基于有限的共享资源模型实现嵌入式硬实时 Linux

余化鹏[1,2],卢显良[1],彭先蓉[2]
(1. 电子科技大学 计算机科学与工程学院,四川 成都 610054;
2. 中国科学院 光电技术研究所,四川 成都 610209)

摘 要:首先分析了采用双核方案实现硬实时 Linux 存在的问题,然后基于有限的共享资源模型提出了一种在单 Linux 内核上实现嵌入式硬实时 Linux 的新思路,并在 Linux 2.6 内核上予以完整实现。在目标平台上实测结果表明,从硬实时任务中断产生到硬实时任务得到调度执行的最大延迟小于 100μs,可以满足绝大多数嵌入式硬实时系统的要求。

关键词:实时系统;嵌入式;硬实时 Linux;资源模型
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)04-0001-04

Implementing Embedded Hard-Real-Time Linux Based on Limited Sharing Resources Model

YU Hua-peng[1,2], LU Xian-liang[1], PENG Xian-rong[2]
(1. Coll. of Computer Sci. and Eng., Univ. of Electronic Sci. and Techn. of China, Chengdu 610054, China;
2. Institute of Optics and Electronics, China Academy of Sciences, Chengdu 610209, China)

Abstract: Analyzed the key problems for implementing hard-real-time Linux based on dual-kernel scheme. Proposed a new scheme to implement hard-real-time Linux on single Linux kernel based on the limited sharing resources model. And implemented it on Linux 2.6 kernel. The actual test result on target platform shows that the maximum latency from the generation of hard-real-time interrupt to the execution of hard-real-time task is less than 100μs, which could meet the need of most of the embedded hard-real-time systems.

Key words: real-time system; embedded; hard-real-time Linux; resources model

0 引 言

嵌入式 Linux 在嵌入式系统中越来越广泛而深入的应用,对嵌入式 Linux 内核的实时改造一直以来也都是一个热门话题。许多改造方案被提了出来[1,2],归结起来,主要是两类方案:一类是修改核的方案;另一类是双核方案。前者主要是通过缩短关中时间、插入更多调度点、实现内核态的可剥夺调度这样一些方法,达到降低任务调度延迟的目的,可以满足软实时(soft real time)系统的要求;后者实际上是采用了虚拟机的思想[3],只不过仅仅虚拟中断控制器,即硬实时核截获硬件中断,并为 Linux 内核提供一个虚拟的中断控制器,这种方案可以满足硬实时(hard real time)系统的要求。

收稿日期:2006-07-19
基金项目:国家 863 计划高新技术预研课题资助项目(2003AA823050);2005 年四川省青年软件创新工程资助项目(505)
作者简介:余化鹏(1973-),男,重庆人,硕士研究生,研究方向为计算机体系结构、嵌入式系统上的系统软件;卢显良,教授,研究方向为计算机操作系统与网络软件。

文中分析了双核方案存在的问题,针对这些问题,提出了一种基于单 Linux 内核实现嵌入式硬实时 Linux 的新思路,并在基于 Linux 2.6 内核的目标平台上予以实现,达到了设计要求。

1 实现硬实时——单核还是双核

首先,需要明确一个概念,什么是硬实时?硬实时意味着,无论系统负载有多重,硬实时任务(Hard-Real-Time Task,以下简称 HRTT)必须在给定的时间(称为 PDLT,见 2.2 中的定义)之内得到响应,并在给定的时间(称为死限)之前完成。文中探讨 PDLT,至于死限,则涉及到可调度性分析,不在探讨之列。

由前面的分析可知,双核方案(包括 RT-Linux 和 RTAI)目前是唯一的硬实时解决方案,大量的文献对其进行了详尽的探讨,大量的硬实时应用成功建立在其上。这种方案通过采用虚拟机的思想巧妙地解决了在一个通用操作系统(如 Linux,Windows)上如何实现硬实时调度这一关键问题。实际上,Windows 的 RTX 插件也是源于这个思想。

然而,事情总是具有两面性的。双核方案存在如下一些问题:

第一,HRTT 不能调用 Linux 系统调用,这意味着 Linux 丰富的系统资源 HRTT 并不能被利用;

第二,系统设计必须面对两个独立的世界:不调用 Linux 系统调用的 HRTT,以及可以调用 Linux 系统调用的一般任务。

可想而知,这两个问题将使得系统设计和实现变得异常复杂,与试图利用 Linux 平台获得高的开发效率的初衷背道而驰。

由此看来,为了符合我们的初衷(利用 Linux 平台获得高的开发效率),应该基于单 Linux 内核来实现嵌入式硬实时 Linux,该 Linux 必须满足如下两点要求:

(1)满足系统的硬实时要求;

