

四种嵌入式实时操作系统关键技术分析*

季志均¹, 马文丽^{1,2}, 陈虎², 郑文岭^{1,2}

(1. 上海大学 电子生物技术研究中心, 上海 200072; 2. 南方医科大学 基因工程研究所, 广东 广州 510515)

摘要: 介绍了 RT-Linux, μ CLinux, μ C/OS- 和 eCos 四种源码公开的嵌入式实时操作系统 (Embedded Real-Time Operating Systems, ERTOS), 详细分析比较了关键实现技术——任务管理、任务及中断间的同步通信机制、存储器管理、中断管理等, 指出了不同应用领域所适合的 ERTOS。

关键词: 嵌入式系统; 实时操作系统; 任务调度; 任务同步与通信; 内存分配; 中断处理

中图分类号: TP316.2 文献标识码: A 文章编号: 1001-3695(2005)09-0004-05

Analysis of Key Techniques Based on Four Embedded Real-Time Operating Systems

Ji Zhi-jun¹, Ma Wen-li^{1,2}, Chen Hu², Zheng Wen-ling^{1,2}

(1. Bio-electronics Research Center, Shanghai University, Shanghai 200072, China; 2. Institute of Genetic Engineering, Nanfang Medical University, Guangzhou Guangdong 510515, China)

Abstract: The paper reviews four embedded real-time operating systems, namely the RT-Linux, μ CLinux, μ C/OS- and eCos. The key ERTOS procedures are compared and analyzed systematically, which include task scheduling, task synchronizing and communicating, memory allocating, interrupt handling and so on. The ERTOS's applications are discussed.

Key words: Embedded System; Real-Time Operating System; Task Scheduling; Task Synchronizing and Communicating; Memory Allocating; Interrupt Handling

1 前言

随着微电子技术、软件技术的飞速发展, 嵌入式系统广泛应用于生物医学仪器、智能汽车、通信设备、网络设备、仪器仪表、手持设备等领域, 成为当前研究与应用的热点。

按照系统对时间限制的满足程度, 实时系统可分为硬实时 (Hard Real Time) 系统和软实时 (Soft Real Time) 系统。硬实时系统是指那些对每个任务调度时间要求非常严格的系统, 如果不满足时间限制的要求, 则会对系统带来毁灭性的后果。软实时系统是指那些对每个任务调度时间要求不是很严格的系统, 即使超过了时间限制的要求, 也不会对系统带来毁灭性的后果。

当前, 实时操作系统 (Real-Time Operating System, RTOS) 逐步成为嵌入式系统的主流, 是嵌入式系统软件的最重要组成成分, 也是嵌入式应用软件的基础和开发平台, 所以对嵌入式实时操作系统 (Embedded Real-Time Operating System, ERTOS) 的研究变得尤其重要。全球范围内有数百种 ERTOS, 如 Nucleus Plus, QNX, LynxOS, pSOS, VRTX, VxWorks, Windows CE, Hopen (中国科学院软件研究中心研制), Palm OS, RT-Linux, μ CLinux, μ C/OS-, eCos 等。开放源码的 ERTOS 在成本和技术上有着独特的优势, 并占有越来越重要的地位。本文将介绍 RT-Linux, μ CLinux, μ C/OS- 和 eCos 四种源码公开的 ERTOS, 通过对它们关键实现技术的分析和比较, 为不同应用领域选择合

适的 ERTOS 提供基本依据。

2 ERTOS 简介

(1) RT-Linux

RT-Linux 由美国新墨西哥州大学计算机科学系 Victor Yodaiken 和 Michael Branovan 开发, 现由 FSMLabs 公司开发维护。RT-Linux 的设计思想是在标准 Linux 基础上实现抢占式的硬实时内核, 仅需支持底层任务创建、中断服务例程装入、底层任务通信队列、中断服务例程 (ISR) 和 Linux 进程。将 Linux 作为这个实时内核的一个优先级最低的任务来运行, 所有实时任务的优先级都要高于 Linux 本身以及 Linux 的一般任务, 所有任务都在核心地址空间运行。

