RTLinux对Linux关中断的解决方案

刘 炜,李海华
(中国人民解放军信息工程大学 国家数字交换系统工程与技术中心,河南 郑州 450002)

摘 要:Linux操作系统启用禁止中断的原因有两个:1)操作系统为了保护重要的系统程序代码暂停接受中断;2)Linux内核的进程经常关闭中断以尽快完成自己的任务。在这两种情况下,都可能导致高优先级实时进程的中断发生系统也无法响应中断,从而使得系统实时性能降低。双内核解决方案能够有效地解决Linux内核的关中断问题。它在Linux内核之下插入一个实时子内核,使Linux工作在实时内核的控制下,让实时子内核处理实时任务而Linux内核处理普通任务。

关键词:中断;双内核;实时子内核
中图分类号:TP316.8 **文献标识码**:A **文章编号**:1673-629X(2009)08-0089-03

RTLinux's Solving Scheme Toward the Disable Interrupt of Linux

LIU Wei, LI Hai-hua
(National Digital Switching System Engineering & Technological Center, PLA Information PEngineering University, Zhengzhou 450002, China)

Abstract: There are two reasons for the disable interrupt of linux: 1) in order to protect the important code of the system program; 2) kernel of linux want to fulfill its tasks as soon as possible. On both conditions, possibility exists that it can't lead the OS(operating system) to respond to the intertupt and will reduce the real-time performance of the system. The dual-kernel solving scheme can solve effectively the disable interrupt of the linux. It insert in the linux kernel a realtime sub-kernel and make linux work under the control of realtime kernel. Thus the sub-kernel deals with realtimetasks while linux deal with the normal tasks.

Key words: disable interruption; dual-kernel; realtime sub-kernel

1 Linux临界区关中断

Linux内核可以看成是一个不断对请求进行响应的服务器,这些请求可能来自正在CPU上执行的进程,也可能来自正在执行中断请求的外部设备。因此,内核的各个部分并不是严格按照顺序依次执行的,而是采用交错执行的方式。内核控制路径是指内核用来处理系统调用、异常或中断所执行的指令序列。在交错执行内核控制路径时,引入了临界区的概念来避免可能带来数据混乱的危险,临界区是指一段代码,进入临界区的每一内核控制路径都全部执行完,在其他的内核控制路径才能够进入临界区[1]。关中断是用来确保内核语句序列可以作为一个临界区进行操作的一种主要机制,即使是硬件设备发出IRQ信号时,关中断也允许内核控制路径全部执行完。

收稿日期:2008-12-02;**修回日期**:2009-02-03
基金项目:国家863计划项目(2007AA01Z2a1)
作者简介:刘 炜(1961-),男,河南工业贸易职业学院高级讲师,主要研究方向为网络协议;李海华,副教授,主要研究方向为网络协议、网络安全。

Linux在进入一个临界区之前使用_save_flags_ptr宏来保存eflags的内容,使用cli()宏来清除eflags寄存器的IF标志来关中断,禁止所有的硬件中断;离开临界区时,使用_restore_flags宏来恢复eflags的内容,才允许产生中断。

通常这些宏以以下方法使用:

```
/*
_save_flags_ptr(old);
_cli();
.....
_restore_flags(old);
*/
```

如果临界区比较大,关中断保持相对较长时间就可能使所有硬件活动处于冻结状态。这段时间是不可预测的,必须考虑所有的临界区情况。这对实时操作系统来说是不允许的,因此,要想将Linux改造为实时操作系统就必须解决Linux的临界区关中断机制。

2 进程关中断

外部中断是环境向实时操作系统进行的输入,它

的频度是与环境变化的速率相关的,而与实时操作系统无关。如果外部中断产生的频度不可预测,则一个实时任务在运行时被中断处理程序阻塞的时间开销也是不可预测的,从而使任务的实时性得不到保证,因此,Linux 内核的进程经常关闭中断以尽快完成自己的任务。但同时也引入了问题:如果低优先级的进程关闭了中断,那么即使有高优先级实时进程的中断发生系统也无法响应[2]。这种情况在实时系统中也是不允许发生的。下面将深入分析 RTLinux 对 Linux 关中断的解决方案。

3 RT-Linux 工作原理

Linux 不直接与中断控制硬件进行联系,而是在二者之间加入了一个中断控制硬件的仿真层,或称实时子内核(RT-Linux),Linux 的进程和 RT-Linux 的进程之间通过 FIFO 来通信,使 Linux 工作在实时内核的控制下[3]。让实时子内核处理实时任务而 Linux 内核处理普通任务。当实时任务到来时,实时操作系统就运行实时任务。没有实时任务时,实时内核就调度 Linux 使之运行。