(2)兼容 POSIX 的 API。

对于(1),文中根据目标应用系统的要求确定硬实时指标为:Max(PDLT)$<100\mu s$。对于(2),要求实现的嵌入式硬实时 Linux 不增加任何额外的特殊系统调用,从而保证应用程序开发的完全兼容性。

下文即通过分析 Linux 2.6 内核的特点和嵌入式系统的特殊性,提出了一种可行方案,并予以实现。

2 有限的共享资源模型

2.1 背 景

Linux 2.6 内核是一个非常庞大的软件系统[4],大量的为保护各种共享资源而存在的临界区(critical section)被中断锁和剥夺锁所保护,这些临界区是导致 PDLT 不确定的根本原因。

(1)中断锁。中断锁即通过关中断来实现对临界区的保护,这种保护方式确保了临界区不会被内核的其他任何线程所打断,显然这种保护方式最为严格。为了缩短关中时间,保证系统的硬实时性能,这种临界区应该尽量缩短。幸运的是要做到这一点并不太难,因为内核中这类临界区数量极为有限,而且一般都很短。

(2)剥夺锁。Linux 2.6 内核正式引入了内核态的可剥夺调度,对于单处理器的 Linux 2.6 内核,剥夺锁实现为一个剥夺计数,每个任务都有一个自己的剥夺计数,仅当其为零时,才允许当前任务被剥夺调度。为了降低调度延迟,一个显然的改造方案是将剥夺锁用互斥信号量(mutex)来替代。然而,即便不考虑系统开销的增加,由于剥夺锁在内核中的无处不在以及剥夺锁可嵌套使用,要一蹴而就地把所有剥夺锁用互斥信号量来替代,几乎也是不可能的,这正是基于单 Linux 内核来实现嵌入式硬实时 Linux 的最大困难。

幸运的是,嵌入式系统的一个重要特点在于其资源的有限性和资源访问的可预知性。这意味着,对于嵌入式系统中的 HRTT 能够预知它将访问的有限的共享资源的种类和个数,只需要保证这些有限的共享资源在内核中的互斥访问。也就是说,只需要将这些有限的共享资源所涉及的剥夺锁用互斥信号量来替代,这显然是可行的。

然而,问题仍未解决,未被替代的剥夺锁仍将导致不确定的 PDLT。这使得我们进入了一个两难的境地:一方面,为了获得确定的 PDLT,必须把所有剥夺锁用互斥信号量来替代;另一方面,替代所有剥夺锁并不可行。为了解决这一困境,文中提出了"基于有限的共享资源模型进行强制剥夺调度"的新思路。

2.2 定义及假设

2.2.1 共享资源

在嵌入式系统中,被各种并发执行的任务共享的资源称为共享资源[5]。这些共享资源可以分类为可剥夺的(preemptible)和不可剥夺的(non - preemptible)。不可剥夺的共享资源必须被各种并发执行的任务互斥地访问,否则,将发生无法预测的结果。文中所指的共享资源特指不可剥夺的共享资源。

2.2.2 PDLT

文中提到的 PDLT(Process Dispatch Latency Time)定义如下[6]:从 HRTT 中断产生到 HRTT 被调度并开始执行第一条指令这段延迟时间,见图 1。

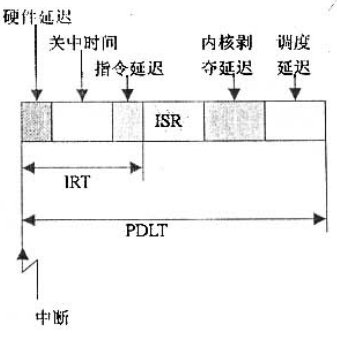

图 1 包含多个延迟的 PDLT

可见,PDLT 主要包括四个延迟:中断响应延迟(IRT);中断处理延迟(ISR Latency);内核剥夺延迟(Preemption Delay);调度延迟(Scheduling Latency)。PDLT 完整地反映了系统的硬实时性能,可以作为评估系统硬实时性能的指标。

另外,PDLT 的定义基于如下假设:

(1)HRTT 由中断驱动,也就是说,HRTT 中断产生将使得 HRTT 被唤醒而处于就绪态;

(2)HRTT 具有实时优先级,将被调度程序优先调度。

2.2.3　延迟分析

基于以上定义和假设,可以进一步分析构成 PDLT 的四个延迟。中断响应延迟,如前所述,导致这个延迟的主要原因是中断锁引起的关中时间,可以尽量将其缩短,但不可避免(如内核中的原子操作一般即通过关中来实现)。幸好,内核中最大关中时间一般也就十几微秒,并不大,如果希望将其降到更低,还可以对多数中断锁采取不关闭 HRTT 中断的措施;中断处理延迟,这实际上是 HRTT 的 ISR 执行的时间,完全是可控的和确定的,可以控制在指标要求之内;调度延迟,这是调度程序从运行队列中选择下一个就绪任务并切换到它开始执行的延迟时间,对于 Linux 2.6 来说,由于采用了 O(1)调度器,该延迟也是确定的。