(2) μ CLinux

μ CLinux 是一个完全符合 GNU/GPL 公约的项目, 完全开放源代码, 现在由 Lineo 公司支持维护。 μ CLinux 这个英文单词中: μ 表示 Micro, C 表示 Control, 所以 μ CLinux 就是 Micro-Control-Linux, 字面上的理解为“微控制领域中的 Linux 系统”。

μ CLinux 最大的特征就是没有内存管理单元 (MMU), 专门针对无存储器管理单元的中低档 CPU, 并且专为嵌入式系统做了许多小型化的工作。这就使得 μ CLinux 的内核同标准的 Linux 内核相比非常之小, 但是它仍保持了 Linux 操作系统的主要优点, 如稳定性、良好的移植性、强大的网络功能、出色而完备的文件系统支持, 以及标准丰富的 API 等。

(3) μ C/OS-

μ C/OS- 是一个著名的、源码公开的、抢占式的多任务实时内核, 由美国嵌入式系统专家 Jean J. Labrosse 用 C 语言编

收稿日期: 2004-08-16; 修返日期: 2004-09-28

基金项目: 国家自然科学基金资助项目 (39880018); 广州市重大科技基金资助项目 (199-Z005-001)

写, 专门为嵌入式应用设计的, 现由 Micrium 公司开发维护。 $\mu\text{C}/\text{OS-}$ 的源代码可供学习免费使用, 但是使用 $\mu\text{C}/\text{OS-}$ 的产品需要购买产品生产授权。 $\mu\text{C}/\text{OS-}$ 由 60 多个系统调用, 包括任务、时间、信号量、互斥型信号量、事件标志组、邮箱、队列和内存等管理功能。

(4) eCos

eCos 的全称是 “Embedded Configurable Operating System”, 源于 Cygnus Solution 公司, 后成为 Redhat 的嵌入式部门, 现由 eCosCentric 公司开发维护。eCos 是一个免费的、无版权限制的 (无版税)、源码开放的、面向深度嵌入式应用的实时操作系统。eCos 最大的特点是采用模块化设计, 可进行源代码级的裁剪配置; 提供可选择的多种调度器 (调度算法); 提供多线程管理函数; 提供丰富的同步原语; 提供可选择的内存分配策略; 提供定时器和计数器; 支持中断和延迟中断; 支持异常处理; 提供 ISO C 库和数学库; 具有开放的 API 接口, 支持 POSIX API, EL/IX 兼容和 μITRON 3.02 API 标准; TCP/IP 网络栈; 文件系统支持 JFFS2 Flash, RAM 和 ROM 格式; 支持远程调试。与其他四种 ERTOS 不同的是 eCos 将实时内核也作为可选配置之一, 当系统没有内核时, 它将作为单任务运行。

(5) ERTOS 的基本内容比较

对 ERTOS 的 API 特征、CPU 种类和存储容量需求、开发环境和工具等基本内容作一简单比较, 如表 1 所示。

表 1 四种 ERTOS 的基本内容比较

操作系统	RT-Linux	$\mu\text{C}/\text{Linux}$	$\mu\text{C}/\text{OS-}$	eCos
供应商	FEM Labs	Lineo	Micrium	eCosCentric
API 特征	POSIX 1003.13 API	Linux 常用的 API; TCP/IP 协议栈, 大量其他网络协议; 各种文件系统	60 多个系统调用; 商业化文件系统 $\mu\text{C}/\text{FS}$ 和 TCP/IP 协议栈	POSIX API; EL/IX 兼容; μITRON 3.02 API; TCP/IP 网络栈
CPU 位数	16 位以上	16 位到 64 位	8 位到 32 位	16 位到 64 位
CPU 种类	x86, PowerPC, StrongARM, Alpha, MIPS	Motorola 68k/ Dragonball/ Coldfire, ARM 7TDMI, PowerPC, Axis, ETRAX, Intel i960, PRISMA, MC68EN302	8051 系列、ARM 系列、PowerPC、x86、NEC V850 E、SPARC lite	ARM 系列、x86、StrongArm、Xscale、PowerPC、Motorola 68k/ Coldfire、MIPS、NEC V8xx、SPARC、SuperH、Matsushita AM3x
存储容量需求	1.5 MB ROM 和 4 MB RAM (MiniRT-Linux)	512KB RAM 和 1 MB ROM/Flash	几 KB 的 ROM 和 RAM	几十 KB 到几百 KB 的 ROM 和 RAM
支持 MMU	支持	不支持	不支持	不支持
开发环境	Linux	Linux	UNIX, Windows 或 Linux	Linux 或 Windows 的 Cygwin
开发调试工具	GNU 系列	GNU 系列	C 交叉编译器、汇编器和连接器	GNU 系列

