核的 CPU 负载。

浙江大学嵌入式中心多年来致力于偏硬件的多核嵌入式系统解决方案研究,提出了多核嵌入式系统实时性改造方案,主要分为任务并行和操作系统支持两部分。任务并行主要利用多核的资源进行基于功能和数据的划分,通过比较划分前后在多核上的运行性能,总结出合理的划分方法以提高实时任务的性能。改造方案还包括在操作系统中设计了基于二级缓存共享的线程分配算法,提供划分后任务的运行支持。该所提出了一种多内核操作系统技术,满足嵌入式系统的混合应用多样化需求(主要是满足软实时),并且利用该多内核技术设计了构件化的操作系统 Pcanel。国内的大唐公司发明了一种嵌入式实时操作系统中多核处理器的核间通信方法,包括在共享内存区中分配内存作为多核共享消息池,源核将要发送的消息写入多核共享消息池,源核将所述消息在多核共享消息池中的地址、目的任务标识通过数据管道发送给目的核,目的核将多核共享消息池中所述地址存放的消息传送给目的任务。本发明还公开了一种核间通信装置。国内很多研究机构也做了类似的体系结构方面的研究和实践。

LITMUS 项目[19]是一种软实时操作系统,通过改造 Linux 平台来搭建一个能够进行实时调度算法(算法主要针对零星任务数据的实时性要求而设计开发)、负载平衡、共享资源管理等系统测试的研究平台,刚开始是针对零星任务模型[19]。在该平台上已经产生了一批成熟的算法,而且研究者正在将其移植到类似于 ARM 体系结构的多核平台上。

1.1　调度算法和负载平衡

嵌入式多核平台以较低的频率来实现较高的性能,又能明显降低系统的功耗,但同时带来了诸如实时性及其调度算法等一系列问题。

目前实时系统的任务调度方法主要有固定优先级调度和动态优先级调度。其中固定优先级调度算法比较成熟的有 RM(Rate Monotonic)调度。1973 年,Liu 和 Leyland[20]提出了适用于可以抢占的硬实时周期任务的静态调度算法。在此基础上发展了 DM 算法,解决了任务周期小于等于截止期限时 RM 调度的局限性问题。动态优先级的调度算法主要包括 EDF、PD (Predictive Deadline)、TBS (Total Bandwidth Server)等[21],在一定程度解决了负载过重的问题。在多核平台上任务调度算法通常采用 EDF-FF (First Fit),PD^2 (a global Pfair algorithm)算法已经成为替代者。Pfair 算法是一种按比例分配时间的调度算法,Anderson[1]在其基础上提出了 ERfair(Early Release Fair)调度,建立了零星任务模型,允许同一任务的不同子任务之间有时间间隔。

从调度队列的角度出发,有两种典型的范例[22]——全局调度和局部调度。在局部调度中,任务(进程)被指定在特定的核上运行,不允许发生迁移;而在全局调度中,允许进行进程的迁移。可以看出,全局调度更能发挥系统的核能力,但是增加了通信和线程迁移的成本[23]。

从调度的模型来看,有从任务属性划分的,如零星任务模型;也有从调度的属性出发的,比如温度敏感的,或者自悬挂 (Self-Suspension, 实时系统中两种锁机制——Spinning Lock 和 Suspension Lock)。

零星任务模型(Sporadic Task Model)是一种周期性任务。在硬实时系统中,一个常见的零星任务的例子是自动驾驶仪控制飞机的飞行控制系统。不可能准确确定某些关键控制操作何时会发生,但当它们发生时,系统必须在最后期限内响应。如果飞行员关掉自动驾驶仪,而系统用了不适当的大量时间将飞机的控制交给飞行员,这肯定会被认为是错误的情况。

1.1.1　任务模型

定义 1　零星任务指在两个相邻的发布之间的时间间隔长度总是大于或等于一个常数(它本身不为零)的任务。即零星的任务。

定义 2　执行成本函数:将一组调度任务的总体的完成时间定义为执行成本。

当调度实时任务时,需要计算任务的执行成本,以作调度决策。总的完成时间为

$$t_c=\max(f_i)-\min(a_i) \tag{1}$$

其中,f_i 表示最后任务完成的时间;a_i 表示第 1 项任务开始的时间。该成本函数计算一组任务的总的完成时间,用最后的任务完成的时间减去第 1 项任务开始的时间;这些可以是不同的任务。

