RFRTOS:基于 Linux 的实时操作系统*

李小群+, 赵慧斌, 叶以民, 孙玉芳

(中国科学院 软件研究所,北京 100080)

RFRTOS: A Real-Time Operation System Based on Linux

LI Xiao-Qun+, ZHAO Hui-Bin, YE Yi-Min, SUN Yu-Fang

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: E-mail: lxq@sonata.iscas.ac.cn

http://www.ios.ac.cn

Received 2002-09-09; Accepted 2003-03-04

Li XQ, Zhao HB, Ye YM, Sun YF. RFRTOS: A real-time operation system based on Linux. *Journal of Software*, 2003,14(7):1203~1212.

http://www.jos.org.cn/1000-9825/14/1203.htm

Abstract: Applications with real-time constraints are not only growing in the field of embedded system, but gaining popularity in the desktop environment as well. In order to satisfy the need of users in these aspects, RFRTOS real-time OS based on Redflag Linux is developed. In this paper, the real-time support about real-time scheduling, preempted kernel, and fine-grain timer in RFRTOS is mainly discussed. The experimental results on RFRTOS show that the improvement gives a better result for applications with real time constraint than original Linux kernel.

Key words: real-time OS; timer management; resource reservation; real-time scheduling; preempted kernel; SMP

摘 要: 随着实时应用领域的逐渐扩大,不仅传统嵌入式系统的需求日益紧迫,而且这种情况也渗透到桌面环境.为了满足国内用户在这方面的广泛需求,基于红旗 Linux 操作系统,开发了 RFRTOS 实时操作系统.主要从实时调度策略、内核抢占机制、细粒度定时器几方面讨论了 RFRTOS 的实时支持.实验结果表明,所作改进有效地提高了 Linux 的调度精度,满足了软实时方面的需求.

关键词: 实时操作系统;时钟管理;预留资源;实时调度;可抢占核心;SMP

中图法分类号: TP316 **文献标识码**: A

实时系统最重要的特点就是实时性,即系统的正确性不仅依赖于计算结果的正确性,还取决于输出结果时间的及时性.对于硬实时系统,如果不能按时完成,则可能造成灾难性的后果.在软实时系统中,偶尔的超时还可以容忍,但若超过一定比例也是不能接受的.

实时系统的发展已有 40 多年的历史,早期主要应用于工业控制、航空航天、国防等嵌入式硬实时领域.运行在这些系统中的操作系统被称为实时操作系统.

随着计算机的发展以及可以使用资源的日益增多,嵌入式系统的复杂性也在不断增加,系统要管理的资源

* **第一作者简介**: 李小群(1969－),女,陕西西安人,博士,主要研究领域为实时操作系统,嵌入式系统,中文处理.

也越来越多,过去只能在通用操作系统平台上实现的功能,如今嵌入式系统也需要实现:例如,机顶盒、多媒体播放等;另一方面,随着网络系统的发展,一些桌面系统也开始有实时要求,例如视频点播等.因此,目前的实时操作系统除了保留传统实时操作系统的功能以外,还需要根据用户的需求提供多种文件系统、网络、用户界面和数据库系统等扩展功能.

为了在实时操作系统中实现这些扩展功能,业内各机构提出了许多解决方案,其中一种解决方案是对通用操作系统进行改造,使其满足实时计算的需要,例如基于 WindowsNT 和 UNIX 等.随着 Linux 的风行,很多大学和研究机构开始基于 Linux 构建实时操作系统.这主要由于 Linux 具有如下一些优点[1~3]:

• 功能强大.支持多任务的进程调度;提供完全的内存保护机制和多种进程间的通信和同步机制,例如共享内存、消息队列、管道、信号、信号量和互斥锁等;支持包括多种网络协议的网络支持功能;支持包括网络文件系统的多种文件系统;启动和初始化进程控制得非常好,可以很容易地定制系统启动的服务和服务启动的顺序,例如,将 Linux 裁减为一个适合某类特定应用的嵌入式系统就是比较容易的.

• 开放源码.Linux 所有源代码开放,这不仅使得我们可以很好地理解它、使用它,更重要的是,如果需要,还可以改动它,以满足具体应用的需求.

• 支持多种硬件平台.Linux 已经被移植支持多种 CPU 构架,驱动程序也很丰富.一旦有新的硬件出现,开放源码团体或硬件厂商会很快提供驱动支持.

• 模块化设计.模块化的设计方案使得它很容易被裁减成小到可以支持一些资源有限的嵌入式设备.红旗公司就成功地将 Linux 运行在彩票机上.目前该系统已在全国各点铺展开.

• 函数接口符合国际和工业标准.由于其开放性以及使用了标准接口函数,许多其他开发源码团体为 Linux 提供了大量的开发工具和实用软件,例如,窗口系统、高级网络协议、视频/音频库、多种编程和脚本语言等.所有这些使得实时系统的开发者可以很快地得到他们所需要的软件,并将精力集中在他们的核心工作中.