3 ERTOS 的比较和分析

本文主要从任务管理; 任务及中断间的同步与通信机制; 存储器管理; 中断管理; 对 CPU 和存储器的需求等方面比较分析这四种 ERTOS。

3.1 任务管理

任务管理是 ERTOS 的核心和灵魂, 决定了 ERTOS 的实时性能。通常涉及到以下技术: 动态优先级、时间确定性、基于优先级抢占式调度、时间片轮转调度、多任务调度机制。

(1) 动态优先级

每个任务都有其优先级。任务越重要, 赋予的优先级应越高。任务的优先级 (特别是应用程序的优先级) 在运行时可以动态变化的, 则称之为动态优先级。

(2) 基于优先级抢占式调度

系统中每一个任务都有一个优先级, 内核总是将 CPU 分配给处于就绪态的优先级最高的任务运行。如果系统发现就绪队列中有比当前运行任务更高的优先级的任务, 就把当前运行任务置于就绪队列中, 调入高优先级任务运行。系统采用优先级抢占方式进行调度, 可以保证重要的突发事件及时得到处理。

(3) 时间片轮转调度

时间片轮转调度算法是让优先级相同的、处于就绪状态的任务按时间片使用 CPU, 以防止同优先级中某一任务独占 CPU。

(4) 多任务调度机制

任务调度算法是直接影响实时性能的因素。尽管调度算法多种多样, 但大多由单调速率 (Rate-Monotonic Scheduling, RMS) 调度算法和最早期限优先 (Earliest-Deadline First, EDF) 调度算法变化而来。前者主要用于静态周期任务的调度, 后者主要用于动态调度。在不同的系统状态下, 这两种算法各有优劣。在一般情况下, ERTOS 多采用基于优先级抢占方式与时间片轮转调度相结合的调度。

(5) 时间的可确定性

ERTOS 函数调用与服务的执行时间应具有可确定性。系统服务的执行时间不依赖于应用程序任务的多少。基于此特征, 系统完成某个确定任务的时间是可预测的。

四种 ERTOS 任务调度机制上的比较, 如表 2 所示。

表 2 四种 ERTOS 的调度机制比较

操作系统	RT-Linux (实时部分)	$\mu\text{C}/\text{Linux}$	$\mu\text{C}/\text{OS-}$	eCos	
				位图调度器	多级队列调度器
内核抢占	是	否	是	是	是
优先级变化	静态 (默认)、动态	动态	动态	动态	动态
调度算法	基于优先级抢占式调度; 静态采用单调速率调度 (RMS); 动态采用最早期限优先调度 (EDF)	分实时进程先来先服务调度; 普通进程时间片轮转调度	基于固定优先级抢占式调度	基于固定优先级抢占式调度	基于优先级抢占式调度; 时间片轮转调度
同优先级调度	有	有	无	无	有
优先级数量	1024 (线程)	100	64	32 (默认)	32 (默认)
任务数量	无限制	无限制	64 (用户最多可用 56)	32 (默认)	无限制
时间的可确定性	是	否	是	是	是

RT-Linux 是在标准 Linux 基础上实现了一个小的基于优先级的抢占式内核, 使用双核设计的一种硬实时操作系统。Linux 本身作为一个可抢占的任务在核内运行, 优先级最低, 随时会被高优先级任务抢占。如果用户觉得有必要, 可以自己编写调度算法来替换 RT-Linux 中的调度算法。

