基于嵌入式 Linux 2.6 的实时优化

肖振华,徐玉斌,解 辉,吕亚男
(太原科技大学 计算机学院,山西 太原 030024)

摘 要:在分析了国内外嵌入式 Linux 实时技术的基础上,根据 Linux 2.6 内核和嵌入式实时操作系统的特点,采用直接修改 Linux 内核的方式,从中断线程化、自旋锁可抢占、优化 O(1)调度算法三个方面提出了一种针对 Linux 2.6 的实时优化方案。该方案的提出使得 Linux2.6 的实时性能在内核可抢占的基础上得到了进一步的提高,扩充了 Linux 在嵌入式领域的实时应用。

关键词:Linux 2.6;实时性;抢占;调度算法
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2008)11-0083-04

An Optimization of Real-Time Capability Based on Embedded Linux 2.6

XIAO Zhen-hua, XU Yu-bin, XIE Hui, LÜ Ya-nan
(Sch. of Computer, Taiyuan Univ. of Science and Technology, Taiyuan 030024, China)

Abstract: According to the characteristics of Linux 2.6 and embedded real-time operating system, based on the analysis of real-time technology of embedded Linux at home and abroad, gives one method to improve the real-time capability of embedded Linux 2.6 by amending Linux kernel directly. Based on the kernel preemption of Linux 2.6, this method improves the real-time capability of Linux 2.6 further, extending the real-time application of Linux in embedded field.

Key words: Linux 2.6; real-time; preemption; schedule algorithm

0 引 言

Linux 的开源、免费、支持多种架构使得它在嵌入式系统中得到了广泛的应用[1],而 Linux 在设计之初并没有对实时性进行任何考虑,内核中也不允许抢占[2]。随着嵌入式系统对实时性要求越来越高,使得 Linux 在嵌入式领域中的应用受到了一定的限制。目前,最新的 Linux 2.6 标准内核已经可以抢占,其实时性得到了加强[3],但是内核中仍有大量的不可抢占区域影响着系统的实时性能。因此,为了满足当前嵌入式系统对实时性的要求,针对 Linux 2.6 内核的实时优化日益成为研究的焦点。

1 现存的 Linux 实时技术

由于 Linux 的开放性和低成本,越来越多的研究机构和商业团体开展了对实时 Linux 的研究与开发,现有著名的实时 Linux 实现包括 RT-Linux(Real Time Linux)[4,5]、RTAI(Real Time Application Interface)[6]、TimeSys Linux and Ingo's RT patch 等。提高 Linux 实时性的方法可以归结为两种[7]:

(1)双内核方式。RT-Linux 和 RTAI 即采用这种方法。其主要实现方法是用一个实时内核负责处理硬件消息,实时任务在该内核上直接运行,把 Linux 内核本身作为优先级最低的 Idle task 运行,只有当没有实时任务需要运行时,Linux 内核才有机会运行。这种方法虽然实现了硬实时,但舍弃了 Linux 固有的稳定性、可靠性、支持多种架构的优点,并且针对实时内核需要重新编写驱动程序,更重要的是 Linux 开发社区的成果无法直接应用到实时核心上。

(2)直接修改 Linux 内核的方式。TimeSys Linux 和 Ingo's RT patch 采用了这种方法。TimeSys Linux 采用了中断线程化、修改了 spinlock 自旋锁、优化了调度器。Ingo's RT patch 也是借鉴这些思路来实现实时性的,且有一些实现是基于 TimeSys 的源代码的,如中断线程化。这种实现方式保持了全部的 Linux 应用编

收稿日期:2008-02-02
基金项目:山西省科技攻关项目(051127)
作者简介:肖振华(1981-),男,硕士研究生,研究方向为嵌入式 Linux 实时系统;徐玉斌,教授,研究方向为工业控制网络、嵌入式系统。

程模式,实时应用和普通的应用采用同样的编程方式,使用同样的 API,只是实时任务需要明确指定自己的优先级与调度策略。

2 制约 Linux 2.6 内核实时性的主要因素

除了研究机构和商业团体为提高 Linux 的实时性所做的贡献,Linux 开发者本身也对改进 Linux 的实时性做了大量工作。Linux 2.4 及以前版本的内核是不可抢占的,但在 Linux 2.6 中,内核已经可以抢占,高优先级内核空间进程可以像在用户空间里那样抢占低优先级进程的资源,因而实时性得到了加强,但是内核中仍有大量的不可抢占区域,制约内核实时性的因素大量存在。