解决 Linux 禁止中断的问题是通过在实时内核中模拟 Linux 中断例程完成的。当 Linux 内核执行 cli() 禁止中断时,一个软中断标志被设置。当一个中断发生时,实时内核将捕获中断并根据这个标志和中断掩码来决定是否将该中断交给 Linux 内核处理。因此,虽然允许 Linux 禁止中断,但中断对实时内核总是可见的。如果该中断将引起一个实时任务的执行,则实时内核保存 Linux 的状态并立即开始执行实时任务。如果不是实时中断,中断就被挂起,之后查看 Linux 对该中断的操作状态,如果 Linux 没有禁止此中断,就将该中断交给 Linux,由 Linux 执行相应的中断处理函数。如果 Linux 禁止中断,则在 Linux 重新开启中断时,实时内核处理所有挂起的中断交由 Linux 执行相应的中断处理函数。

从图 1 中可以看到,实时子内核实际上工作在 Linux 内核和硬件中间,它最先知道硬件的信息,可以得到最快的响应速度;实时任务直接和实时内核进行交互,缩短了到硬件的时间,也提高了实时性。并且,实时任务和 Linux 普通进程之间也有通讯方式,对实时任务的监控可以在 Linux 进程这一端执行。这种实现方法可以充分利用 Linux 的强大功能和实时内核的实时性能。

这样,无论 Linux 处于用户模式或者内核模式,它是否禁止中断或是运行在临界区,实时子内核总是可以响应实时中断。这样便可以解决 Linux 禁止中断给实时任务增加延迟的问题[4]。

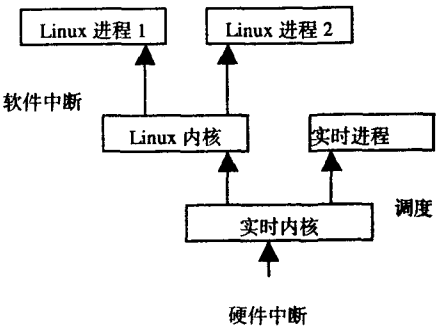

图 1 RTLinux 双内核结构

4 RTLinux 解决方案

RT-Linux 有两种中断:硬中断和软中断。软中断就是常规 Linux 内核中断[5]。下面将从软中断实现、注册中断例程、RT-Linux 中断处理程序和实时进程与普通 Linux 进程通信这几个方面阐述 RT-Linux 的具体实现。

4.1 软中断操作

RT-Linux 在初始化模块 Init_module() 中首先调用结构相关函数 arch_takeover() 来覆盖原来的关中断相关的函数,主要有:使用 rtl_soft_cli 替换_cli;使用 rtl_soft_sti 替换_sti;使用 rtl_soft_save_flags 替换_save_flags_ptr;使用 rtl_soft_restore_flags 替换_restore_flags。

(1)当 Linux 调用_cli()时,其实是调用 rtl_soft_cli(),它只是简单的清除中断允许标志 l_ienable。

(2) rtl_soft_sti()判断是否有挂起的中断,如果有就处理中断(rtl_process_pending()),最后设置 l_ienable 标志。

(3) rtl_soft_save_flags()将中断允许标志 l_ienable 保存在变量 x。

(4) rtl_soft_restore_flags()读取变量 x 的值,恢复标志 l_ienable。

4.2 注册中断服务例程

在 init_module()中调用 rtl_request_irq(IRQ_LINE, irq_handler)注册中断服务例程 irq_handler。在 cleanup_module()中调用 rtl_free_irq(IRQ_LINE)释放已注册的中断。

RTLinux 在执行实时中断服务例程时将禁止 IRQ。因此,中断服务例程 irq_handler 须在退出前调用 rtl_hard_enable_irq(IRQ_LINE)重新使能中断[6]。

4.3 RT-Linux 中断处理函数

使用 rtl_intercept()来代替 do_IRQ(),用来执行与一个中断相关的所有中断服务例程。这里主要做的

修改是将实时中断与普通 Linux 中断分开处理。

(1)它首先调用函数 rtl_irq_controller_get_irq(regs)判断是否获取到中断。

(2)如果获取到中断,并且是实时 irq 请求,就调用 dispatch_rtl_handler 执行实时 irq 对应的中断服务例程,然后,从中断返回。

(3)如果是非实时 irq 请求,就挂起 irq 请求 G_PEND(irq),设置挂起标志表明有挂起的 irq 请求 G_SET(g_pend_since_sti)。

若 Linux 开启了中断,就解除 irq 挂起状态 G_UNPEND(irq),调用函数 dispatch_Linux_irq 执行该 irq 对应中断服务程序,之后,从中断返回 。