$\mu\text{C}/\text{Linux}$ 则在结构上继承了标准 Linux 的多任务实现方式 (分实时进程和普通进程采用不同的调度策略, 即先来先服务调度和时间片轮转调度), 仅针对中低档嵌入式 CPU 特点进行改良。当前如果需要实现比较强的实时性效果, 减少中断响应时间的唯一办法是使用 RT-Linux 或者 RTAI (<http://www.aero.polimi.it/~rtai/>) 所提供的 Linux-in-an-ERTOS 方法。

$\mu\text{C}/\text{OS-}$ 内核是针对实时系统的要求设计实现的, 只支

持基于固定优先级抢占式调度, 相对简单, 可以满足较高的实时性要求, 用法也简单。μC/OS- 通过任务就绪表中的 OSRdyGrp 和 OSRdyTbl [0.. 7] 变量以及任务调度器 OSSched(), 完成对最多 64 个不同优先级任务的调度, 而且任何时刻都不可能具有相同优先级的任务在就绪表中。

eCos 具有调度方法丰富, 当前 2.0 版本的 eCos 提供了两种基于优先级的调度器, 即位图 (Bitmap) 调度器和多级队列 (Multi-Level Queue, MLQ) 调度器, 用户在进行配置时可以根据实际需要在这两种调度器中选择一个。位图调度器的主要思想是设置若干个不同的线程优先级 (系统默认设置 32 个优先级), 任何时刻在每一个优先级只允许运行一个线程。位图调度器所允许的线程个数有严格的限制, 如果系统被设置成具有 32 个优先级, 那么系统中最多只能有 32 个线程。位图调度器具有简单、高效的特点, 但并不支持避免优先级反转的方法、对称多处理器 (SMP) 系统的支持等功能。MLQ 调度器是一种支持优先级继承和同优先级的 FIFO 调度策略。MLQ 调度器中相同优先级的线程形成一个 FIFO 队列, 这些线程采用时间片轮转策略进行调度。如果内存允许, 系统所允许的任务/线程数目是没有限制的。MLQ 调度器调度功能丰富, 但需要查找最高优先级这样的大开销操作, 执行效率较低。

由于 eCos 采用的是开放式的可配置结构, 所以今后还可以再加入别的调度方法。

3.2 任务及中断间的同步与通信机制

任务及中断间的同步与通信是 ERTOS 的重要内容之一, 通常通过若干任务和中断服务程序共同完成, 任务与任务之间、任务与中断服务程序之间必须协调动作互相配合, 这就牵涉到任务及中断间的同步与通信问题。ERTOS 通常是通过信号量 (Semaphore)、互斥型信号量 (Mutex)、事件标志 (Event Flag) 和异步信号 (Asynchronous Signal) 来实现同步, 通过消息邮箱 (Message Box)、消息队列 (Message Query)、管道 (Pipe) 和共享内存 (Shared Memory) 来提供通信服务。

ERTOS 中由于使用了互斥量等, 因此常常会面临优先级反转/倒置 (Priority Inversion) 问题。优先级反转是一种不确定的延迟形式, 任何 ERTOS 都必须处理这一问题。一般的例子是: 一个高优先级任务等待一个互斥量, 而这个互斥量正被一个低优先级任务所拥有, 如果这时这个低优先级任务被一个或多个中等优先级的任务剥夺了运行, 那么就发生了优先级反转, 即这个高优先级任务被一个不相关的较低优先级任务阻止了继续执行, 实时性难以得到保障, 如图 1 所示。

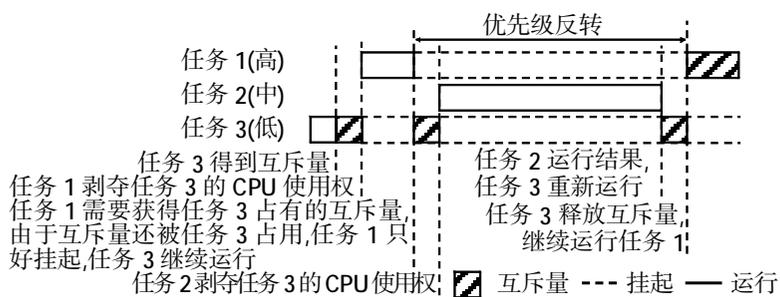


图 1 优先级倒置问题

因此, ERTOS 应尽量避免出现优先级反转, 通常可以采用优先级置顶协议 (Priority Ceiling Protocol) 和优先级继承协议 (Priority Inheritance Protocol) 这两种方法。

