2006 年 2 月
第 33 卷 第 1 期
西安电子科技大学学报(自然科学版)
JOURNAL OF XIDIAN UNIVERSITY
Feb. 2006
Vol. 33 No. 1

基于 Linux 的仪器操作系统研究与实现

何福贵[1,2], 王家礼[1]
(1. 西安电子科技大学 机电工程学院,陕西 西安 710071;
2. 太原理工大学 信息工程学院,山西 太原 030024)

摘要: 分析了 Linux 的特点和作为仪器操作系统的不足. 为了设计能够满足仪器要求的操作系统,在时钟粒度、内核可剥夺性和优先级反转等方面,对 Linux 的内核进行了改造. 通过分析 CPU 的时钟结构,实现了时钟粒度的细化,通过双内核在原内核中增加抢占点相结合,实现了具有可剥夺性的新内核,通过优先级继承协议的实现,避免了优先级翻转. 最后进行了实验测试. 实验结果表明,该结构增强了实时性,拓宽了内核的适用范围,可满足仪器的需要.

关键词: 仪器操作系统;调度延迟;时钟粒度;优先级反转
中图分类号: TP316 **文献标识码:** A **文章编号:** 1001-2400(2006)01-0129-04

Analysis and design of the instrument OS based on Linux

HE Fu-gui[1,2], *WANG Jia-li*[1]
(1. School of Mechano-electronic Engineering, Xidian Univ., Xi'an 710071, China; 2. College of Information Engineering, Taiyuan Univ. of Technology, Taiyuan 030024, China)

Abstract: The demand for the instrumental OS is becoming more and more urgent because of instrumental development. The paper analyzes Linux's characteristics and shortages if it acts as an instrumental OS. In order to design this kind of OS that satisfies instrumental requirement, Linux's kernel is modified in timer resolution, kernel preemption and priority reversion and so on. Improving the timer is realized by the analysis of the CPU structure; and a new preemptive kernel is realized by the integration of the dual-kernel with inserting preemptive points, and priority reversion is avoided by the realization of PIP. Finally, the test is done. The result shows that real-time performance is efficiently improved and the scope of use is extended, and thus it is effective for instrumental requirement.

Key Words: instrumental OS; schedule-latency; timer resolution; priority reversion

现代仪器软件部分设计采用操作系统和应用软件相分离的原则. 目前采用的操作系统主要有 Windows 系列和 Vxwork, Vxwork 主要用于功能较简单的仪器, Windows 用于功能较复杂的仪器,但 Vxwork 是商业软件,其使用费用昂贵,极大地增加了产品的成本;而 Windows 占用的资源较大,由于其源代码不开放,可定制能力差. Linux 作为操作系统的后起之秀,以其开放源代码、免费、支持多种硬件平台、模块化设计、对网络支持好等优点,逐渐受到开发者的重视. 仪器应用软件属于嵌入式应用领域,一般具有实时性的要求,所以要求操作系统必须能够支持实时性. Linux 是一个桌面操作系统,符合 POSIX1003.1b 关于实时扩展部分的标准,例如支持 SCHED_FIFO 和 SCHED_RR 实时调度策略、锁内存机制、实时信号等功能,但由于最初的设计目标为通用分时操作系统,追求的是运行的公平性,对实时的支持有限,需要对其进行实时改造,才能满足要求. Linux 作为一个实时系统,主要存在以下缺陷[1]:(1)Linux 是整体结构,它的调度算法是非抢占式. 这就不能

收稿日期: 2005-02-25
基金项目: 电子测试技术重点实验室基金资助项目(51487010503DZ104)
作者简介: 何福贵(1966-),男,西安电子科技大学博士研究生.

确保任意时刻系统运行的进程是具有最高优先级的进程. (2)Linux 的时钟粒度被设置为10 ms,而实时应用一般要求微秒级的响应精度,10 ms的时钟粒度不能满足要求. (3)Linux 为了保证核心数据的完整性,在进入对关键数据结构进行修改时,通常采用'关中断'的方式. 这时系统无法对中断做出响应,而非周期实时进程大多是由中断做出响应的,对于周期性实时进程而言,也需要调度模块来调度运行,而调度模块的执行是由时钟中断触发的. 所以频繁的关中断会导致实时任务不能被及时调度执行. (4)Linux 未能解决优先级翻转问题.