定义 3　迟的任务

$$\text{late}(t_i)=\begin{cases}1 \ if\ d_i>0 \wedge f_i>d_i, else\ 0\\ 1 \ if\ D_i>0 \wedge C_i>D_i, else\ 0\end{cases} \tag{2}$$

如果该任务的结束时间超过了其绝对的最后期限,或者,这项任务的完成时间超过其相对的最后期限(这取决于任务的最后期限类型),那么这个函数返回值为 1,表明是真。

迟的任务数目为

$$N_{late}=\sum_{i=1}^{n}\text{late}(t_i) \quad (3)$$

式(3)计算了在某 n 项任务的实时系统中错过它们的最后期限的任务总数。为了简化方程和未来的讨论,定义了一项任务的最后期限 d_i。当任务有绝对最后期限时,d_i 等于其绝对最后期限;当任务有相对的最后期限时,$d_i=a_i+D_i$。

最迟,$L_{max}=\max(f_i-d_i)$。此函数使用上文定义的 d_i 计算任务的最迟(以最大数量的时间错过了最后期限)。这个值可以是负的(当所有任务在最后期限前完成),或是正的。

平均响应时间为

$$R_{avg}=\frac{1}{n}(\sum_{i=1}^{n}C_i) \quad (4)$$

式(4)计算了给定实时系统的平均响应时间,即所有任务完成时间的总和(C_i)除以系统中的任务总数。

零星调度有其局限性,多媒体类型的数据或许并不属于这类问题[21],它们一般被称为流任务模型。流任务模型的提出促使了网络积分学在实时系统上的应用,开创了实时系统积分学,提出了按照时间属性和资源属性进行积分[23]。

1.1.2 负载平衡

负载平衡的研究和调度是不能分开的,例如可能由于负载平衡导致实时性的缺陷,如何区别任务类型也是一个重要问题。是否考虑异构的体系结构使得在实时核心上运行实时任务,在非实时核心上运行非实时任务,是一个问题。如果系统的实时任务很少,大量的任务是非实时任务,那么实时内核处于空闲状态,反之则可能影响系统的实时反应时间。

有很多单纯针对多核平台进行的任务负载研究[22,24-26],这类似于运筹学类的问题。但这些大多数都不合适于嵌入式实时多核的条件。

Zhang 等[27]研究了基于 Linux 的线程迁移机制,对于实时系统而言,由于系统不可识别任务的调度属性,动态线程平衡可能影响系统的实时性能,而且线程迁移的成本是无法忽略的;同时提出了使用实时中间件的解决方式。

Fedorova 等[28]对异构多核操作系统的调度进行了研究,对于多核系统而言,不当的调度会导致抖动现象;同时提出了 3 个要素——最优性能、内核指派平衡、反应时间最佳的算法,并且对此进行了证明。可以看出,Fedorova 等提出的操作系统模型经过改造有可能适合嵌入式多核,同时满足实时性的要求。

资料显示,LabEIVW 为实时嵌入式硬件引入多核性能,利用嵌入式多核系统的特性进行最优化的并行式编程。LabVIEW 8.5 软件为确定性实时系统引入了台式机的自动多线程调度器(SMP)。LabVIEW 8.5 的实时模块加入了多核系统支持,在嵌入式实时系统中,在多个核上自动进行负载均衡。实验和算法还有待研究。

对于时间关键(time-critical)的代码,可以将定时循环分配到指定的处理器上,将定时循环结构中的关键代码与应用程序中的其他代码隔离。

利用 Real-Time Execution Trace Toolkit 2.0 工具,用户可以方便地对 VI 程序运行过程中的线程和处理器核进行图形化表示,以更好地调整实时系统,进而获得最佳性能。

1.2 同步机制的研究现状

多核环境带来了进程的同步问题。由于进程在不同的执行核上运行,其同步不仅仅是线程的同步,而有可能是执行核或 CPU 之间的同步。调度策略的合理选择与执行对于保证系统整体运行效率至关重要。在多核环境下,可以使用一种硬件原子操作——总线锁。由于 CPU 需要使用共享总线来访问内存,而总线的锁住使其他处理核没有办法执行任何与共享内存有关的指令,从而保证了数据的访问是排他的。在硬件提供同步原语的基础上,可以构建软件同步原语。由于多核技术出现时间不长,如何实现多处理器的同步尚没有统一标准,因此各个厂家实现的方法不尽相同。在 Linux 中提供总线锁、原子算术、原子位等操作。