优先级置顶协议是指所有获得互斥量的任务把它们的优

优先级提升到一个事先规定好的值。该方法的缺点是: 需要事先知道使用这个互斥量任务的最高优先级; 如果这个事先规定的值太高, 它相当于一个全局锁禁止了所有的调度。

优先级继承协议是指拥有互斥量的任务的优先级被提升到与下一个在等待这个互斥量的最高优先级任务的优先级相等。该技术不需要预先知道准备使用这个互斥量的任务的优先级, 当一个更高优先级任务处于等待时, 只有拥有互斥量的任务的优先级被提升。该方法减少了对别的任务进行调度的影响, 但是它增加了每次同步调用的开销。内核支持优先级继承。上述优先级反转例子的过程如图 2 所示。

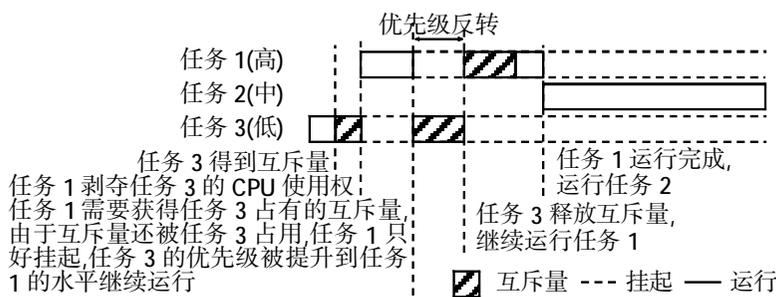


图 2 内核支持优先级继承

四种 ERTOS 的同步与通信机制的比较, 如表 3 所示。

表 3 四种 ERTOS 的同步与通信机制比较

操作系统	RT-Linux 注	μCLinux	μC/OS-	eCos	
				位图调度器	多级队列调度器
同步与通信机制	FIFO 队列、共享内存、互斥体、条件变量、信号量	信号、管道、消息队列、信号量和共享内存	信号量、互斥体、信号量、邮箱、队列、事件标志	互斥体、条件变量、信号量、邮箱和事件标志	互斥体、条件变量、信号量、邮箱和事件标志
避免优先级反转方法	基本优先级继承、优先级置顶	不支持	优先级置顶	不支持	基本优先级继承、优先级置顶

注: RT-Linux 提供三种与 Linux 进行通信的方法, 即共享内存、FIFO 设备和 MIBUFF 驱动程序。

3.3 存储器管理

存储管理也是 ERTOS 的重要内容之一, 主要包括: 内存分配原则、存储保护和内存分配方式。

(1) 内存分配原则

快速性: 系统强调对实时性的保证, 要求内存分配过程要尽可能地快, 通常都采用简单、快速的内存分配方案。

可靠性: 系统强调对可靠性要求, 也就是内存分配的请求必须得到满足。

高效性: 系统强调对高效性要求, 不仅仅是对系统成本的要求, 而且系统本身可配置的内存容量也是很有限的, 所以内存分配要尽可能地少浪费。

(2) 存储保护

在 ERTOS 中, 内存中既有系统程序, 又有许多用户程序。为使系统正常运行, 避免内存中各程序相互干扰, 通常需要对内存中的程序和数据进行保护。存储保护的内容包括: 保护系统程序区不被用户有意或无意侵犯; 不允许用户程序读写不属于自己地址空间的数据, 如系统区地址空间; 其他用户程序的地址空间。存储保护通常需要有硬件支持 (如 MMU), 并由软件配合实现。而事实上在许多 ERTOS 中, 内核和用户程序同在相同的内存空间, 也就是说没有存储保护。