1 Linux 的实时性扩展分析

近几年来,人们对实时调度进行了广泛的研究[2,3],但实时调度理论是以可剥夺性为基础. 为了增强 Linux 的实时性,对 Linux 内核的实时性进行了扩展,提出了一些方案,从不同的方面加强 Linux 的实时性,主要有以下一些方面.

1.1 可剥夺补丁

由 Monta Vista 开发,为了提高 Linux 调度的响应时间,他们开发了一个剥夺补丁,对调度器进行实时化修改. 剥夺补丁的基本思想是增加调度函数运行的机会,通过修改 spinlock(旋转锁)和中断返回代码来实现,另外在任务结构中增加一个变量 preempt_count,这个变量通过以下3个宏修改:preempt_disable(),preempt_enable()和 preempt_enable_no_resched(). 宏 preempt_disable()增加变量 preempt_count 的值,宏 preempt_enable()减少变量 preempt_count 的值. 宏 preempt_enable()检查调度条件,如果 need_resched 的值是1且 preempt_count 的值是0,调用函数 preempt_enable()发起一次调度. 宏 spin_lock()被修改,首先调用 preempt_disable(),然后操作锁变量;宏 spin_unlock()被修改,首先操作锁变量,然后调用 preempt_enable();宏 spin_trylock()被修改,首先调用 preempt_disable(),然后调用 preempt_enable(). 在剥夺补丁中也修改了中断返回代码,增加 preempt_enable()做调度测试. 可剥夺补丁增加了 schedule()函数执行的机会,每一次 spinlock 释放和中断返回时,有一次调度的机会.

1.2 低延迟补丁

由 Ingo Molnar 开发,现在由 Andrew Morton 维护. 这个补丁集中在 Linux 阻塞代码中引进剥夺点,这些阻塞代码可能有很长的执行时间. 例如在一个时间较长的循环迭代中,设置一个门限,当超过这个门限时引进一次调度. 有时需要去掉一个 spinlock,进行调度,然后重新加 spinlock,这个过程也称作破锁. 低延迟是一个简单的概念,但实现并不简单,发现延迟长的阻塞代码是一个工作量很大的调试任务. 例如通过调试,发现 prune_dcache()函数(这个函数的功能是释放目录缓存)是引起阻塞时间长的一个函数,对这个函数进行的修改详见:http://www.linuxdevices.com/articles027/rh-rtpaper.pdf.

1.3 RTlinux 补丁

RT-Linux 由美国新墨西哥洲 Victor Yodaiken 提出设想,与 Michael Barabanov 共同开发的一个基于 Linux 的硬实时补丁,主要用于仪器设备和嵌入式系统,它主要通过:(1)接管全部中断,原来 Linux 中的中断使用软中断代替;(2)基于通用分时系统核心构造硬实时操作系统,通过在 Linux 内核附加一个简单的实时内核,而 Linux 本身作为这个实时内核的优先级最底的任务,所有的实时任务的优先级都要高于 Linux 本身以及 Linux 中的一般任务. 操作系统结构见图1.

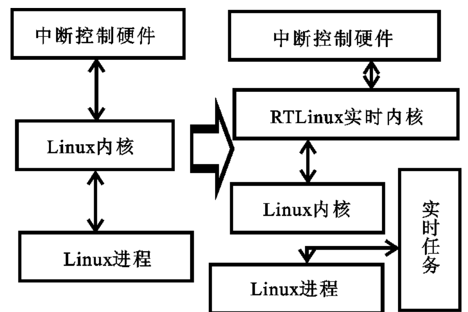

图1 Linux 到 RTlinux 体系结构的变化

2 基于 Linux 的仪器操作系统的设计

与一般的桌面操作系统相比,仪器操作系统有很大的不同. 仪器测试必须在一定时间内完成,要求软件系统具有很强的实时性. 仪器产品在完成以后,其应用的硬件范围和软件完成的功能相对固定,变化范围较小. 为了使 Linux 能够满足仪器测试的要求,进行了以下几方面的工作.

