嵌入式 Linux 系统实时性的研究

袁旭峰,李泽滔,李捍东,张希周
(贵州工业大学电气工程学院自动化系,贵州 贵阳 550003)

摘　要:通过对 Linux 内核及其在实时应用方面不足的分析研究,在细粒度微定时器、内核抢占机制、实时调度策略等几方面提出了改善系统实时性方法。

关键词:实时操作系统;Linux;嵌入式系统;内核;进程;调度策略
中图分类号:TP316.2;TP316.81　　**文献标识码**:B

0　前　言

嵌入式操作系统是指一种以应用为中心,以计算机技术为基础,软硬件可剪裁,适应应用系统对功能、可靠性、成本、体积、功耗要求严格的专用计算机系统[6]。嵌入式操作系统主要有 Palm OS、Windows CE、VxWorks 等等。但由于价格高,难以接受;可扩展性差,难以移植。Linux 遵循 GPL,具有源代码公开,定制方便,支持广泛的计算机硬件等许多优点。但是,Linux 内核是类 UNIX 分时系统,欲将其运用于实时性要求较高的控制领域,进一步提高其实时性是相当必要的。

本文从操作系统的角度出发,讨论了 Linux 内核在实时性方面的不足;分析了现有实时 Linux 内核实时化方面的突破及特点;提出了在构造嵌入式 Linux 系统时,提高实时性的方法。

1　Linux 内核在实时性方面的不足

Linux 系统是类 UNIX 系统,从实质上说,Linux 系统也是分时系统。其在实时性方面存在的问题有:

(1)Linux 分为用户态和内核态两种模式,进程运行在用户态时,实时进程具有高的优先级,能进行进程抢占,故可以较好的完成任务;运行在内核态时,如系统调用,实时进程不能抢占该进程。因此,从实质上来说,Linux 内核是非抢占式的。

(2)在定时器方面,有下列几方面缺陷:第一,Linux 的周期模式定时器频率仅为 100 Hz,远不能满足多种实时应用的要求。第二,软定时由时钟定时器完成,当软定时器较多时,势必引起共享时钟定时器的冲突。第三,Linux 中断句柄不可调度,但在实时系统中,期望能在一个可调度整体内处理这些中断句柄,从而能更有效的区分不同实时任务的紧迫度,分配不同的优先级。因此,单纯缩短时间片在对实时性能严格要求的场合是不受欢迎的。第四,增加系统负荷。

(3)Linux 进程采用多级轮转调度算法,该调度算法,仅能获得秒级响应时间,一个实时进程在一个时间片内未完成,其优先级将降低,从而可能造成到截止时间无法完成。

(4)Linux 虽然给实时进程提供了较高的优先级,但是,并没有加入时间限制。例如完成的最后期限、应在多长时间内完成、执行周期等等。同时,其它大量的非实时进程也可能对实时进程造成阻塞,无法确保实时进程的响应时间。另外,当有多个实时进程互斥请求共享资源时,由于其使用的同步原语不支持优先级继承协议(PIP),易产生优先级倒置。

最后,Linux 系统是一个通用的操作系统,以满足大多数用户需要为目的。而嵌入式系统是面向用户、面向产品、面向应用的,因此在构造嵌入式 Linux 系统时,可以根据具体的需要进行定制和修改,为 Linux“瘦身”,以完成目标功能为目的,而无须在其它方面尽善尽美。

2 当前 Linux 在实时内核方面的发展[1-4]

下面就当前改进 Linux 内核实时性能方面的思路和方法进行分析和讨论。

(1) 兼容内核方法。即含有一个兼容 Linux 内核,符合 POSIX 的 API 层的实时内核。该方法适用于拥有独立实时操作系统和兼容 Linux 的 API 开发商。但由于未使用 Linux 内核,无法充分利用 Linux 的优势,且采用非 Linux 内核模式,不遵循 GPL,故获得内核源代码费用非常昂贵。

(2) 双内核方法。双内核系统在硬件平台上增加一个实时内核,建立双内核,如现在流行的 RT-Linux. 但在提高其实时性时,难以保证功能上的完整性和对原有 Linux 资源的充分利用,且调度机制简单,不能满足复杂需要。在协调双内核工作时,寻求一种合理的实时和非实时的混合调度机制是亟待解决的问题。