(3) 内存分配方式——静态分配和动态分配

静态分配: 程序要求的内存空间是在目标模块连接装入内存时确定并分配的, 并且在程序运行过程中不允许再申请或在内存中移动, 即分配工作是在程序运行前一次性完成。

采用静态分配方案时,程序在编译时所需要的内存都已分配好,不可避免地使系统失去了灵活性,必须在设计阶段就预先考虑到所有可能的情况,根据需要对内存进行分配,一旦出现没有考虑到的情况,系统就无法处理。当然如果系统对于实时性和可靠性的要求极高(硬实时系统,如 RT-Linux),不允许一点延时或一次分配失败,则必须采用静态分配方案。

动态分配:程序要求的基本内存空间是在目标模块装入时确定并分配的,但是在程序运行过程中允许申请附加的内存空间或在内存中移动,即分配工作可以在程序运行前及运行过程中逐步完成。

采用动态分配方案时,嵌入式系统的程序设计者可以方便地将原来运行于非嵌入式操作系统的程序移植到嵌入式系统中,灵活地调整系统的功能,在系统中各个功能之间作出权衡。因此,大多数实时操作系统提供了动态内存分配接口,例如 malloc() 和 free() 函数。

四种 ERTOS 的存储器管理比较,如表 4 所示。

表 4 四种 ERTOS 的存储器管理比较

操作系统	RT-Linux	μCLinux	μC/OS-	eCos
支持 MMU	支持	不支持	不支持	不支持
管理方式	实存储器	实存储器	实存储器	实存储器
存储保护	有	无	无	无
存储机制	分页	分页	分区	不分段也不分页
分配方式	静态分配 连续区	静态或动态 页为单位 非连续	静态、动态 分配连续区	基于内存池的动态内存 分配机制,变长内存池使用 链表管理,定长内存池 使用位图管理

RT-Linux。它采用虚拟缓冲技术的页存储机制,每个实时任务以内核线程形式存在,运行在单一的核心地址空间,这样有利于存储保护。RT-Linux 实时内核不直接支持内存分配初始化,将这部分工作交给了 Linux 完成。在实时任务启动之前, Linux 为实时任务动态分配所需要的内存空间,采用 MBUFF 模式(mbuff_alloc, mbuff_free)在 RT-Linux 与 Linux 之间实现内存共享。因为 RT-Linux 需要系统具有硬实时能力,必须使用静态分配的内存来完成硬实时任务,不可以使用动态内存分配函数 malloc() 和 free()。

μCLinux。它是针对没有 MMU 的处理器设计的,不能使用处理器的虚拟内存管理技术,只能采用实存储器管理策略(Real Memory Management)。系统使用分页内存分配方式,系统在启动时把实际存储器进行分页。系统对于内存的访问是直接的,所有进程中访问的地址都是实际的物理地址。操作系统对内存空间没有保护,多个进程实际上共享一个运行空间。

μC/OS-。它是采用实存储器管理策略,分区内存分配方式,系统在启动时把实际存储器进行分区。

μC/OS- 为了消除多次动态分配与释放内存所引起的内存碎片的问题,把连续的大块内存按分区来管理,每个分区中都包含整数个大小相同的内存块,但不同分区之间内存块的大小可以不同。利用这种机制,μC/OS- 对 malloc() 和 free() 进行改造,使得它们可分配和释放固定大小的内存块,并且使这两个函数的执行时间也固定下来。用户需要动态分配内存时,选择一个适当的分区,按块来分配内存。释放内存时将该块放回它以前所属的分区。μC/OS- 也可以使用常规的 malloc() 和 free() 内存管理函数来增强可移植性,但在使用更严格的场合,应使用系统提供的特殊内存管理。

eCos。它的内存管理有点特殊,但也相对简单,既不分段也不分页,唯一复杂的是可配置部分。eCos 采用一种基于内存池的动态内存分配机制。

eCos 的内存管理通过内存池对象来实现,它提供两个类模板,即 Cyg_Mempoolt 和 Cyg_Mempolt2。每个类模板又提供两种可选的内存池。eCos 通过两种内存池类来实现两种内存管理方法:一种是变长的内存池(Variable Size Memory Pool),即内存池根据申请的内存大小进行分配;另一种是定长的内存池(Fixed Size Memory Pool),即内存池以固定大小的块为单位进行分配。变长的内存池使用链表来进行管理,定长的内存池使用位图来进行管理。默认情况下,eCos 使用变长内存方式管理内存,eCos 的 C 库函数 malloc() 就是使用变长内存池来实现内存分配的。