Linux 虽然具有以上优点,并且符合 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(memory locking)、实时信号等实时功能,但由于其最初的设计目标为通用分时操作系统,因此作为一个实时操作系统,Linux 仍然存在如下缺陷:

• Linux 的调度算法是非抢占式的.也就是说,Linux 不能保证在任一时刻系统所运行的进程是具有最高优先级的进程,这主要是由于被动调度和优先级反转[4]等问题造成的.

• 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,系统无法对中断作出响应,而非周期实时进程大多是由中断作出响应的,对于周期性实时进程而言,也需要调度模块来调度运行,而调度模块的执行恰恰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.

• 针对一般情形,为了提高资源利用率和系统整体处理能力所做的优化,也在响应时间上造成了很多不确定性.比如,请求分页的内存管理机制使得代码、数据被调入的时间成为不确定的,而磁盘操作的缓冲机制又给读取文件的时间带来不确定性.在实时系统中,这些不确定性是不能容忍的.

• 时钟粒度粗糙.时钟管理是操作系统的脉搏,操作系统环境建立之后,任务的执行和中止在很多情况下都是由时钟直接或间接唤起的.时钟也是许多操作系统基本活动的基准.系统用它来维持系统时间,监督进程运行,它还是进程调度的重要依据.通常,Linux 的时钟粒度被设置为 10ms,而实时应用一般都需要微秒级的响应精度,10ms 的时钟粒度显然不能满足实时应用的需求.

1 基于 Linux 内核的实时操作系统简介

从以上不足可以看出,Linux 的实时支持是很弱的,但是由于其具有源代码开放等优点,近年来人们对 Linux 内核的实时改造提出了一些方案和设想,主要有 KURT,RT-Linux 等.它们采用了不同的思路和技术方案,各有优劣,以下将简要地加以介绍.

1.1 KURT

首先是 University of Kansas 计算机及电子工程系信息和远程通信技术中心的 KURT(Kansas University Realtime OS).该技术中心研制 KURT[5,6]的目的是为了满足他们在 ATM 及多媒体方面研究工作的需求.这些研究工作对操作系统提出了特殊要求,其特殊性在于,既要求有很高的实时性,如与时间相关的 QoS(服务质量)保证,又要求全面的操作系统服务.分时系统无疑可以满足后者,但无法满足前者,即便实现了 POSIX1003.1b,其实时性依然显得太弱;而专用实时系统可以满足前者,对于后者却又无能为力.因而 KURT 的研制者认为,软实时或硬实时操作系统均不符合他们的要求.为了兼具两方面的特性,他们专门提出了一个新概念:firm realtime.

KURT 的研制者通过对 Linux 核心进行改造来实现他们这个所谓的 firm realtime 操作系统.他们采用的方法比较简洁,没有大动干戈,却基本达到了目的.KURT 主要对 Linux 核心做了如下两点改动:

- 修改了时钟中断机制.在以 X86 为处理器的 PC 上,系统时钟可以达到的最高频率超过了 1MHz,但 Linux 通过对它编程,将时钟频率设定为 100Hz,即时钟中断间隔为 10ms.对于实时操作系统而言,这种时钟粒度太粗糙,无法满足实时应用的需求.如何解决这个问题呢?最容易想到的莫过于提高时钟频率了,然而,简单地提高时钟频率意味着时钟中断的相应处理过程将占用更多的处理器时间,从而使得整个系统的有效利用率急剧下降,所以这不是一个好办法.KURT 的办法非常巧妙,它改变了时钟中断的固定频率模式,通过重新设定,使得时钟以微秒为单位,在任何需要的时候产生中断.这样,既保证了响应时间,又避免了不必要的开销.
- 增加了新的实时调度模块.KURT 核心可以有如下 3 种调度状态:

(1) 普通态.内核不提供实时支持,这时与通常的 Linux 没有区别.

(2) 实时态.内核仅仅调度实时进程,非实时进程全部被挂起.

(3) 混合态.实时进程和非实时进程同时被调度.表面看来,混合态似乎也能保证实时任务被优先调度.其实不然,因为此时核心不可抢占的性质依然没有改变,所以,一个运行中的一般任务一旦进入核心,则很可能妨碍实时任务的及时调度,因此混合态引入了很大的风险.

经改动之后,KURT 获得了较强的实时性能,这时,原则上说,调度可以精确到微秒.此外,KURT 并没有对核心做更多的改动,例如,没有对核心的不可抢占性做改进.所以,不难看出,这个标称 firm realtime 的操作系统实质上依然是软实时操作系统,不过,KURT 的设计者通过精巧的方法,以不大的代价达到了设计目的.

1.2 RT-Linux, RTAI

RT-Linux(由 New Mexico Institute of Technology 计算机系研制开发)是一个基于 Linux 的硬实时系统[7].基于通用分时系统核心构造硬实时操作系统,这听起来有些不可思议,那么,RT-Linux 又是如何实现这一点的呢?