2.1　时钟粒度的细化

Linux 的时钟中断周期为 10 ms,而一般的实时系统要求达到微秒级,Linux 时钟粒度不能满足要求. 如果缩短 Linux 的时钟周期,那么时钟中断处理程序带来的开销增加. 经验表明当 CPU 的时钟中断周期达到 40 μs时,中断处理开销几乎占据了所有的 CPU 运行时间. 所以缩短时钟周期来提高时钟粒度显然是不可行的. 一种已经使用的方法是将 Linux 的时钟中断由周期性中断改为非周期中断,这样既能够将时钟周期缩短,又能够减少系统开销. 但是 Linux 的调度程序是以时钟中断作为调度的时间依据,这样做势必影响到有关的程序.

这里通过提供一个与 Linux 核心时钟并行的一个具有细粒度的实时核心时钟来达到要求. 核心时钟运行在周期模式,实时核心时钟运行在非周期模式,即一次模式.

一般的计算机系统提供 3 种时钟硬件:(1)实时时钟(RTC),也称作 CMOS Timer;(2)可编成定时器(PIT);(3)时间戳计数器(TSC). Linux 核心时钟使用 PIT,这里使用 RTC 作为实时时钟.

实现过程简单地描述如下:首先通过 Linux 的可安装模块机制,将需要执行的中断服务程序与 RTC 中断(PC 机的 RTC 中断为 8 号)联系起来. 在应用程序使用实时时钟时,通过系统调用首先设置 CMOS 状态寄存器设置周期,然后开中断;不需要实时时钟时,通过系统调用设置 CMOS 状态寄存器关中断.

2.2　可剥夺内核实现和调度策略

可剥夺补丁和低延迟补丁通过修改 Linux 内核实现了内核的可剥夺性,支持软实时调度;RTLinux 通过在 Linux 内核下增加一个新的实时内核,实现了支持硬实时系统.

Clark Williams 对可剥夺补丁和低延迟补丁进行详细的测试,首先对两个补丁分别进行了测试,然后将两个补丁同时加入 Linux,合在一起进行了测试. 测试结果表明,两个补丁都能够减少内核延迟,如果单独使用,低延迟补丁效果好于可剥夺补丁;两个补丁结合在一起,相互之间并不矛盾,效果略好于低延迟补丁. 基于 Clark Williams 的方法,采用如下的方法:采用低延迟补丁和可剥夺补丁加入 Linux 内核,将 RTLinux 补丁加入内核. 这样使 3 个补丁和 Linux 内核合在一起,使新的内核不但支持软实时系统,而且支持硬实时系统,使内核的适用范围更宽. 最终的测试结果表明,改进后内核的精度有了明显的提高.

调度分为静态调度和动态调度两种,静态调度的灵活性差,但它的上下文切换的时间短;动态调度则反之. 具体采用哪一种要根据具体的应用环境来确定,由于仪器上运行的应用软件的功能相对固定,所以宜采用静态调度.

2.3　优先级反转的解决

在内核能够剥夺的情况下,由于多进程共享资源而互斥,具有最高优先级的进程被低优先级进程阻塞,反而使具有中优先级的进程先于高优先级的进程执行,导致系统的崩溃,这就是所谓的优先级反转问题[4]. 优先级反转问题能够导致死锁发生,文献[5]提出了一种判断方法,但根本问题应避免优先级反转. 目前,优先级反转普遍使用优先级继承和优先级极限,在优先级继承方案中,当高优先级任务在等低优先级的任务占有的信号量时,使低优先级任务继承高优先级任务的优先级,当低优先级任务释放高优先级任务等待的信号量时,立即把其优先级降到原来的优先级. 在优先级极限方案中,系统把每一个临界资源与一个极限优先权联系,这个极限优先权等于系统此时最高优先权加 1,当一个任务进入临界区后,系统把这个极限优先权传递给这个任务,使得这个任务的优先权最高;当一个任务退出临界区后,系统立即把它的优先权恢复正常,从而保证系统不会出现优先权反转的情况.

Linux 对资源的互斥保护是通过信号量来实现的,所以 PIP 是通过操作信号量来实现的. 文献[4]对优先级反转的各种情况进行了分析,但没有描述实现方法. 文献[6]对 Linux 源代码做了详细的分析,下面描述其实现过程:

(1) 修改 Linux 的 task_struct 数据结构,将原来的优先级作为初始优先级,增加一项:pip_pointer,指向数据结构 pip_priority 指针. 该数据结构信号指针和继承优先级(被该信号阻塞的进程中的最高优先级).