3.4 中断管理

中断管理是 ERTOS 内核的重要部分,主要考虑是否支持中断嵌套、中断处理机制、中断延时等内容。大多数的中断处理细节是与体系结构有关的,但 ERTOS 内核中同时会有一些通用的处理中断的机制和接口。ERTOS 一般会允许中断嵌套,也就是说在中断服务期间,ERTOS 可以识别另一个更高级别的中断,并服务于那个更高级别的中断。ERTOS 的实时性能大部分体现在系统对中断请求的响应 IRQ 和中断服务例程 ISR 的处理效率上,因此 ISR 必须尽可能地有效,不能经常或者长时间阻塞中断,是各 ERTOS 的主要目标。

(1) RT-Linux

RT-Linux 中断分成普通 Linux 中断和实时中断两类。普通 Linux 中断可无限制地使用 Linux 内核调用,作为实时中断处理的第二部分是相当不错的。RT-Linux 采用在 Linux 内核和中断控制硬件之间增加一层仿真软件的方法截取所有的硬件中断,实现了一个虚拟中断机制。Linux 本身永远不能屏蔽中断,它发出的中断屏蔽信号和打开中断信号都修改成向 RT-Linux 发送一个信号。如果 RT-Linux 内核收到的中断信号是普通 Linux 中断,那就设置一个标志位;如果是实时中断,就继续向硬件发出中断。

Linux 用禁止中断的方法作为同步机制,由于关中断和开中断的混合使用使得中断延时不可确定。但是无论 Linux 处在什么状态,都不会对实时系统的中断响应时间增加任何延迟,导致时间上的不可确定性。Linux 进程的屏蔽中断虽不能禁止实时中断的发生,却可以禁止普通 Linux 中断。Linux 不能中断自身,只能被 RT-Linux 中断;而 RT-Linux 不仅可以中断 Linux,而且可以中断自身。如果想在实时处理程序和常规 Linux 驱动程序中处理同一设备 IRQ,必须为每一个硬中断单独设置 IRQ。

(2) μCLinux

μCLinux 中断处理的核心机制,沿袭 Linux 中断处理的方法,将中断处理分为两个部分:“顶半(Top Half)处理”和“底半(Bottom Half)处理”。在“顶半处理”中,必须关中断运行,进行必要的、非常少、非常快的处理,其他的处理交由“底半处理”处理。“底半处理”中,执行那些流程复杂的、耗时的且又不十分紧迫的多数工作,并且是开中断运行,所以运行时,可以接受中断。在 Linux 系统中,因为有许多中断的“底半处理”,

这些“底半处理”形成一个任务队列,由 Linux 调度管理。在这样的中断机制下,因为“底半处理”队列的调度,所以会引起中断处理的延时。此外, Linux 中的中断可以被系统屏蔽,即使是优先级很高的硬件中断也会因为被系统屏蔽而引起中断响应的延时。

(3) $\mu\text{C}/\text{OS}$

$\mu\text{C}/\text{OS}$ 中断处理,在四种 ERTOS 中是最简单的。一个中断向量上只能挂一个中断服务子程序 ISR,而且用户代码必须都在 ISR 中完成,ISR 做的事情比较多,中断延时相对较长。系统提供两个函数 $\text{OSIntEnter}()$ 和 $\text{OSIntExit}()$ 用来进行中断管理。 $\text{OSIntEnter}()$ 通知内核即将开始 ISR,使内核可以跟踪中断嵌套,最大嵌套深度为 255。在 ISR 的末尾,使用 $\text{OSIntExit}()$ 判断中断是否已经脱离了所有的中断嵌套。如果脱离了中断嵌套,内核函数需要判断是否有更高优先级的任务进入就绪状态,如果有系统要让更高优先级的任务进入就绪状态。在这种情况下,中断要返回到更高优先级的任务,而不是被中断了的任务,因而中断恢复时间要稍长一些。