(3) 内核修改方法。通过修改 Linux 内核源码来实现。如 RED-Linux 就采用该方法。在内核代码中增加抢占点,从而减少内核抢占延迟。然而,由于这些修改是局部,属于软实时,而非硬实时。

(4) 源内核方法。源内核方法是面向对象的方法,提出一个动态实时任务的执行应依赖于产生该响应的源事件,而非应用程序本身。该方法建立的实时内核能提供更快捷响应,提高实时性。但是,由于源事件的不确定性,使建立一个应用范围较广的实时系统较为困难。该方法适用于小型嵌入式系统的设计。

通过以上分析可知。软实时扩展空间较小,在控制领域,只有硬实时才符合 RTOS(Real Time Operating System) 的发展方向。而上述的硬实时方式也有自身的缺陷,尚须进一步的深入探讨和改进。

3 改进嵌入式 Linux 内核实时性的方法

从微定时器、可抢占式内核体系结构、实时任务调度策略三个方面对嵌入式 Linux 内核进行研究,并提出相应的提高实时性的方法。

3.1 微定时器

在分时系统中,定时器触发时钟中断,每个时钟中断是一次进程调度时机,Linux 中每个时钟中断触发 3 个函数协同工作,共同完成进程调度过程,它们是:

(1) shedule() 进程调度函数,根据当前调度策略从就绪队列中选择下一个将被执行的进程;

(2) do_timer(),该函数是时钟中断服务程序的主要组成部分,被调用频率就是时钟中断频率;

(3) ret_from_sys_call() 系统调用返回函数,当一个系统调用或中断完成时,该函数被调用处理一些收尾工作。

可以看出,定时器频率直接影响到系统响应速度和上下文切换的系统开销。Linux 的定时器提供 10 ms 的调度粒度,对于许多实时系统来说,通常需要微秒级的定时器,故此调度粒度不满足系统响应速度要求。

在这方面,我们的解决办法是在硬件上提供一个细粒度定时器。但是,在这种情况下,不能再采用单模定时器,应选择多模定时器。原因如下:1、定时器频率太小,将极大地增加系统消耗,不论是否有事件产生调度需要,CPU 总是频繁地被中断。2、一个实时系统并不要求系统在每一个微秒都产生中断,而是在任何一个微秒都能允许发生中断。因此在这种情况下,定时器的时钟频率不必固定,仅需设置为下一个事件要发生的时间,即一次模式。单模定时器只能使用于周期性较强的实时操作系统中。

为平衡系统功能,应该采用具有一次模式和周期模式混合的细粒度定时器。在引入细粒度定时器后,我们在 Linux 系统中增加对定时器模式和粒度的控制函数,根据不同的实时应用的响应时间需要,选择不同定时器模式和粒度。从任务响应时间的角度出发,即可控制内核抢占时间(KPT).

3.2 双内核可抢占式体系结构

嵌入式实时操作系统微内核是一般操作系统的子集,担负着任务管理、任务控制、任务间的通信,任务的同步与互斥、资源管理等诸多功能。在设计 Linux 内核体系结构时,我们采用双内核结构,见图 1. 其中,使用实时内核来运行实时任务,Linux 内核来运行非实时任务。如,利用实时内核运行一个实时任务来完成数据采集,另一个实时任务完成控制输出功能;同时利用 Linux 内核上运行的图形界面来进行数据显示。

如图 1,为解决 Linux 实现硬实时的最大障碍,使 Linux 内核为完全被抢占的采用混合内核方案,我们在嵌入式系统对 Linux 系统进行了如下修改:

(1)添加一个实时内核。由它管理中断,提供一些必要的功能,如底层服务创建、中断服务程序,并且为底层服务、ISR和Linux进程间的通信排队。其中实时核优先级较高,而Linux核的优先级较低。