RT-Linux 的设计者充分意识到通过改造通用分时系统实现硬实时操作系统困难重重,几乎不可能.这也就意味着硬实时操作系统很难提供通用分时实操作系统所能提供的丰富的系统服务和完善的开发环境及调试手段等,而这些都是实时应用本身及其开发工作所希望获得的支持.RT-Linux 的设计者在认真分析了实时应用的整体特征后发现,绝大多数实时应用都可分为实时和非实时两部分.例如,对于实时数据采样分析而言,其数据采样过程必须是实时的,而分析之后的图表显示却可以是非实时的,而且这两部分分别具有如下特征:实时部分较少需要操作系统支持,甚至可以不需要这样的支持,至少应用程序的设计者可以做到这一点;而非实时部分一般则需要相应的操作系统支持,有这样的支持将会极大地方便应用的开发.基于这种认识,RT-Linux 的设计者便确定了如下设计思想:构造一个简单的硬实时内核,应用的实时部分作为实时进程直接运行在这个硬实时内核之上;原来的常规 Linux 核心这时作为一个优先级最低的任务,也为这个实时内核所调度,应用的非实时部分作为非实时进程运行在 Linux 核心之上,从而可以获得 Linux 核心所提供的一切服务.为了做到这一点,RT-Linux 采用了如下方式:

(1) 对 Linux 核心进行改动,将其与中断控制器隔离,不再允许它任意关中断.核心中的所有中断操作指令都被替换为相应的宏,可以简单地理解为,这时的开中断、关中断指令实际上仅仅变更一个中断状态标志的值,并不真正改变中断状态.此时,中断控制器由实时核心控制,所有的中断首先被实时核心所截获,并进行相关的中断处理,然后才把中断“传”给 Linux 核心.这样,Linux 核心的一切活动都无法导致中断被关闭,也就无从影响

实时核心的任务调度,从而保证了核心的硬实时性.同时,容易看出,这种方法依然维护了 Linux 核心中数据结构的完整性.

(2) 改变了时钟中断机制.与 KURT 一样,RT-Linux 也需要用粒度更细的时钟.事实上,RT-Linux 采用的方法与 KURT 是一样的,这里不再重复.

(3) 提供实时调度.人们对实时任务调度问题进行了长期而深入的研究.实际上,这是实时操作系统领域内投入最多、研究最为透彻的一个方面.目前已经提出了形形色色的调度算法.这些算法适用于不同类型的实时应用场合,各有千秋.但是,要在一个具体实时操作系统内部实现一个“万用”的调度模块也不是不可能的,只是效率会低些.RT-Linux 在这方面采用了非常灵活的做法.它把调度模块的编写任务交给了系统的使用者,使用者编制的模块可以很容易地被加入到系统中.这样,使用者可以根据自己的不同需求,采用适宜的调度算法.当然,为了方便起见,RT-Linux 自身也提供了两个实时调度模块.

(4) 实时进程与非实时进程间的通信.如前所述,RT-Linux 的全部设计思想基于实时应用的划分.这里,一个实时应用被划分成一个运行于实时核心之上的实时进程以及运行于 Linux 核心之上的分时进程.于是,实时进程与非实时进程间的通信就成了一个必须解决的问题.RT-Linux 通过建立特殊的命名管道来解决这个问题,这样便在实时进程与非实时进程之间建立了数据传输机制.

RTAI[8]的方法与此类似,明显的不同之处在于,它定义了一组 RTHAL(real-time hardware abstraction layer).RTHAL 将 RTAI 需要在 Linux 中修改的部分定义成一组程序界面,RTAI 只使用这组界面与 Linux 沟通.这样,可以把对内核源码树的改动降低到可以控制的程度.其优点是,可以避免 RT-Linux 对内核源码改动过大的问题,减少跟踪移植到新版 Linux 的工作量.

RT-Linux 和 RTAI 的这种双内核机制的优点是极低的中断延迟和转换时间,而且以一种非常简单的方法实现了硬实时支持,使得实时应用程序的运行时间得到保证.但缺点也是显然的,所有实时任务都必须用内核模块的格式书写,从而导致实时应用的开发非常复杂,要求用户必须非常熟悉 Linux 内核和设备驱动程序的互动.另外一个重要隐患就是,由于实时任务在核心中运行,因此没有内存保护,不当的编程可能会导致内核的崩溃,同时,实时程序的调试也很困难.

RT-Linux 和 RTAI 技术开创了 Linux 的硬实时支持功能,其对硬件中断的接管方法对我们有一定的借鉴意义.然而,它们提供的调度机制还不能满足复杂应用场合的需要,对 Linux 原有资源的利用也不够方便.

1.3 REDICE-Linux

REDICE-Linux 是美国 REDSonic 公司推出的产品[9].该产品整合了加州大学欧文分校的开放源码项目 RED-Linux 和 RTAI.RED-Linux 支持用户层的实时程序,而 RTAI 支持内核层的实时程序.