旋锁是几乎所有多核操作系统采用的互斥技术,通常用于保护某个全局数据结构。旋锁通过获取和释放两个操作来保证任何时候只有 1 个拥有者。旋锁的状态有两种,闲置或被某个 CPU 拥有。另外自旋锁不允许任务睡眠(持有自旋锁的任务睡眠会造成自死锁,因为睡眠有可能造成持有锁的内核任务被重新调度,而再次申请自己已持有的锁),它能够在中断上下文中使用。无论是自旋锁还是挂起锁都是可以继承的。Mellor-Crummey[29]对 FIFO 队列自旋锁任务进行了研究,自旋锁适合和内核交互量较小的任务系统。相对而言,挂起锁适合于通过系统调用方式使用共享资源的操作系统同步协议[30]。目前倾向于将两者混合进行研究[31]。无论哪种方式,它的可分析性很重要,即检查调度性时可以分析出它的代价或者成本。Chen 等[32]提出了一种基于部分(分区)静态优先级的 PCP(Priority-Ceiling Protocol)协议。Lopez 等[33]提出了针对 P-EDF 算法的协议,局限性在于它是基于周期性任务模型(非零星)的。Gai 等[34]进一步解决了这种局限性。由北卡罗来纳州立大学[3]提出的 FMLP 协议不再限制临界区的类型,能够支持 G-EDF 和 P-EDF。在 FMLP 协议中,资源被自旋锁或者悬挂锁保护。FMLP 协议是目前唯一支持 G-EDF 算法的任意临界区协议。进一步的原理分析在实例分析中给出。

2 应用实例研究

美国北卡罗纳州立大学的 LITMUS 项目,旨在在多核平台上,通过扩展原有的 Linux 操作系统以支持实时任务。项目本身并不是要形成一个产品,而是使用这个平台来测试所开发的算法。项目的实时并非硬实时,而是软实时。虽然基于 Linux 的多核实时化改造了很多年,已经形成了诸如 RTLinux、RTAI 等[5]系列基于 Linux 的多核操作系统。但多数的变种没有实现最新的实时调度算法。

最新的 LITMUS 是基于 Linux 2.6.36 核心,允许不同的实时调度算法以插件的方式(Plug-in Components)加以运行和

实践。同时,实现了一种新的多核实时锁机制。

2.1　LITMUS 的体系结构

LITMUS 项目的最终目标是建立一个稳定的，对针对多核操作系统的复杂实时应用系统进行算法实验的平台,通过对调度算法、同步机制等进行研究,为实时多核的研究打下坚实的基础。

具体的实现方式是：通过改变标准 Linux 的内核、调度器、中断功能来增加插件,以静态实时最高的优先级别调度。用拥有最新调度算法的插件来模拟实时任务[35],当在系统启动的时候,以命令行的方式启动实时内核。内核的修改包含 3 个组件。在零星任务模型的基础上实现实时调度,具体的系统构成见图 3。

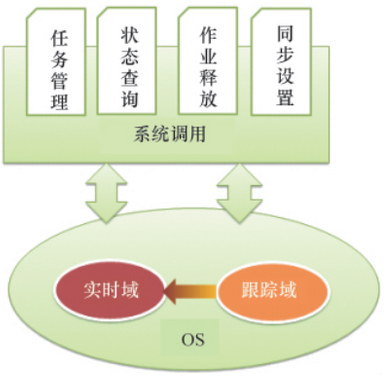

图 3　LITMUS 系统结构图

Fig. 3　System structure for LITMUS

在设计和实现上采用模块化的设计方法。在实时域(RTD)中使用 rt_domain_t 函数模块实现,分为就绪队列和释放队列。在具体实现上,将原有 Linux 系统的 list.h 函数扩展为 list_insert()和 list_qsort()函数,进一步包装成 sort 和 wrapper 函数进入队列。

在跟踪域中,使用 trace()函数替换 printk 函数,使用 featherTrace()函数记录细粒度的事件代价。通过系统调用和 OS 内核进行系统交互[36]。

2.2　实时调度算法

多核实时调度算法可以分为两类。一类是通过对任务进行划分来划分任务集，使自己任务的不同在不同的核上运行,这个问题属于 NP-hard 问题,最好的方法是找到近似算法。另一类是按调度的范围分为全局调度和局部调度。前者是 NP 完全问题,但可以在每个队列上进行单核的调度算法;后者是全局一个队列,所带来的 cost 可能是迁移代价。