(2)对Linux内核进行修改和定制。首先,对Linux进行定制,即"瘦身"。嵌入式系统是以完成应用功能为目的,我们可以根据实际需要定制Linux.以设计一个数据采集器为例,其功能是对现场数据进行采集,并传送到上位机,而且要求各个数据采集器之间实现多点通信。首先,根据该数据采集器的功能定制Linux如下:在进程管理方面,由于数据采集和传送,即多点通信均属于实时任务,为此我们设计进程调度时,采用可抢占式优先级调度,而且使采集和传送任务的优先级高于多点通信;在设备方面,去掉声显卡等无关设备的功能支持,保持串口、并口等相关设备的支持;在存储管理方面,去掉虚拟存储功能;在文件系统方面,采用单一文件支持,无须多文件格式;在网络方面,无须服务器功能,只须保持数据的上传功能和采集器之间的对等通信功能。其次,对已"瘦身"的Linux内核进行修改,主要有三方面:第一,在Linux内核中影响实时性能的地方增加控制点,使内核在控制点可以被抢占,减少内核抢占延迟;第二,将执行时间较长的系统调用划分为几个甚至十几个较小的块执行,使实时任务能随时中断非实时任务;第三,根据实际需要,增加部分功能。

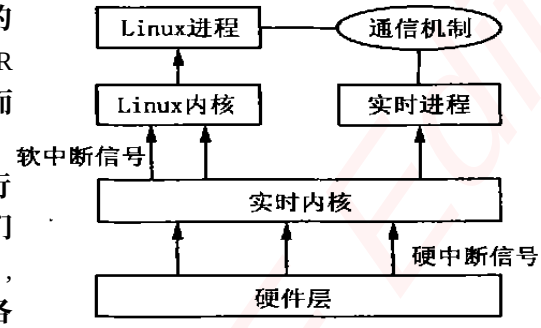

图1 双内核结构

在设计中,我们对实时要求强的应用编写成实时任务在实时内核上直接运行,而其它任务在Linux内核运行。Linux内核由于优先级较低,可以随时被实时任务抢占。而所有的任务是运行在实时内核还是Linux内核均由实时内核来决定。在运行每一个任务时,首先判断任务的来源,如来自优先级较高的实时事件,则判定为实时任务,直接在实时内核中运行。如该任务是非实时任务,则交与Linux内核运行。这部分在下一节详细讨论。由于,嵌入式系统中的实时任务都是通过计算机接口进入系统的,因而,可以在硬件上固定每一个任务或任务子集的优先级,如通过8259中断器来设计事件的优先级,当8259产生一个中断时,系统只是简单地在事件表中报告这一事件的发生,然后立即将对CPU的控制权交给实时内核,由其通过软件方法判断事件的属性及优先级,进而使之在相应的内核中执行,而不象一般Linux核心那样去查中断向量表并执行相应的中断服务程序。

可抢占式内核体系结构的设计中引入这种双内核设计方案,取得了较高的硬实时性能和丰富的系统调用支持。

3.3 实时系统调度策略

常用的实时调度算法有三种:优先级调度;时间驱动调度;共享驱动调度。在本文中我们提出混合调度算法,其结构示意图如下:

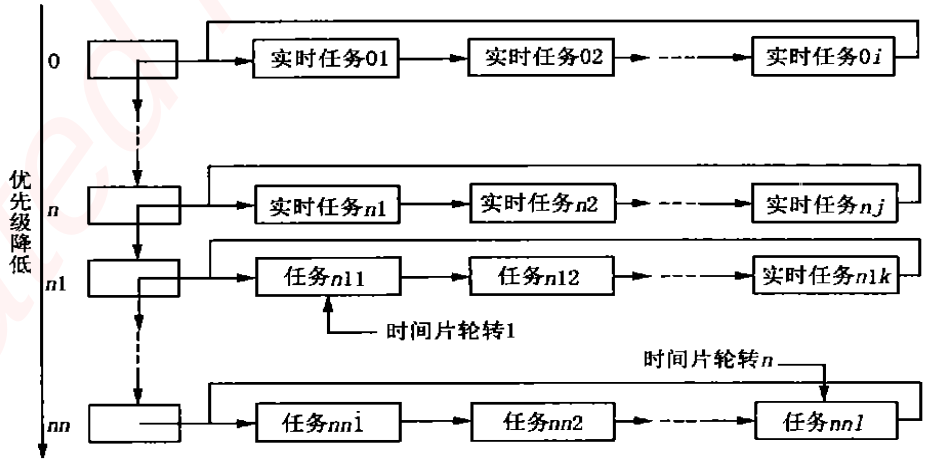

图2 可抢占式调度策略

如图 2,系统中支持 0 - *nn* 多级优先权,其中:0 - *n* 为实时任务占用优先级; *n*1 - *nn* 为非实时任务优先级; *n*1 = *n* + 1. 在 0 - *n* 优先级中,我们把不同的实时任务分为 0 - *n* 个实时任务集,分配给不同的优先权,采用不同的实时调度算法。相同优先级别的实时任务按 FIFO 原则放入队列中。在 *n*1 - *nn* 优先级中,所有的非实时任务均采用 Linux 内核本身提供的多级轮转反馈优先级调度算法。

为实现上述调度算法,我们采用二阶段调度法。

首先,在描述实时任务时,为每一个任务设置调度属性:优先级、起始时间、截止时间和权数组。针对每一个任务集,设置相应的优先级表、时间表和权值表。优先级表中含有任务或任务集的优先级,其设置在系统设计中根据实际对象确定;时间表中包含每一任务要求的起始时间和截止时间;权值表为同一任务集中任务产生的先后顺序或任务的紧迫度。

然后,分两个阶段实现调度:第一阶段,构造一个属性判别模块,判断当前实时任务的调度属性,即优先级、起始时间、截止时间及权值。随即进入调度选择模块,完成当前实时任务的调度算法选择,如优先级在 0 - *n*,则为实时任务,根据其优先级选择相应的实时任务队列;反之,进入 Linux 内核,进入相应的任务队列。第二阶段,进行实际调度。如为实时任务,根据其属性调用相应的实时调度算法;为非实时任务,则 Linux 内核调用本身的多级轮转反馈调度算法进行调度。

4 总结与展望

Linux 是类 UNIX 分时系统,在实时性支持方面存在很大不足,因此,开发基于 Linux 下的实时操作系统相当繁琐复杂,有许多技术难点须克服。本文在研究已有的前沿技术的基础上,从细粒度微定时器、内核抢占机制、及实时调度策略三个方面,提出了改善系统实时性能的方法,进而使 Linux 系统具备良好实时性能。这几种提高实时性能的方法相当复杂,尚有许多技术难题须进一步探讨和解决。但是随着其实时性能的逐步完善,以及嵌入式 Linux 系统在控制领域的深入应用,必然进一步促进新型控制算法的实现,开辟一个崭新的控制天地。

参考文献:

[1] Shui Oikawa, Raj Rajkumar. A Portable Resource Kernel in Linux[EB/OL]. http://www-2.cs.cmu.edu/rajkumar/recent-papers.html, 2002,6.

[2] Y C Wang, K J Lin. Enhancing the Real-Time Capability of the Linux Kernel[EB/OL]. http://www.computer.org/proceedings/rtcsa/9209/9209toc.htm,1998-10-29.

[3] Dave. The Use of Linux in An Embedded System[EB/OL]. http://www.linuxjournal.com/article.php?sid=3555,1999-12-01.

[4] 曹计昌,余隽. 关于提高 Linux 核心实时处理能力的讨论[J]. 计算机应用,2001,21(1):75-78.

[5] 邹思轶. 嵌入式 Linux 设计与应用[M]. 北京:清华大学出版社,1999.

[6] 王学龙. 嵌入式 Linux 系统设计与应用[M]. 北京:清华大学出版社,2001.

[7] 马季兰,冯秀芳,等. 操作系统原理与 Linux 系统[M]. 北京:人民邮电出版社,2001.

The Study of Real Time Capabilities for Embedded Linux System

YUAN Xu-feng, LI Zhe-tao, LI Han-dong, ZHANG Xi-zhou

(College of Electrical Engineering, GUT, Guiyang 550003, China)

Abstract: After researching and analyzing the kernel of Linux and limitation in real time capabilities, this paper brings forward the methods to improve the real time capabilities from three aspects: high resolution, kernel preemption and real-time scheduling policy.

Key words: real-time operating system; Linux; embedded system; kernel; process; scheduling policy

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发

10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码

9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备