完全抢占型内核虽然使得实时程序可以在用户层执行,同时享受到普通 Linux 进程的优势,但是会带来过长的延迟时间,因此,RED-Linux 通过直接修改 Linux 内核源码,将内核中较长的例程剖分为较小的代码块,并且有选择地加入了有限的抢占点(preemption point),内核在抢占点就可以被抢占,从而减小内核的抢占延迟,将标准 Linux 内核修改为可抢占型内核.

另外,RED-Linux 的调度架构使得 RED-Linux 可以符合多种不同复杂度系统的调度需求.基本上,它分成调度器(dispatcher)和分配器(allocator)两部分,分配器在用户态执行,而调度器在内核态执行.分配器可以是中间件的一部分,负责将应用程序的资源请求转换成内核可以理解的形式.调度器作为一个内核模块存在,可被动态加载,由其最终决定进程的执行顺序.

另外,RED-Linux 提供了 3 种调度策略:

(1) 优先级驱动的调度策略(priority-driven scheduling,简称 PD),以静态或动态的任务优先级作为参数提供给调度器,调度器仅以此作为寻求下一个执行任务的依据.

(2) 时间驱动的调度策略(time-driven scheduling,简称 TD).以当前时间作为寻求下一个执行任务的依据.

(3) 共享驱动的调度策略(share-driven scheduling,简称 SD).这是一种近来越来越受到关注的实时调度模式,基于 GPS(general processor scheduling)的算法,系统给每个实时任务分配一定的共享资源,参与调度的系统

中的所有实时任务按照一定比例共享计算资源.

RED-Linux 由于内核不可完全抢占而只能作为一种软实时方案,但它的调度架构是扩展 RTOS 适用范围的有益尝试.

这种两种策略组合的方式的一个比较大的隐患就是,由于存在 RTAI 在底层不断地偷取或抢占 Linux 运行进程的时间片,用户层的实时进程不能得到及时的响应和调度,同时,也增加了进程运行的不可预测性,从而失去了实时的意义.另外,其调度框架增加了系统负荷延迟,最多可达 100 多微秒.

2 RFRTOS 的设计与实现方案

为了改进 Linux 的实时性能,我们在 Linux 核心 2.2.17 上设计并实现了处理器预留资源、时钟粒度的细化、调度效率的提高、优先级继承协议(PIP)的支持、核心可抢占性、SMP()的支持、互斥锁机制的改进等多方面的实时支持功能,Linux 的 schedule()函数几乎进行了重写.例如,为了提高调度效率,进程就绪队列和实时定时器队列等都采用了堆的数据结构管理.该系统(红旗实时操作系统 RFRTOS)已经通过最后测试和文档整理阶段,并作为红旗公司的嵌入式实时产品投放市场销售.

2.1 时钟粒度的细化

Linux 内核在每次中断服务完毕返回用户空间之前都要检查是否需要重新调度,若有需要就进行进程调度.因此,我们可以将时钟中断看作是一个抢占剥夺点,当时钟中断到来时,如果有更高优先级的任务在等待运行,则当前运行的任务被抢占.因此时钟中断的间隔是决定实时任务能否被及时响应的一个重要因素,遗憾的是,Linux 的时钟粒度非常粗糙(10ms),根本不能满足实时应用的需要,而简单地提高时钟频率又会造成不必要的开销.为此,业内研究人员提出了各种改进算法,其中以 KURT 最为典型,是成功的范例,它通过将周期性的时钟中断改为非周期性的时钟中断这种简单而且代价较小的方式,从而达到了设计目标.

我们借鉴了 KURT 的方法,但是与之不同的是,我们提供了与标准 Linux 核心时钟并行运行的一个具有精密刻度的实时核心时钟处理系统,与原 Linux 核心时钟区别开来.不但提高了系统的稳定性和效率,而且独立的实时核心时钟易于维护改进.

RFRTOS 支持如下几种实时定时器:

(1) Jiffy-tmr.标准 Linux 的 jiffies 定时器,其周期为 10ms.如果服务函数返回值为 1,标志到期定时器为 jiffies.

(2) Enforce-tmr.CPU 资源控制定时器.在系统初始化时,给每个 CPU 创建一个 Enforce-tmr 定时器.如果定时器到期,说明正在当前 CPU 上运行的实时任务的运行时间已经超过了其请求的预留资源,故根据实时任务的属性,将其降为普通进程并置 need_resched 为 1;或设置进程的实时优先级为–1,并置 need_resched 为 1,这样,当 schedule()函数被激活后,会将其移出进程就绪队列.

(3) Replenishment-tmr.补充时间配额定时器.系统为每个周期性实时任务创建一个 Replenishment-tmr 定时器,在每个周期结束时,该定时器为实时任务加满时间配额.

(4) Period_tmr.唤醒实时任务定时器.系统为每个周期性实时任务创建了一个 Period-tmr 定时器,每个周期的开始,该定时器将唤醒实时任务.

(5) Nanosleep_tmr.标准 Linux 中 nanosleep()系统调用的优化.该定时器唤醒调用 nanosleep()系统调用进入睡眠的进程.