目前实现的调度算法包括 PEDF(Partitioned EDF)、GEDF & NP-GEDF 算法 (Preemptive & Non-preemptive Global EDF)、CEDF 算法 (Clustered EDF)、PD^2 算法、S-PD^2 算法 (Staggered PD^2)。

CEDF 算法(簇调度)是将多核按照一定簇组织成不同的调度单元,运行前静态的指定任务到簇,每个簇独立调度各自拥有的作业队列。运行时按照优先级进行调度,允许作业在簇内迁移。簇内作业可以按照 EDF(可以按照递增截止期限升序)、FP(固定优先级)的方法进行排序。当每个核心指定一个队列时,算法退化为分区调度模型。当将所有核心作为一个簇的时候,算法退化为全局调度算法。簇越大,Cache 争用现象越严重,最后导致高代价。内核开销是任务数量的函数,Cache 命中率是工作集大小的函数。对于 HRT,采用最坏开销设计;对于 SRT,采用平均开销设计。

采用单调逐段线性差值的方法分析分区调度算法和簇调度算法的优劣。

对于每一种调度,使用超过 200 个任务子集,每个核心至少使用 20 个跟踪程序，执行时间超过 110h，收集了超过 500GB 的原始样本。在 LITMUS 实验平台上进行算法的测试,调度算法以插件的方式被扩展入该系统。结果如表 1 所示。

表 1　算法在硬实时和软实时下反应时间比较

Table 1　Comparison of reaction time for the algorithms between the hard and soft real-time

调度算法类型	HRT	SRT
PEDF	Util loss	Util loss
GEDF	Util loss	No loss
NP-GEDF	Util loss	No loss
CEDF	Util loss	Util loss
PD^2	No loss	No loss
S-PD^2	Slight loss	No loss

测试平台使用的是 Sun Niagara(UltraSPARC T1)。该平台有 8 核,每核有 4 个线程,使用 16KB 的 LI Cache,共享 3MB L2 Cache 32 位操作系统。1.2 GHz RISC 指令相对简单,没有指令分支预测技术,相对于 Intel 而言,Cache 较小。使用的操作系统为 LITMUS。

对于硬实时任务而言,PEDF 通常效果最好,S-PD^2 次之,PD^2 和 GEDF 很差。而对于软实时任务而言,PEDF 的效果不如硬实时任务好，但对于轻量级的任务尚可,CEDF 效果最好,S-PD^2 性能其次,GEDF 一般。详细的评测结果可以参考文献[37]—[38]。算法对于如何实现的共享队列方式很敏感,目前实验是链表和二叉堆的方式。

2.3　同步调度算法研究

多核系统中,代码可以在多个内核上同时执行,这意味着如果不对线程进行同步,就有可能发生各个线程之间相互覆盖共享数据的情况，造成访问数据处于不一致的状态,这是使系统不稳定的一大隐患。为了解决这一问题，自旋锁、读-写锁、屏障锁、信号量、邮箱、消息等数据结构都可以被同步机制利用,实现内核的同步。

一个理想的多核同步算法,尤其是对于众核(many core)系统,应具备以下特征。(1) 细粒度。一个并行系统中能够开发的并行度受同步的粒度限制,粗粒度不具有可扩展性。(2) 低延迟。如果延迟支配着执行时间则同步算法是不可用的。(3) 无竞争。理想的同步算法应是无竞争的。(4) 可扩展性。

理想的同步算法要求同步规模能够随着系统中核数量的增加而提高。(5) 适应性。理想的同步算法是和应用无关的。

LITMUS 在同步机制中,设计开发了 Flexible Multiprocessor Locking Protocol (FMLP)[39]。FLMP 协议不再区分临界区(CS)的类型,能够支持 G-EDF 和 P-EDF。在这个协议中,资源被自旋锁或悬挂锁(Suspension Lock)锁定。系统区分长资源和短资源,短作业使用队列锁,长作业使用信号量协议[30]。

FLMP 协议之所以被称为灵活的多核锁同步协议,是因为它既可以在全局调度算法下,也可以在分区调度算法中使用。在忙等与挂起之间找到平衡的方法有 3 种:(1) 忙等→短作业,其中长短由用户自己设定,申请使用短资源就不可以申请长资源;(2) 短作业不可以抢占的作用采用忙等;(3) 按长短群组资源。