4.4 RTLinux 实时进程与普通 Linux 进程的通信

有些实时应用程序无需任何用户界面即可在后台平静地运行,然而,越来越多的实时应用程序需要一个用户界面及其它系统功能。有效的嵌入式应用程序设计方法是将实时部分与非实时部分分离开来[7]。如果应用程序的任一部分,如用户界面、图形、数据库或网络仅需软实时性能,最好是将该部分写入用户空间。然后,仅将必须满足时序要求的那部分写成实时任务。

例如,如果利用实时内核上运行一个实时任务来对外界环境进行数据采集,那么采集出来的数据可以通过非实时内核上运行的图形界面显示出来。这样,既可以提高实时系统的可用性,也可以节省计算资源,同时将实时系统的一部分任务划分出来,降低了实时内核需要处理的复杂度,提高了实时的计算效率。

在 RTLinux 中使用实时 FIFO 进行实时进程与非实时进程之间的通信。当 insmod 将 rtl_fifo.o 驱动程序插入 Linux 内核时,该驱动程序将自己注册为 RTLinux 的一部分,并成为 Linux 驱动程序。一旦插入 Linux 内核,用户空间进程和实时任务都可使用实时 Linux FIFO(如图 2 所示)。

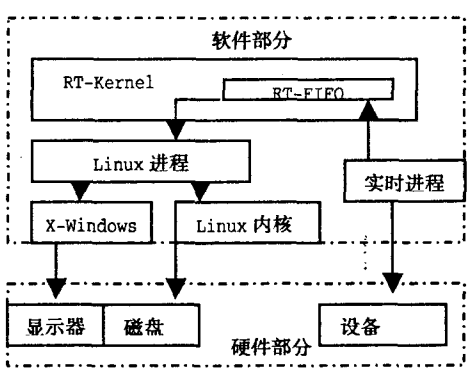

图 2　实时 FIFO 通信

任何硬实时任务都是在 RTLinux 的控制下运行的,该任务一般可执行周期性任务、处理中断并与 I/O 设备驱动程序通信,以采集或输出模拟和数字信息。当实时任务需要告诉用户进程有一个事件将发生时,它便将这一消息送给实时 FIFO。每一个 FIFO 都是在一个方向上传送数据:从实时任务到用户空间,或反之。因此,双向通信需要使用两个 FIFO。

下面说明实时 FIFO 的使用方法:

(1)FIFO 是在 init_module()时调用 rtf_create()创建的,在 cleanup_module()中调用 rtf_destroy()撤销。

(2)在创建 FIFO 时,可以调用 rtf_create_handler()注册该实时 FIFO 的处理程序。每次 Linux 进程读或写该 FIFO 时,rtl_fifo 驱动程序都要调用该处理程序。

(3)对 FIFO 的写入和读出是通过函数 rtf_put()和 rtf_get()来实现的。任何读出或写入实时任务一侧的操作都是非模块操作,因此 rtf_put()和 rtf_get()都立即返回,而不管 FIFO 状态是什么。

如上例数据采集,可以注册一个 FIFO。当有中断表明数据到来时,该中断服务例程可以调用 rtf_put()将数据写入 FIFO,而 Linux 进程就可以使用 rtf_get()从 FIFO 中读出数据进行处理。

5　结束语

RT-Linux 的双内核解决方案能够有效地解决 Linux 内核的关中断问题。无论 Linux 内核处于什么状态,它处于内核态还是用户态,它是禁止中断或是开启中断,它位于 spinlock 保护之内还是之外,实时内核都可以响应中断。这就解决了由于 Linux 关中断而导致的中断响应延迟的问题。双内核的解决方案对内核的修改较少,不容易产生错误,而且可以实现硬实时,适用于对实时性能要求较高的场合。

参考文献:

[1] Stallings W. 操作系统——内核与设计原理[M]. 魏迎梅,王　涌等译. 北京:电子工业出版社,2001.

[2] 郭玉东. Linux 操作系统结构分析[M]. 西安:西安电子科技大学出版社,2002.

[3] Proctor F. Introduction to Linux for Real-Time Control[R]. Spain: National Institute of Standards and Technology,2002.

[4] Wang Yu-Chung, Lin Kwei-Jay. Some Discussion on the Low Latency Patch for Linux[R]. Irvine: Department of Electrical and Computer Engineering, University of California, 2001.

[5] Yodaiken V. The RT-Linux Manifesto[R]. Department of Computer Science, New Mexico Institute of Technology, 2003:759-775.