以上所有实时定时器都是由堆的数据结构组织和管理的.在时钟中断的处理函数 do_timer_interrupt()(arch/i386/kernel/Time.c)中,调用相应的函数处理实时定时器队列.具体过程为判断堆顶定时器是否到期,如果到期,则调用其服务函数.

我们设定了一个进程的唤醒周期为 200ms,随后在 nanosleep()系统调用中记录进程唤醒时刻与睡眠之前的时刻之差作为实验统计数据.实验结果显示,与标准 Linux 相比,RFRTOS 的设计方案有效地提高了系统时钟的精度(30μs:10ms).另外,RFRTOS 细粒度时钟机制所带来的额外开销普遍被控制在 10μs 之内.

2.2 基于预留资源机制的实时调度方案

Linux 的调度策略基本上是从 UNIX 继承下来的以优先级为基础的调度[2,3].内核根据优先级为系统中的每个进程计算出一个反映其运行资格的权值(时间片),然后挑选权值最高的进程投入运行.在运行过程中,当前进程的资格随时间而递减,从而在下一次调度时原来资格较低的进程可能就更有资格运行了.如果所有进程的资格都变成了 0,就重新计算一次所有进程的资格.资格的计算主要是以优先级为基础,所以说是以优先级为基础的调度策略.

但是,为了适应各种实时应用的需要,内核在此基础上实现了 3 种不同的策略:SCHED_FIFO,SCHED_RR 以及 SCHED_OTHER.每个进程都有自己适用的调度策略,并且,进程还可以通过系统调用 sched_setscheduler()设定自己的调度策略.其中,SCHED_FIFO 和 SCHED_RR 为实时调度策略,调度函数在每次挑选新进程时,实时进程会被赋予很高的权值,即其实时优先级加上 1 000.除此之外的 SCHED_OTHER,则为传统的调度策略,比较适合交互式的分时应用.

从以上实现可以看出,Linux 对实时进程的支持非常有限,对进程完全没有资源使用方面的控制,例如,如果一个高优先级的实时进程需要很长时间才能运行完毕,那么其他进程就没有办法得到及时的响应.

为此,我们采用预留资源机制[10,11]为实时进程提供及时的、有保障的资源保护访问模式,实时任务优先级的确定采用基于固定优先级的 RM 或 DM 调度策略,非周期性任务采用带宽保留算法,并增加了可调度性测试等.相应地,调度函数 schedule()也作了改进,为了提高查找效率,进程就绪队列改为堆的数据结构,挑选新进程的算法也作了改动.

在 RFRTOS 中,有实时要求的进程面对的是一个或数个资源集合:处理器资源、硬盘(I/O)资源、网络资源等.进程在这几种资源集合中定义自己的预留部分(reservation),而不同的调度器(scheduler)根据当前进程预留资源的消耗状态决定调度的换入和换出:预留资源耗尽则将执行进程调度换出;根据较早到期的预留资源和等待调度进程的优先级综合考虑调度换入.不同的预留资源根据不同的调度器选择,而一个进程也可能因多个预留资源而在不同的调度策略上加以选择.CPU 资源属于经典的实时分析和应用范畴,而其他两种资源则由于牵涉问题较多,可能有不同的观点和实现,比如,网络资源在 Linux 上就有数据包调度等多种算法,对此我们暂时回避.

在 RFRTOS 中,主要通过资源集(resource-set)、抽象预留资源(reserve)和具体 CPU 预留资源(CPU-reserve)等方式实现了对进程占用处理器资源的控制.当实时进程申请运行时,只有通过可调度性测试证实系统可以满足其申请的预留资源才可进入进程就绪队列.如果实时进程被调度运行时耗尽了系统为其预留的处理器资源,即其申请要占用 CPU 的时间,则当前进程所在的 CPU 的强制定时器就会到期.该定时器到期执行的服务程序按照用户指定的模式决定处理进程的方式:将进程降为普通进程;或在将进程降为普通进程的同时,将其 rt_priority 设为−1,这样,当 schedule()函数被调度时,进程将被移出进程就绪队列;在调整优先级时,设置 need_resched 为 1,以激活 schedule()函数的调度.

实验结果显示,在标准 Linux 系统中,除非显式地声明放弃 CPU,进程总是尽可能多地占用 CPU,即 $\frac{C}{T} \gg 1$.同时,由于进程运行队列的变化会引起其他进程的调度执行,对于我们所关心的进程,某一时刻的执行又是不能保证的,因而 Linux 的调度机制对实时应用的支持是很弱的.而经过改进的实时核心则显著地提高了调度的准确度,并且将进程的执行时间误差普遍控制在 30μs 之内(如图 1 所示).

2.3 SMP的支持

随着实时应用的飞速发展,人们对实时操作系统的性能也提出了更高的要求.单处理器计算机系统已不能很好地满足某些复杂实时应用系统的需要,开发支持多处理器结构的实时操作系统已成为发展方向.这方面比较成功的有 pSOS.至于分布式实时操作系统,国外一些厂家虽然已经推出部分产品,如 QNX,但分布式实时操作系统的研究尚未完全成熟,特别是在网络实时性和多处理器间任务调度算法上还需进一步研究.因此,我们的系统中主要考虑支持 SMP.