请求规则为:长资源使用信号量,短资源使用非抢占队列锁。设 G_R 为一个资源组。

$l1, l2 \in G_R \Leftrightarrow$ (存在一个 job,请求 l1 包含在请求 l2 中,(即先请求 l2 再请求 l1)且 l1、l2 要么均为 short 型,要么都为 long 型。

最外层的请求与非嵌套资源被定义成与访问资源的类型相关,例如:请求非嵌套长资源的可以包含段资源请求,且段资源请求只能被包含在最外层场资源的请求内。为避免冲突,long outmost 标为 l_outmost 或者 s_outmost。

在队列锁中,被锁进程忙等按 FIFO 顺序,在企图获得那个锁的时候,job 首先必须成为非抢占的并且必须保持原有状态直到它放弃锁。

FMLP 被证明是无死锁的。目前来看该系统还面临如下的挑战。

(1) 时钟对齐问题。

一些多核实时系统调度算法如 PD^2 需要同步机制支持,它们的正确运行有赖于多核时钟中断的同时发生。如何对齐及其对齐因子的设定依然面临问题。

(2) I/O 支持。

多核实时调度分析还要产生对 IO 时间进行分解的有效方式,除此以外,要支持 IO 关键业务的实时性,比如在此平台上稳定快速的存储登陆数据。

2.4 LITMUS 在 ARM A9 多核芯片上的移植

Cortex-A9 处理器提供了具有高扩展性和高功耗效率的解决方案。利用动态长度、八级超标量结构、多事件管道及推断性乱序执行 (Speculative Out-of-order Execution),Cortex-A9 处理器能在频率超过 1GHz 的设备中,在每个循环中执行多达 4 条指令,同时还能减少目前主流八级处理器的成本并提高效率。

通过对 MPCore 技术作进一步优化和扩展,Cortex-A9 MPCore 多核处理器的开发为许多全新应用市场提供了下一代的 MPCore 技术。此外,为简化和扩大对多核解决方案的使用,Cortex-A9 MPCore 处理器还支持与加速器和 DMA 的系统级相关性,进一步提高性能,并降低系统级功耗,在 250mW 移动功耗预算条件下为手机提供性能显著提升的可综合 ARM 处理器。

基本的移植方法如下:(1) get Linux 2.6.36;(2) apply the LITMUS RT patch;(3) make menuconfig;(4) compile the kernel;(5) proceed to install kernel;(6) minicom。

2.5 LITMUS 存在的主要问题

目前,LITMUS 系统还处在研究阶段,实现了一些算法,提高了多核平台的利用率和实时性,极大提高了吞吐量。存在的主要问题和需要进一步研究的内容如下。

(1) 对算法而言,如何完全实现共享队列,对于 HRT/SRT 混合的负载如何检查,同步以及动态行为的因素有哪些都尚待研究。

(2) 对平台而言,目前的实验平台不是嵌入式的平台。在嵌入式的平台上目前算法可能出现的问题,以及是否需要对特定的平台,如 ARM、INTEL-ATOM 平台进行格外的优化处理,或者考虑新的算法并且实现还有待研究。嵌入式平台的体积、速度、功耗等问题都将为移植提出挑战。

(3) 伸缩性问题也亟待研究。Intel 宣布核的数目将很快达到 80 个[38-39],这么多核的结构形式,是对软件层面的操作系统提出的新课题,目前的算法是否存在良好的伸缩性都有待研究(从定量和定性的角度进行研究)。

3 结论

嵌入式多核系统将越来越普及,针对此平台进行的操作系统的实时调度算法、负载平衡及其同步机制的研究有重大意义。基于 LITMUS 平台进行的测试和分析,为今后的研究提供了非常好的基础,它以插件化、简单化的实验路线对研究分析工作提供了可能。随着硬件技术的发展,针对嵌入式多核的应用也将越来越广泛,不仅局限在操作系统层面,应用软件层面的研究也已经展开。

参考文献(References)

[1] 李磊,沈海斌,严晓浪. 多核设计中的共享与分布式存储系统[J]. 江南大学学报: 自然科学版, 2008(3): 253-257.
Li Lei, Shen Haibin, Yan Xiaolang. *Journal of Jiangnan University: Natural Science Edition*[J]. 2008(3): 253-257.