2.1 中断处理

在 Linux 2.6 及以前版本中,中断(包括软中断)是最高优先级的,不论在任何时刻,只要产生中断事件,内核将立即执行相应的中断处理函数以及软中断,等到所有挂起的中断和软中断处理完毕才执行正常的任务。因此在标准的 Linux 系统中,中断不可抢占,中断延时无法预计,实时任务根本不可能得到实时性保证。例如,假设在一个标准 Linux 系统上运行了一个实时任务(即使用了 SCHED_FIFO 调度策略并且设定了最高的实时优先级),而此时该系统有非常繁重的网络负载和 I/O 负载,那么系统可能一直处在中断处理状态而没有机会运行任何任务,这样实时任务将永远无法运行,抢占延迟将是无穷大。

2.2 自旋锁(spinlock)

自旋锁是在可抢占内核和对称多处理器 SMP (Symmetric Multiple Processors)情况下对共享资源的一种同步机制。一般情况下,一个任务对共享资源的访问是非常短暂的,如果两个任务竞争一个共享的资源时,没有得到资源的任务将自旋以等待另一个任务使用完该共享资源。这种锁机制是非常高效的,但是任务在保持自旋锁期间将失效抢占,这意味着抢占延迟将增加。在 Linux 2.6 内核中,自旋锁的使用非常普遍,有的甚至对整个一个数组或链表的遍历过程都使用自旋锁。因此抢占延迟非常不确定。

另外,由于历史原因,Linux 内核一直保留有几个大内核锁,其实质上也是一种自旋锁,它与一般的自旋锁的区别在于它是用于同步整个内核的,而且一般该锁的保持时间较长,也即抢占失效时间长,因此它的使用将严重地影响抢占延迟。

2.3 调度算法

在 Linux 2.6 内核中引入的 $O(1)$ 调度算法比 Linux2.4 及以前版本的 $O(n)$ 调度算法具有更好的性能,它是一个基于优先级的抢占式调度器,为每一个进程分配一个唯一的优先级,高优先级的任务能够抢占低优先级的任务。然而,基于优先级的抢占式调度策略会造成优先级逆转现象。所谓优先级逆转,就是优先级高的进程由于与优先级低的进程保持了竞争资源而被迫等待,而让中间优先级的进程运行,中间优先级进程的执行时间的不确定性导致了高优先级进程抢占延迟的不确定性,进而高优先级进程抢占延迟增大。因此为了保证实时性,必须消除优先级逆转现象。

3 Linux 2.6 内核实时性的提高方法

3.1 中断线程化

中断线程化(包括软中断)是提高 Linux 实时性的一个重要步骤。中断线程化之后,中断将作为内核线程运行并被赋予不同的实时优先级,实时任务可以有比中断线程更高的优先级,这样,实时任务就可以作为最高优先级的执行单元来运行,即使在严重负载下仍有实时性保证。

目前较新的 Linux 2.6.17 还不支持中断线程化。但由 Ingo Molnar 设计并实现的实时补丁,实现了中断线程化。在 Ingo Molnar 的实时补丁中,中断线程化的实现方法如下:

在初始化阶段,中断线程化的中断初始化与常规中断初始化基本相同,在 start_kernel()函数中都调用了 trap_init()和 init_IRQ()两个函数来初始化 irq_desc_t 结构体。不同点主要体现在内核初始化创建 init 线程时,中断线程化的中断在 init()函数中还将调用 init_hardirqs,来为每一个中断创建一个内核线程,其实时优先级最低为 25,最高为 50。在 init_hardirqs()函数中创建中断内核线程的主要代码如下:

```
void_init init_hardirqs(void)
{
    ……
    for (i = 0; i <NR_IRQS; i++) {
      irq_desc_t *desc = irq_desc + i;
      if (desc->action && !(desc->status & IRQ_NODELAY))
        desc->thread = kthread_create(do_irqd, desc, "IRQ %d", irq);
    ……
    }
}
```