在单 CPU 架构上,传统的分时系统需要一个外部的时钟源来完成关键的控制功能,作为分时进程记帐的依

据,同时向系统提供定时器功能.在我们已经实现的基于单CPU架构的实时系统中,利用该时钟源实现了实时调度必不可少的高精度时钟.在单 CPU 架构中,这样的时钟源一般作为 CPU 外围芯片组的一部分,可编程,并与系统的中断控制器相连接,以产生定时的中断信号,如 8259A 中断控制器.但在 SMP 架构中,如果仍采用 8259A 中断控制器,那就只能静态地把所有的外部中断源划分成若干组,分别把每一组连接到一个 8259A,而 8259A 则与 CPU 有一对一的连接.然而这样处理就达不到动态分配中断请求的目的,也使硬件设计变得很不简洁.因此,Intel 为其奔腾处理器设计了一种更为通用的中断控制器,称为“高级可编程中断控制器”(advanced programable interrupt controllor,简称 APIC).另一方面,考虑到处理器之间中断请求的需要,每个 CPU 实际上都需要一个本地的 APIC,同时,在 SMP 结构中还需要一个外部的、全局的 APIC.

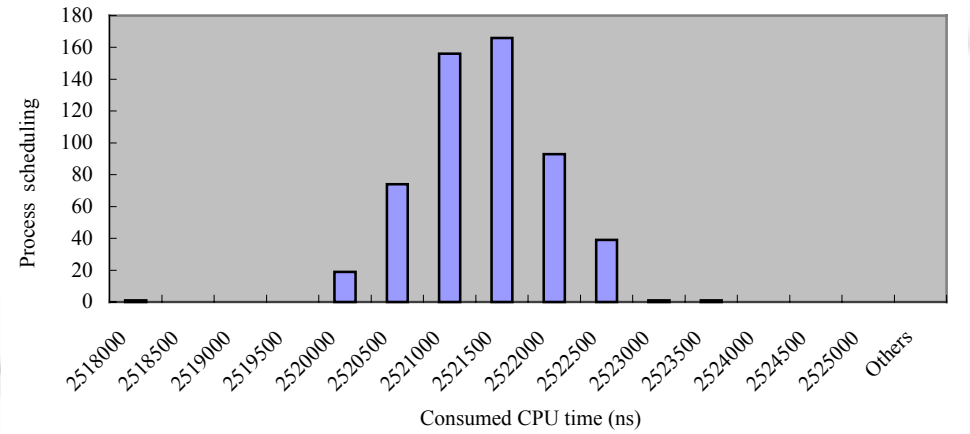

Fig.1 Real-Time process scheduling

图 1 实时进程的调度执行

在 SMP 结构的系统中,本地 APIC 本身就包含了一个 32 位可编程定时器,所以各个 CPU 可以选择使用本地的时钟中断源,也可以选择使用外部的全局时钟中断源.各个 CPU 对本地 APIC 时钟中断的服务程序为 void smp_local_timer_interrupt()函数.

但是,如果我们利用这一机制来实现高精度时钟,将为每个 CPU 设置局部于该 CPU 的实时定时器队列,并将周期性时钟中断模式改为非周期时钟中断模式,那么系统的开销可能会过大.因此,我们将实时定时器队列放在全局中断控制器上,当系统启动时,给每个 CPU 创建一个强制定时器,当调度一个新进程投入运行时,将该进程请求占用 CPU 的时间写入与 CPU 对应的强制定时器的到期时间.

对于多处理器调度算法,RFRTOS 设计并实现了由用户指定 CPU 的静态实时多处理器调度算法.相应地,RFRTOS 对预留资源机制也作了改变.在系统启动时,为每个 CPU 创建了一个数据结构,该结构中最重要的两项为运行在该 CPU 上的进程的 CPU 预留资源和控制在当前 CPU 上运行的进程的强制定时器.

2.4 可抢占核心

尽管由于种种原因,导致了 Linux 对实时支持的缺陷,但其中最主要的原因就是 Linux 核心的不可抢占性.

在内核中,一个进程可以通过 schedule()函数自愿地启动一次调度.除此之外,非自愿的强制性调度只能发生在每次从系统调用返回的前夕以及每次从中断或异常处理返回到用户空间的前夕.但是,如果在系统空间发生中断或异常是不会引起调度的.这种方式使内核实现得以简化.早期的 Unix 内核正是通过这种方式简化其设计与实现的,否则,内核中所有可能为一个以上进程共享的变量和数据结构就都要通过互斥机制加以保护,或者说都要放在临界区中.

在实时应用中,某个中断的发生可能不但要求迅速地中断服务,还要求迅速地调度有关进程投入运行,以便在较高层次上对事件进行及时的处理.可是,如果这样的中断发生在内核中,本次中断返回是不会引起调度的,而要到最初使 CPU 从用户空间进入内核空间的那次系统调用或中断(异常)返回时才会发生调度.如果核心中

的这段代码恰好需要较长时间完成,或者连续又发生几次中断,就可能将调度过分地延迟.良好的内核代码可以缓解这个问题,但不能从根本上解决问题.所以,这是一个设计问题,而不是实现问题.