[2] 刘加海,杨茂林. 基于多核处理器平台的公平调度算法[J]. 浙江大学学报: 工学版. 2011(9): 1566-1570.
Liu Jiahai, Yang Maolin. *Journal of Zhejiang University: Engineering Science Edition*, 2011(9): 1566-1570.

[3] Brandenburg B, Block A, Calandrino J, *et al*. LITMUSRT: A status report [C]//Proceedings of the 9th Real-Time Linux Workshop, North Carolina, USA, November 12-13, 2007. North Carolina: The University of North Carolina at Chapel Hill, 2007: 107-123.

[4] Hara Y, Tomiyama H, Honda S, *et al*. Chstone: a benchmark program suite for practical c-based high-level synthesis [C]//IEEE International Symposium on Circuits and Systems, Seattle, WA, United States, May 18-21, 2008. New York: IEEE, 2008: 1192-1195.

[5] Yiqiao P, Qingguo Z, Kairui S, *et al*. Various freed multi-cores RTOS based Linux, 2008 [C]//Proceedings of 2008 IEEE International Symposium on IT in Medicine and Education, Xiamen, China, December 12-14, 2008. New York: IEEE, 2008: 900-905.

[6] Sato H, Yakoh T. A real-time communication mechanism for RTLinux[C]//26th Annual Conference of the IEEE Electronics Society, Nagoya, Japan, October 22-28, 2000. New York: IEEE, 2000: 2437-2442

[7] 卢宇彤, 杨学军, 所光. 一种面向多核系统的并行计算任务分配方法[J]. 计算机研究与发展, 2009, 23(3): 359-364.
Lu Yutong, Yang Xuejun, Suo Guang. *Journal of Computer Research and Development*, 2009, 23(3): 359-364.

[8] Baumann A, Barham P, Dagand P E, *et al*. The multikernel: A new OS architecture for scalable multicore systems, 2009 [C]//Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles, Big Sky, MT, United states, October 11-14, 2009. Mohamed Zahran: ACM SIGOPS, 2009: 29-43.

[9] Imad Sousou. What's next for MeeGo. moblin.org[EB/OL]. [2012-03-28]. http://moblin.org/.

[10] Peter S, Sch U, Pbach A, *et al*. Design principles for end-to-end multicore schedulers, 2010 [C]//HotPar'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Parallelism, CA, USA, May 4, 2010. Boston, MA: USENIX, 2010: 10-13.

[11] Fedorova A. Wind river VxWorks RTOS — the world's #1 RTOS for embedded real time systems[EB/OL]. [2012-03-28]. http://www.windriver.com/products/vxworks/.

[12] Krten R. QNX neutrino RTOS[EB/OL]. [2012-03-28]. http://www.qnx.com/products/index.html.

[13] MontaVista software [EB/OL]. [2012-03-28]. http://www.mvista.com/real_time_linux.Php.

[14] Kandemir M, Chen G. Locality-aware process scheduling for embedded MPSoCs, 2005[C]//Proceedings-Design, Automation and Test in Europe, DATE '05, v II, Munich, Germany, March 7-11, 2005. Munich, Germany: IEEE Conference Publications, 2005: 870-875.

[15] Fedorova A, Seltzer M, Small C, *et al*. Throughput-oriented scheduling on chip multithreading systems, Technical ReportTR-17-04 [R]. Cambridge, USA: Harvard University, 2004.

[16] Anderson J H, Calandrino J M. Parallel task scheduling on multicore platforms [J]//Proceedings of 2006 IEEE 27th Real-Time Systems Symposium, Rio de Janeiro, Brazil, December 5-8, 2006. New York: IEEE, 2006.

[17] Gontmakher A, Mendelson A, Schuster A. Using fine grain multithreading for energy efficient computing, 2007 [C]//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, CA, United States, March 14-17, 2007. Mohamed Zahran: ACM SIGPLAN, 2007: 259-269.

[18] Software-Enea software [EB/OL]. [2012-03-28]. http://www.enea.com/software/.

[19] LITMUS RT: Linux testbed for multiprocessor scheduling in real-time systems[EB/OL]. [2012-03-29]. http://www.litmus-rt.org/.

[20] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment [J]. *Journal of the ACM* (*JACM*). 1973, 20(1): 46-61.