在中断处理阶段,中断线程化的中断和常规中断两者之间相同点是当发生中断时,CPU 将调用 do_IRQ()函数来处理架构相关部分,之后调用_do_IRQ()处理架构独立部分。而两者最大的不同点是在_

do_IRQ() 函数中,将判断该中断是否已经被线程化,如果中断描述符的状态字段不包含 IRQ_NODELAY 标志,则说明该中断已被线程化。

对于已被线程化的中断,调用 wake_up_process() 函数唤醒中断处理线程,并开始运行,内核线程将调用 do_hardirq() 函数来处理相应的中断,该函数将判断是否有中断需要被处理,如果有就调用 handle_IRQ_event() 函数来处理中断。对于没有被线程化的中断,将直接调用 handle_IRQ_event() 函数来处理中断。不管是线程化还是非线程化的中断,最终都会执行 handle_IRQ_event() 函数来调用相应的中断处理函数,只是线程化的中断处理函数是在内核线程中执行的。在 _do_IRQ() 函数中判断中断是否被线程化的主要代码如下:

```
fastcall notrace unsigned int _do_IRQ(unsigned int irq, struct pt_
regs * regs)
{
  ......
    if (redirect_hardirq(desc))
      goto out_no_end;
  ......
}
int redirect_hardirq(struct irq_desc * desc)
{
  ......
if (! hardirq_preemption || (desc -> status & IRQ_NODE-
LAY)|| ! desc -> thread)
    return 0;
  ......
}
```

3.2　用互拆锁替换自旋锁

在内核中大量地使用了自旋锁 spinlock,致使有大量的临界区存在,它们将严重地影响着系统的实时性。使用互斥锁 mutex (mutual - exclusion lock)来替换自旋锁 spinlock 的目的是为了让 spinlock 可抢占,以提高 Linux2.6 的实时性能。

原来的 spinlock 结构如下:

```
typedef struct {
  volatile unsigned long   lock;
# ifdef CONFIG_DEBUG_SPINLOCK
    unsigned int   magic;
# endif
# ifdef CONFIG_PREEMPT
      unsigned int break_lock;
# endif
} spinlock_t;
```

替换成 mutex 之后,它的结构为:

```
typedef struct {
    struct rt_mutex   lock;
    unsigned int   break_lock;
} spinlock_t;
```

其中 struct rt_mutex 结构如下:

```
struct rt_mutex {
    raw_spinlock_t   wait_lock;
    struct plist   wait_list;
    struct task_struct   * owner;
    int   owner_prio;
# ifdef CONFIG_RT_DEADLOCK_DETECT
    int   save_state;
    struct list_head   held_list;
    unsigned long   acquire_eip;
    char   * name, * file;
    int   line;
# endif
};
```

类型 raw_spinlock_t 就是原来的 spinlock_t。在结构 struct rt_mutex 中的 wait_list 字段就是优先级化的等待队列。等待队列优先级化的目的是为了更好地改善实时性,每次当 spinlock 保持者释放锁时总是唤醒排在最前面的优先级最高的进程或线程,而唤醒的时间复杂度为 $O(1)$。

3.3　优先级继承和优先级置顶

Linux 2.6 内核采用 $O(1)$调度算法后会产生优先级逆转现象。如图 1 所示,低优先级任务 L 和高优先级任务 H 都需要占用资源 a 来运行,低优先级任务 L 首先开始运行不久后,高优先级任务 H 也准备就绪,发现资源 a 正在被占用,于是任务 H 转入休眠状态,等待任务 L 释放资源 a。此时一个占用资源 b 的中优先级任务 M 出现,$O(1)$调度算法依据优先原则转而执行任务 M。这就进一步延长了高优先级任务 H 的等待时间,如果出现更多这样的中优先级 M,有可能使任务 H 错过临界期限而导致系统崩溃。

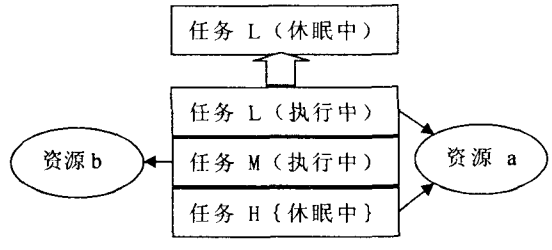

图 1　任务调度

优先级继承协议[8]可用来解决优先级逆转问题。当发生优先级逆转时,低优先级任务将继承和它竞争同一资源的高优先级任务的优先级,使得低优先级任务能尽快执行,释放出高优先级的任务所需要的资源。

这样高优先级任务就能很快完成,不给中优先级任务有可趁之机,使得高优先级任务抢占延迟更确定,更短。

优先级置顶协议也是专门针对优先级逆转问题提出的另一种解决办法,它为每个临界资源都分配一个优先级。假设任务 H 在所有要共享某资源的任务中优先级最高,则将此资源的优先级确定为任务 H 的优先级加 1。调度程序将该资源的优先级赋给任何访问该资源的任务,当低优先级的任务 L 访问该资源时就可获得 H+1 的优先级,这样就能保证低优先级任务 L 尽快完成对临界资源的访问,接着再运行任务 H。直到对该资源的所有操作完成后,其优先级恢复正常。