(下转第 95 页)

要增大 N_S 以获得最大有效吞吐量 S，N_S 的选择具有门限性，即：

$$N_S=\begin{cases}1 & K>m_1\\ 2 & m_2<K\leqslant m_1\\ 3 & m_3<K\leqslant m_2\\ 4 & m_4<K\leqslant m_3\\ 5 & m_5<K\leqslant m_4\\ \cdots & \cdots\\ n & m_n<K\leqslant m_{n-1}\end{cases}$$

门限 m_n 的值由以下方程确定：

$$S_{MAX}(N_S=n,L_{DATA}^{OPT}(K,N_S=n))=S_{MAX}(N_S=n-1,L_{DATA}^{OPT}(K,N_S=n-1)) \quad (15)$$

综合 2.2、2.3 节的结论即可确定 N_S，L_{DATA}^{OPT}。

3　协议仿真

由于通信业务中数据占据相当大的份量，这里只考虑数据业务，数据到达服从 Poisson 分布。

假定所有节点业务负荷相同，表 1 列出主要仿真参数。

表 1　协议主要仿真参数

最大发送速率	50Mbps	RTS	104bit
发射功率	−8dbm	CTS	120bit
网络规模	100m * 100m	ACK	120bit
最大数据单元长度	22000bit	热噪声功率谱密度	−86dbm
活动用户数	1～20	系统同步时间	10μs
控制帧速率	10Mbps(N_S=5)	最大重传次数	7

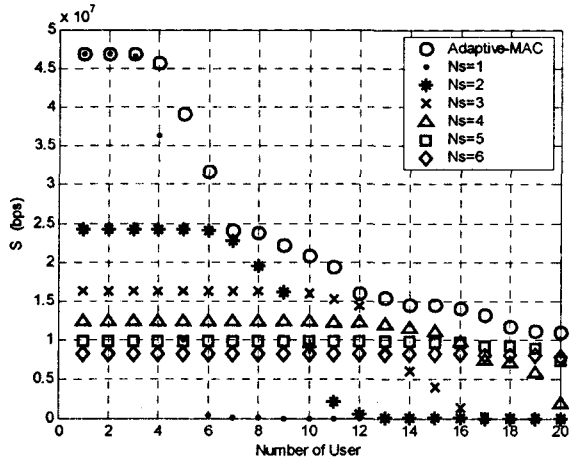

图 3　活动用户与吞吐量

如图 3 所示，仿真结果表明自适应 MAC 协议对系统性能有较大改善，在活动用户数量较低时，能保证较高的吞吐量，随着活动用户数量的增加，其有效吞吐量的下降速度明显低于使用固定速率和分组长度的 MAC 协议。

4　结束语

由于自适应 MAC 协议根据超宽带技术特点，充分利用了其数据速率可灵活调整的优势，并考虑了降低其同步时间相对于高速传输速率较长的不利影响，在接收端根据干扰程度自适应调整速率和数据帧长度，当干扰较小时，增大发送速率和数据帧长度，当干扰较大时，降低发送速率和数据帧长度，可最大限度改善基于超宽带技术的系统有效吞吐量，利于在 UWB Ad Hoc 网络中实现。

参考文献：

[1] Xiaojing H, Yunxin L. Performance of impulse train modulated ultra－wideband systems[J]. IEEE Trans. on Information Theory, 2001, 46(8): 2197－2417.

[2] 李晓冬，邹传云，陈跃波. 基于超宽带的高速 MAC 协议[J]. 微计算机信息，2006，13(4)：32－38.

[3] Parr B, Cho B, Wallace K, et al. A Novel Ultra－wideband Pulse Design Algorithm[J]. IEEE Communication Lett, 2003, 7(5): 219－221.

[4] 何昆鹏，李腊元. Ad Hoc 网络中按需路由协议的仿真与性能分析[J]. 计算机技术与发展，2008，18(3)：81－84.

[5] 刘　可. 无线局域网中的认证机制[J]. 计算机技术与发展，2008，18(1)：164－167.

[6] Guvenc I, Arslan H. On the Modulation Options for UWB Systems[C]//Proceedings of IEEE Military Communications Conference, Vol2. Monterey CA, USA, 2003. Piscataway NJ, USA: IEEE, 2003: 892－897.

[7] Li D. IEEE P802.15－03/141r3－TG3a. TI Physical Layer Proposal: Time－Frequency Interleaved OFDM[J]. IEEE Communication Lett, 2004, 8: 43－47.

[8] 杨菊英，吕光宏. 无线传感器网络分层路由协议研究[J]. 计算机技术与发展，2008，18(6)：115－118.

[9] 余　平，王汝传. 基于无线传感器网络的普适计算模型研究[J]. 计算机技术与发展，2006，16(4)：1－3.

[10] 雪　莲. 一种流量自适应 MAC 算法[J]. 鞍山科技大学学报，2007(4)：41－46.

(上接第 91 页)

[6] Heursch A C, Grambow D, Horstkotte A, et al. Steps Towards a Fully Preemptable Linux Kernel[C]//WRTP' 03, 27th IFAC/IFIP/IEEE Workshop on Real－Time Programming. Lagow, Polen: [s. n.], 2003.

[7] Dankwardt K. Comparing real－time Linux alternatives[J]. Linux Journal, 2003, 31(2): 26－34.