[21] Palopoli L, Abeni L, Cucinotta T, *et al*. Weighted feedback reclaiming for multimedia applications, 2008 [C]//6th Workshop on Embedded Systems for Real-Time Multimedia, Atlanta, GA, USA, October 23-24, 2008. Texas: IEEE Conference, 2008.

[22] 袁云, 邵时. 基于多核处理器并行系统的任务调度算法[J]. 计算机应用, 2008, 28(2): 22-24.
Yuan Yun, Shao Shi. *Journal of Computer Applications*, 2008, 28(2): 22-24.

[23] Leontyev H. Compositional analysis techniques for multiprocessor soft real-time scheduling [D]. North Carolina, USA: University of North Carolina, 2010.

[24] 李实, 刘乃琦, 郭建东. 多核架构下的多线程负载平衡 [J]. 计算机应用, 2008, 28(12): 139-140.
Li Shi, Liu Naiqi, Guo Jiandong. *Journal of Computer Applications*, 2008, 28(12): 139-140.

[25] 刘轶, 张昕, 李鹤, 等. 多核处理器大规模并行系统中的任务分配问题及算法[J]. 小型微型计算机系统, 2008, 29(5): 972-975.
Liu Yi, Zhang Xin, Li He, *et al*. *Journal of Chinese Computer Systems*, 2008, 29(5): 972-975.

[26] 姚震. 并行程序设计模型若干问题研究 [D]. 北京: 中国科学技术大学, 2006.
Yao Zhen. Study on parallel programming models[D]. Beijing: University of Science and Technology of China, 2006.

[27] Zhang Y, Gill C, Lu C. Real-time performance and middleware for multiprocessor and multicore linux platforms [C]//2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009), Beijing, China, August 24-26, 2009. New York: IEEE, 2009.

[28] Fedorova A, Vengerov D, Doucette D. Operating system scheduling on heterogeneous core systems [C]//Proceedings of the First Workshop on Operating System Support for Heterogeneous Multicore Architectures. Brasov, Romania: PACT, 2007.

[29] Mellor-Crummey J M, Scott M L. Algorithms for scalable synchronization on shared-memory multiprocessors[J]. *ACM Transactions on Computer Systems* (*TOCS*), 1991, 9(1): 21-65.

[30] Lim B H, Agarwal A. Waiting algorithms for synchronization in large-scale multiprocessors [J]. *ACM Transactions on Computer Systems* (*TOCS*), 1993, 11(3): 253-294.

[31] Sha L, Rajkumar R, Lehoczky J P. Priority inheritance protocols: An approach to real-time synchronization[J]. *IEEE Transactions on Computers*, 1990, 39(9): 1175-1185.

[32] Chen C M, Tripathi S K. Multiprocessor priority ceiling based protocols [R]. Maryland USA: UM Computer Science Department, 1998.

[33] Lopez J, Diaz J, Garcia D. Utilization bounds for EDF scheduling on real-time multiprocessor systems [J]. *Real-Time Systems*, 2004, 28(1): 39-68.

[34] Gai P, Di Natale M, Lipari G, *et al*. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform, [C]//Proceedings the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Ont, Canada, May 27-30, 2003. New York: IEEE, 2003: 189-198.

[35] Anderson J H, Ramamurthy S. A framework for implementing objects and scheduling tasks lock-free real-time systems [C]//17th IEEE Real-Time Systems Symposium (Cat. No.96CB36024), Los Alamitos, CA, USA, December 4-6, 1996. New York: IEEE, 1996: 94-105.

[36] Ramamurthy S, Moir M, Anderson J H. Real-time object sharing with minimal system support [C]//Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, PA, USA, May 23-26, 1996. Texas: ACM, 1996: 233-242.

[37] Wentzlaff D, Gruenwald III C, Beckmann N, *et al*. An operating system for multicore and clouds: mechanisms and implementation [C]//Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, Indianapolis, IN, United States, June 6-11, 2010. Texas: ACM, 2010: 3-14.

[38] Li T, Brett P, Knauerhase R, *et al*. Operating system support for overlapping-isa heterogeneous multi-core architectures [C]//2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), Bangalore, India, January 9-14, 2010. New York: IEEE, 2010.

[39] Block A, Leontyev H, Brandenburg B B, *et al*. A flexible real-time locking protocol for multiprocessors [C]//13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, South Korea, August 21-24, 2007. New York: IEEE, 2007: 42-51.

(责任编辑　安莹,吴晓丽)