至于内核剥夺延迟,这正是由于前面提到的剥夺锁所导致的,是导致不确定的 PDLT 的主要因素,如何采取有效的措施保证该延迟降到一个确定的小值(记为△)之内,是基于单 Linux 内核实现嵌入式硬实时 Linux 的关键所在。

2.3　强制剥夺调度

如前所述,Linux 2.6 内核正式引入了内核态的可剥夺调度,通过剥夺锁来保护内核中的共享资源,当一个任务没有持有剥夺锁时,即可在系统调用返回之前或在中断返回之前被更高优先级的任务剥夺。为了保证将内核剥夺延迟降到△之内,允许当前任务在持有剥夺锁时被 HRTT 剥夺,这就是文中提出的强制剥夺调度的概念。

显然,为了保证对共享资源的互斥访问,必须采取如下两个措施:

(1)对于 HRTT 将访问的有限的共享资源,必须将内核中相应的剥夺锁替换为互斥信号量,以保证当前任务和 HRTT 对这些共享资源的互斥访问;

(2)HRTT 执行完成让出 CPU 时,必须确保原来被强制剥夺的任务得到调度执行,以保证除 HRTT 以外的其他任务对共享资源的互斥访问。

这样就能保证将内核剥夺延迟降到△之内吗?这取决于当前任务和 HRTT 是否竞争共享资源。如果为否,则内核剥夺延迟将为零,能保证;否则,由于优先级反转问题的存在,该延迟仍是不确定的,见图 2。在该图中,高优先级任务由于与低优先级任务竞争共享资源而被后者阻塞($t_1 \sim t_3$),之后低优先级任务被中优先级任务剥夺($t_4 \sim t_5$),而后者的执行时间将是不确定的,由此高优先级任务何时得到调度执行将是不确定的,也就是说,优先级反转问题将导致不确定的内核剥夺延迟。幸运的是,已有现成的采用优先级继承协议的补丁来克服这一问题[6]。

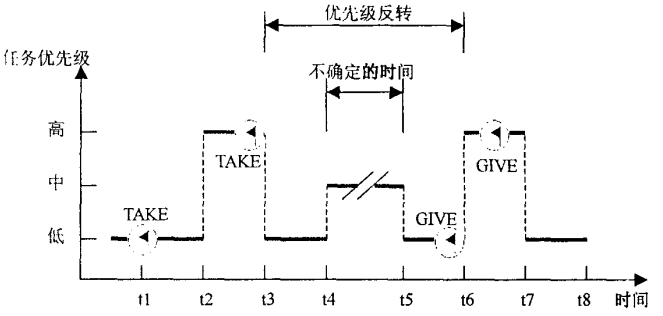

图 2　优先级反转导致不确定的内核剥夺延迟

2.4　有限的共享资源模型

如前所述,嵌入式系统的一个重要特点在于其资源的有限性和资源访问的可预知性,这使得我们可以从 HRTT 对共享资源需求的角度建立如下三个模型,分别对应三类不同的硬实时系统。

(1)模型一:HRTT 不访问共享资源。

这类系统中,HRTT 在执行过程中不会访问任何内核共享资源。在实现效果上,它与双核方案是一样的,即 HRTT 在执行过程中不调用任何 Linux 系统调用;唯一的区别在于本方案中 HRTT 的生成和退出仍调用 Linux 系统调用。该模型并非不切实际,因为一些嵌入式硬实时系统中,HRTT 通过独占的存储器资源(如 DSP 的片内存储器)和 I/O 资源进行处理和输入输出。对于这种模型,实现强制剥夺只需采用措施(2),内核剥夺延迟将为零。

(2)模型二:HRTT 访问有限的共享资源。

这类系统中,HRTT 在执行过程中只会访问可预知的的有限种类和个数的共享资源。这符合一般嵌入式硬实时系统的实际情况。对于这种模型,实现强制剥夺需要采用措施(1)和(2),而且需要用带优先级继承的互斥信号量来替代剥夺锁,以保证将内核剥夺延迟降到△之内。

(3)模型三:HRTT 访问共享资源。

这类系统中,HRTT 在执行过程中可能会访问任何种类和个数的共享资源。这意味着 HRTT 与一般任务一样可能调用任何 Linux 系统调用,这可以认为是我们最终希望达到的目标。如前所述,为了达到这个目标,必须将内核中所有剥夺锁替换为互斥信号量。

由此看来,我们实际上是提供了一个能力模型,以这种"渐进"的思路来克服"一蹴而就地把所有剥夺锁用互斥信号量来替代"的巨大困难,可以在较短的周期内建立满足能力要求的嵌入式硬实时 Linux 系统。同时,这种"渐进"的思路将最终可以达到模型三的要求,