另外一个问题是优先级反转[4].在 Linux 中,在核心态运行的任何操作都要优先于用户态进程,这就有可能导致优先级反转问题的出现.例如,一个低优先级的用户进程由于执行软/硬中断等原因而导致一个高优先级的任务得不到及时响应.

RFRTOS 通过如下 3 种方式实现了 Linux 核心的可抢占性:

(1) 中断处理线程化

简单的设计思路描述如下:

硬件层:IRQ_task 在中断初始化后,被赋值为相应的中断服务线程的结构.每当硬件中断发生时,底层中断处理函数唤醒相应的中断服务线程,设置调度标志,随后的 schedule()函数将在合适的时机选择执行服务线程.中断服务线程(对用户而言可以说是驱动程序)初始状态主动放弃 CPU,schedule()函数将选择其他进程,直到它被底层的相应中断唤醒,进入中断服务实体任务,如读写 I/O 端口等.

在以上机制中,每个中断线程也被赋予不同的优先级,用户进程也可以和中断线程一起竞争 CPU 时间.因此,在某些特殊情况下,一些中断线程的优先级可以被设置得低于用户进程的优先级,例如,在发生优先级反转的情况下.

(2) 强制二元信号量的互斥锁机制

Linux 中的互斥锁机制采用了硬件锁的方式,互斥锁普遍基于 spin_lock_irq /spin_unlock_irq 的原语引申,在单 CPU 构架中,spin_lock_irq /spin_unlock_irq 的作用是开/关中断,这种互斥锁通过禁止中断,回避多线索引起的并发冲突,但这是代价非常高的一种互斥方案:任务只能交替执行.另外,该方法不能适用于多处理器的情况,当系统中的处理器多于 1 个时,就有可能有 1 个以上的进程同时执行,在这种情况下,禁止中断依然不能解决问题.在多处理器结构下,Linux 的 spin_lock 原语增加了 testset 指令集的软件互斥实现,在一定程度上提高了互斥锁的效率.

这两种实现属于“轻量级”的互斥锁:实现简单,不涉及进程切换,因而在核心级和进程级空间均可用.这些优点使得其适用于 Linux 的宏内核设计,故被普遍采用,但在以下几方面明显地存在着缺陷:

- 使用忙等待;
- 可能出现饿死(starvation);
- 死锁(dead lock)的可能性较大.

在进程级上,Linux 也提供了基于信号量的锁机制,这与我们以下的思路相近,但它的实现较为简单,不能支持多种互斥锁协议,这里我们不再赘述.

我们在 RFRTOS 中采用了强二元信号量的互斥锁机制改进方案,实现了较为高效的互斥锁,且这种设计易于实现复杂的互斥锁协议,如 PIP,PCP.这种互斥锁机制属于“重量锁”,它的实现是与进程调度相关的.算法采用了一个阻塞进程队列 queue,由于 wait 调用而被阻塞的进程将进入这一队列.当持有锁的进程退出关键数据区时,发出 signal 调用,它在阻塞进程队列中选择一个被阻塞的进程调度执行.

这种互斥锁的设计与实现依据一定的互斥锁协议选择进程,因而避免了可能出现的某些进程饿死的现象.在这种改进的互斥锁方案的基础上,我们进一步实现了针对实时任务的优先级继承协议(PIP),改善了优先级反转(priority inversion)问题.

这种机制也是可抢占核心必不可缺少的.在标准 Linux 中,其所以设计成在核心态运行的进程是不可抢占的一个重要原因是,因为核心有一些临界资源是所有进程共享的,因此可抢占核心机制就必须保护这些临界资源.为了支持可抢占核心,我们通过强二元信号量的互斥锁改进机制实现了对临界资源的保护.

(3) 调度的时机

在以上两种机制支持的基础上,我们在每次中断返回时,都会判断重新调度标志是否需要重新调度,如果需要,则调用 schedule()函数.

最终的实验结果显示,改进后内核的调度精度有了明显的提高,首先表现在唤醒执行时间精度的提高,偏差

值大于界限的次数较少.另外,平均偏差比较表明,唤醒时间与平均值的离散度也普遍较低.

3 实验测试

本文中所有实验的运行环境均为 CPU:PII400,内存为 128M,硬盘为 5400 转的 15G 的 IBM 硬盘.

限于篇幅,本文已将大多数实验结果在上文各节中给出.图 1 给出的是一个对无限循环进程的控制执行结果(周期为 10ms,每个周期的运行时间为 2.5ms).

从图 1 可以看出,RFRTOS 对实时进程占用 CPU 资源的时间误差普遍控制在 30μs 之内,这主要包括定时器的响应和处理、调度函数的执行等.而在标准 Linux 中,如果一个优先级最高的实时任务不显式地声明自愿放弃 CPU,该任务就会一直占用 CPU,直到自愿放弃为止.因此,RFRTOS 的改进效果是显而易见的.