4 结束语

Linux 2.6 内核已经可以抢占,实时性得到了加强,但是内核中仍有大量的不可抢占区域,内核可抢占后必然会带来优先级逆转等问题。通过使用中断线程化、互拆锁可抢占、优化调度算法使得 Linux2.6 的实时性能得到了大大提高,扩充了 Linux 在嵌入式领域的实时应用。嵌入式技术具有广阔的应用前景,把 Linux 自身固有的优越性融入嵌入式领域是嵌入式技术发展的一个重要方向,因此通过直接修改 Linux 内核的方式提高 Linux 的实时性能仍然是今后研究的热点。

参考文献:

[1] 刘文峰,李程远,李善平. 嵌入式 Linux 操作系统的研究[J]. 浙江大学学报:工学版,2004,38(4):447-452.

[2] Michael B. A Linux-based Real-time Operating System [D]. Socorro, New Mexico: New Mexico Institute of Mining and Technology, 1997.

[3] 杨沙洲. Linux 2.6 调度系统分析[EB/OL]. 2004-04-12. http://blog.csdn.net/sah/articles/136098.aspx.

[4] Yodaiken V. The RT-Linux Manifesto[EB/OL]. 2003. http://www.rtlinux.org.

[5] 姜 醋. 基于 RT-Linux 的嵌入式实时系统设计[D]. 杭州:浙江大学,2003.

[6] 于 旭. 基于 Linux 的实时内核 RTAI 的实现机制研究[J]. 网络与信息,2004,18(5):94-94.

[7] 陈文星,张辉宜. 嵌入式 Linux 的实时性改进技术[J]. 计算机技术与发展,2006,16(10):114-117.

[8] Sha L, RajKumar R, Lehoczky J. Priority inheritance protocols: An approachto real time synchronization[J]. IEEE Transactions on Computers, 1990, 39(9): 1175-1185.

(上接第 82 页)

出现这种问题的原因当然很复杂,但纠其根源往往是企业不能或没有能力更新数据,在移动 Agent 电子商务中这种情况应该及时解决。TMA 服务器可以主动以移动 Agent 的形式对企业发布的信息定时地反馈给企业进行确认,保持信息的有效性,对过期的商品信息应及时清除并通知企业。

5 结束语

随着 Internet 的发展,基于商业需求的增加,移动 Agent 将会飞速发展,一种合理有效的移动 Agent 电子商务模型将有利于移动 Agent 的应用。文中在现有的一些移动 Agent 模型的基础上提出一种委托式移动 Agent(Trust-Mobile Agent, T-MA)模型,并提出了一系列电子商务中对企业及顾客的电子商务交易过程中切实可行的方案,尤其针对移动 Agent 电子商务的安全问题提出了较详细的解决方案,当然在实际应用时,可能产生未提出的一些问题,有待进一步研究。

参考文献:

[1] Das R, Hanson J E, Kephart J O, et al. Agent-human interactions in the continuous double auction[C]//Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence. Seattle, USA: IJCAI Press, 2001: 1169-1176.

[2] Barjis J, Chong Y C, Dietz J L G, et al. Development of agent based E-Commerce Systems Using Semiotic Approach and Demo Transaction Concept[J]. International Journal of Information Technology & Decision Making, 2002, 1(3): 491-510.

[3] He Minghua, Jennings N R, Leung Ho-Fung. On agent-mediated electronic commerce[J]. IEEE Transactions on Knowledge and Date Engineering, 2003, 15(4): 985-1003.

[4] 徐 锋,吕 建,陶先平. MABEMS:一个基于移动 Agent 的电子市场空间[J]. 南京大学学报,2002,38(2):131-138.

[5] 胡 健,刘锦德. 一个基于协作信息中间件的电子商务信息系统[J]. 计算机应用, 2001,21(4):4-6.

[6] 祁 明,卓光辉. 多智能代理网络购物系统的设计与分析[J]. 计算机工程与设计,2001,22(3):12-15.

[7] 王汝传,徐小龙,黄海平. 智能 agent 及其在信息网络中的应用[M]. 北京:邮电大学出版社,2006:39-40.

[8] Sander T, Tschudin C F. Towards Mobile Cryptography[C]// In Proceeding of the 1998 IEEE Symposium on Security on Security and Privaey. Oakland, CA: [s.n.], 1998.