这正是希望达到的终极目标。

3 在 ADSP BF53x 目标平台上的实现

3.1 目标平台介绍

该目标平台采用 ADI 公司的新一代混合处理器(DSP 核 + 通用 MPU 外设)ADSP BF53x[7],该处理器支持内核态/用户态,支持 4GB 物理地址空间,不支持页式内存管理和保护。因此,采用的 Linux 为 μClinux,内核版本为 2.6.12。对物理地址空间的直接访问对于嵌入式硬实时系统来说,不失为一个好的折衷,即牺牲一部分安全性来保证硬实时性能。

接下来简要探讨一下在该目标平台上实现嵌入式硬实时 Linux 所涉及的一些关键技术。

3.2 任意中断嵌套层次的可剥夺调度

这是实现内核态可剥夺调度的基础。由于 ADSP BF53x 采用了一种严格基于优先级的中断控制机制,即当前中断返回之前同优先级和低优先级的中断将被自动屏蔽,这使得实现任意中断嵌套层次的可剥夺调度需要一些技巧和足够清醒的头脑。主要的技巧是:剥夺调度之前采用"假的中断返回"以允许其后对同优先级和低优先级中断的响应。限于篇幅,此处不再叙述。

显然,这一点是与体系结构紧密相关的。对于非基于优先级的中断控制机制的情况(如 x86 和大多数通用 MPU),这一点并不需要什么额外工作。

3.3 用带优先级继承的互斥信号量替代剥夺锁

如前所述,关于这一点,已有现成的补丁,主要的工作在于对其的测试和进一步改进,尤其是对于所实现的优先级继承协议的理解和改进(参考文献[6])。

这一点可以实现为与体系结构无关。

3.4 强制剥夺调度

首先,定义 HRTT 将访问的内核共享资源;其次,将内核中这些共享资源所涉及的剥夺锁替换为带优先级继承的互斥信号量;最后,修改内核态剥夺调度的相关代码,允许 HRTT 中断返回之前强制剥夺调度到 HRTT,在 HRTT 执行完成让出 CPU 时,跳过常规的调度策略,直接调度之前被强制剥夺的任务。

这一点在实现中既包含与体系结构紧密相关的部分也包含与体系结构无关的部分。

至此,就基于有限的共享资源模型高效地建立了一个嵌入式硬实时 Linux。

4 测试和对比

在以下三种情况下分别对 PDLT 进行了测试:

(1)未实现"任意中断嵌套层次的可剥夺调度";

(2)实现"任意中断嵌套层次的可剥夺调度";

(3)实现"强制剥夺调度"。

测试结果如表 1 所示。这说明该嵌入式硬实时 Linux 能够达到 Max(PDLT)<100μs 的硬实时性能指标,满足设计要求。

表 1 Max(PDLT)对比测试结果

	Max(PDLT)
情况(1)	<10ms
情况(2)	<1ms
情况(3)	<100μs

注:硬件环境为 ADSP BF53x CCLK/SCLK = 398MHz/79MHz

5 结 语

文中提出了一种基于有限的共享资源模型在单 Linux 内核上实现嵌入式硬实时 Linux 的新思路,并在目标平台上予以实现,对比测试结果表明可以达到 Max(PDLT)<100μs 的硬实时性能指标,同时满足文中提出的嵌入式硬实时 Linux 的两点设计要求。这说明基于单 Linux 内核实现嵌入式硬实时 Linux 是可行的。

进一步的工作是围绕这一模型对该单核方案进一步改进和完善,向终极目标(HRTT 与一般任务一样可调用任何 Linux 系统调用)靠近。

参考文献:

[1] Marchesin A. Using Linux for Real - Time Applications[EB/OL]. 2004. http://www..computer.org/software.

[2] 肖皓月,罗克露. Linux 实时化技术研究[C]//中国西部嵌入式系统与单片机技术论坛 2005 学术年会论文集. 北京:北京航空航天大学出版社,2005:66 - 70.

[3] Raja S. Linux for Real Time Requirements[EB/OL]. 2004 - 08 - 27. http://www.isofttech.com.

[4] Love R. Linux 内核设计与实现[M]. 第 2 版. 北京:机械工业出版社,2006.

[5] 李 庆. 嵌入式系统的实时概念[M]. 北京:北京航空航天大学出版社,2004.

[6] Heursch A C, Grambow D, Roedel D, et al. Time - critical tasks in Linux 2.6 - Concepts to increase the preemptability of the Linux kernel[EB/OL]. 2004. http://www.informatik.unibw - muenchen.de/inst3/index.

[7] ADI. ADSP - BF53x Blackfin(r) Processor Hardware Reference[EB/OL]. 2004 - 09. http://www.analog.com. Revision 3.0.