为了测试由于 RFRTOS 改进所造成的系统负载的增加,我们分别在标准 Linux 和 RFRTOS 下测试了两个 IDE 接口以及鼠标和键盘的最大中断处理延迟时间,测试结果比较如图 2 所示.实验结果显示,RFRTOS 以不大的代价获得了对实时应用的良好支持.

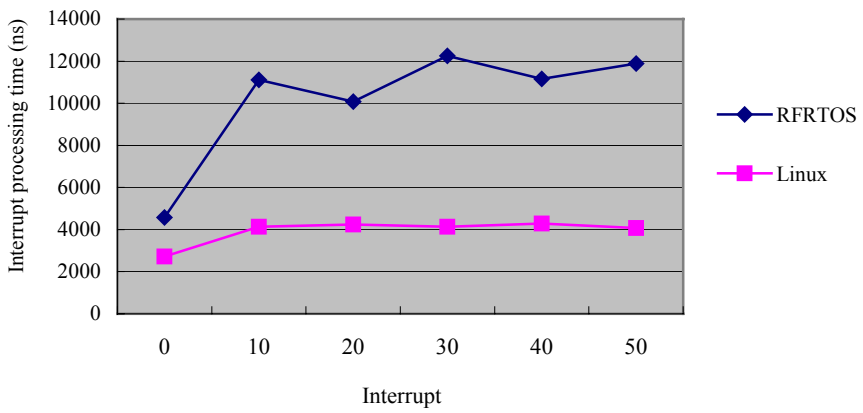

Fig.2 The interrupt processing time of standard Linux and RFRTOS
图 2 标准 Linux 和 RFRTOS 中断处理时间比较

4 结 论

Linux 虽然是通用的分时操作系统,但由于其功能强大、源码开放以及免费等优势使得改进 Linux 核心适合实时系统成为一种选择.虽然目前有 KURT,RT-Linux 等几种基于 Linux 的实时改造方案,并且这几种方案各有其可借鉴之处,但如果综合考虑任务响应时间、核心可强占性、中断控制和实时调度策略等影响操作系统实时性能的各个方面,它们还不能很好地满足实时应用的需求.

为了满足用户在工控、嵌入式、多媒体等领域的广泛需求,在借鉴目前已经提出的各种 Linux 实时化技术的基础上,我们设计并实现了基于红旗 Linux 的 RFRTOS 实时操作系统.该系统主要在处理器预留资源、时钟粒度的细化、调度效率的提高、优先级继承协议的支持、核心可强占性、SMP 的支持、互斥锁的设计与实现等几个方面进行了改进.实验结果显示,与其他改进方案相比,RFRTOS 以较小的中断延迟,在调度精度的提高、进程运行的控制以及中断事件的及时响应等实时性能方面获得了良好效果,达到了预期的目标.

References:

[1] Barabanov M, Yodaiken V. Introducing real-time Linux. Linux Journal, 1997,(34):19~23.

[2] Mao DC, Hu XM. Linux Kernel: Scenarios and Analyses. Hangzhou: Zhejiang University Press, 2001 (in Chinese).

[3] Li SP, Zheng KG. Linux Operation System and experiment tutorial. Beijing: China Machine Press, 1999 (in Chinese).

[4] Lui S, Rajkumar R, Lehoczky J. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers, 1990,39(9):1175~1185.

[5] Hill R, Srinivasan B, Pather S, Niehaus D. Temporal resolution and real-time extension to Linux. Technical Report, ITTC-FY98-TR-11510-03, Information and Telecommunication Technology Center, Electrical Engineering and Computer Science Departmaent, University of Kansas, 1998.

[6] Srinivasan B, Hill R, Pather S, Niehaus D. A firm real-time system implementation using commercial off-the-shelf hardware and free software. In: Proceedings of the Real-Time Technology and Applications Symposium. Denver, 1998. 112~119.

[7] Yodaiken V. The RT-Linux approach to hard real-time. 1997. http://rtlinux.lzu.edu.cn/documents/papers/whitepaper.html.

[8] Dissecting DIAPM RTHAL-RTAI (with some comparisons to NMT-RTL, here and there) (draft). Paolo Mantegazza Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano. http://www.aero.polimi.it/~rtai/documentation/articles/ paolo-dissecting.html.

[9] Wang YC, Lin KJ. Implementing a general real-time framework in the RED-Linux real-time kernel. IEEE Real-Time System Symposium, 1999. 246~255.

[10] Oikawa S, Rajkumar R. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In: Proceedings of the IEEE Real-Time Technology and Applications Symposium. Vancouver, 1999. 111~120.

[11] Rajkumar R, Juvva K, Molano A, Oikawa S. Resource kernels: A resource-centric approach to real-time systems. In: Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking. 1998.

附中文参考文献:

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

[3] 李善平,郑扣根.Linux 操作系统及实验教程.北京:机械工业出版社,1999.