(2) 修改 Linux 的信号量 semaphore 数据结构,增加一项 holding_task 进程指针. 指向持有该信号的进程.

(3) 信号量操作函数_down()的修改　在原来代码的基础上增加如下代码:如果能够进入临界区,则将该进程挂入 holding_task 指向的进程队列,否则修改 semaphore 的 holding_task 指向的进程的 pip_pointer 指针的继承优先级队列,将本信号和本进程的优先级作为数据结构 pip_priority 的成员项加入 pip_pointer 指向的队列. 在加入的过程中,如果本信号已存在,则比较两个继承优先级,取优先级高者.

(4) 信号量操作函数_up()的修改　在原来代码的基础上增加如下代码:将数据结构 task_struct 的 pip_pointer 指

向的该信号的 pip_priority 结构删除,将数据结构 semaphore 的 holding_task 指向的进程中,删除本进程.

(5) 调度函数 schedule()的修改 进程运行优先级确定修改为:如果 pip_pointer 指针为空,运行优先级为初始优先级,否则将 pip_pointer 指向的最高继承优先级为运行优先级.

优先级继承协议的实现可采用类似的方法.

3 实验测试

在测试调度延迟的过程中,使用 Linux 工具 stress 来逐渐增加系统的载荷,通过载荷的变化来观测调度延迟时间的变化. 测试系统的调度延迟的工具选用 Linux 测试工具 amlat. 这个工具主要用来测试调度延迟. 选用 Linux 的内核版本为 Linux-2.4.20,其他3个补丁的版本也为 2.4.20.

该实验运行环境:CPU 2.6 GHz,内存 256 M,磁盘 80 G. 加载命令如下:

Stress -cpu 8 -io 4 -vm 2 -vm-bytes 128 M -tineout 100 s -用 1-Load 表示.

Stress -cpu 16 -io 8 -vm 4 -vm-bytes 128 M -tineout 100 s -用 2-Load 表示.

依次类推. 测试结果见表1.

表1 调度延迟测试结果

内核类别 / 载荷 延迟	Linux 内核/ms	Preemption + lowlatency 内核/ms	Preemption + lowlatency + RTLinux 内核/ms
1-Load	991.747	0.659	0.897
2-Load	1011.232	2.033	1.387
4-Load	1138.092	2.125	1.503
8-Load	1502.578	2.111	1.727
16-Load	抖动	抖动	抖动

表1中的数据均为延迟的最大值,最后一行表明载荷太大,系统出现抖动.

从表中可看出,标准 Linux 的延迟较大,3个补丁加在一起比两个补丁加在一起略有提高,但3个补丁的内核的应用范围拓宽,适用性更强,表现出很好的性能.

4 结 束 语

Linux 虽然是一个通用的分时操作系统,但由于其功能强大、源代码开放及免费等优势使得改进 Linux 适合实时系统成为一种选择. 在分析仪器特点的基础上,提出适合仪器操作系统的改造方案,主要在时钟粒度的细化、内核的可剥夺性和优先级反转等方面进行了改进,实验结果表明,该方案有效地提高了系统的性能,可满足实际的需要.

参考文献:

[1] 李小群,赵慧斌,叶以明,等. RFRTOS:基于 Linux 的实时操作系统[J]. 软件学报,2003,14(7):1203-1212.

[2] Zhang Huijuan, Zhou Shuisheng, Zhou Lihua. A Fair-scheduling Algorithm for Apordic and Period Tasks on Multiprocessor Systems[J]. Journal of Xidian University, 2004, 31(2):272-275.

[3] Wang Yuchung. Design and Implementation of RED-Linux[D]. California:University of California, 2002.

[4] Sha L, Rajkumar R, Lehoczky J P. Priority Inheritance Protocols:an Approach to Real-time Synchronization[J]. IEEE Trans on Computers, 1990, 39(9):1175-1185.

[5] Li Zhiwu, Liu Hong. A Method for Deciding Deadlock Structures Caused by Parallel-shared Resources[J]. Journal of Xidian University, 1999, 26(1):18-21.

[6] 毛德操,胡希明. Linux 内核源代码情景分析[M]. 杭州:浙江大学出版社,2001.

(编辑:齐淑娟)