(4) eCos

eCos 使用了分层式中断处理机制,这种机制将中断处理分为两部分,即传统的 ISR 和滞后中断服务程序(Deferred Service Routine, DSR)。这种机制类似于在 Linux 中将中断处理分为顶部处理和底部处理。

这种机制可以在中断允许时运行 DSR,因此在处理较低优先级的中断时,允许别的潜在的更高优先级的中断发生和处理。为了使这种中断模型能够有效地工作,ISR 应当快速运行。如果中断引起的服务量少,ISR 可以单独处理这个中断而不需要 DSR。但是,如果中断服务比较复杂,则在 ISR 中只处理必要的工作,这种情况下,ISR 一般仅仅屏蔽中断源,通知 DSR 处理该中断,然后结束。eCos 会在稍后合适的时间执行这个 DSR,在这个时候系统已经允许进行线程调度的了。推迟到这个时候运行 DSR,可以使内核使用简单的同步方式。

如果中断源是爆发性的,在执行一个被请求的 DSR 之前,可能已经产生了多个中断并且多次调用了这个 ISR, eCos 内核会记录下被调用的 DSR 的次数。在这种情况下,这个 DSR 最终会被调用,其中一个参数告诉它有多少个 ISR 请求执行这个 DSR。

通过上述分析可知: RT-Linux 由于采用硬实时解决方案,因而中断延时最短; $\mu\text{C}/\text{Linux}$ 由于采用软实时方案,中断延时最长; $\mu\text{C}/\text{OS}$ 由于采用微内核设计,中断机理简单,中断延时相对较长; eCos 采用可配置微内核设计,中断机理复杂,中断延时相对较短。

4 结论

这四种源码开放的 ERTOS 在嵌入式系统中已有大量的应用,但它们的特点又不完全相同。通过本文第三部分的分析比较,得出不同 ERTOS 各自适用的领域:

(1) RT-Linux

RT-Linux 最大特点是在成熟的 Linux 基础上实现了一个硬实时内核,充分利用了 Linux 的系统服务,因此适合于需要比较复杂的系统服务,对时间有严格要求的应用领域,航空航

天、科学研究、机器人技术以及工业控制和测量。

(2) $\mu\text{C}/\text{Linux}$

$\mu\text{C}/\text{Linux}$ 最大特点在于针对无 MMU 处理器设计,可以利用完全免费且功能强大的 Linux 资源,因此适合开发对时间要求不高的小容量、低成本的产品,特别适用于开发与网络应用密切相关的嵌入式设备。

(3) $\mu\text{C}/\text{OS}$

$\mu\text{C}/\text{OS}$ 是一个非常容易学习、结构简单、功能完备和实时性很强的嵌入式操作系统内核,适合于广大的嵌入式系统开发人员和嵌入式 ERTOS 爱好者的入门学习,以及大专院校教学和科研。 $\mu\text{C}/\text{OS}$ 很适合开发那些对系统要求不是很苛刻的,而 RAM 和 ROM 有限的,各种小型嵌入式系统设备。

(4) eCos

eCos 最大特点是其配置灵活,而且是面向深度嵌入式应用的,很适合于用在一些商业级或工业级对成本敏感的嵌入式系统,如一些消费电子、汽车制造、工业控制等,比较知名的应用有 Brother HL-2400 CeN 网络彩色激光打印机、Delphi Communiport 车载信息处理系统(MPU) 车载信息处理系统(Mobile Productivity Center, MPC)、Iomega Hip Zip 数字音频播放器、Ik-endi 指纹识别系统等。

参考文献:

- [1] 李飞. 实时操作系统的比较[J]. 电子世界, 2003, (10): 26-27.
- [2] 何小庆. 选择 ARM CPU 的操作系统 $\mu\text{C}/\text{OS}$, $\mu\text{C}/\text{Linux}$, 还是 Linux[J]. 电子产品世界, 2004, (2): 38-40.
- [3] 李垣陵. $\mu\text{C}/\text{OS}$ 和 $\mu\text{C}/\text{Linux}$ 比较[J]. 电子产品世界, 2002, (10): 18-20.
- [4] FSMLabs 公司. RT-Linux Articles and Documentation[EB/OL]. <http://www.fsmlabs.com/articles/articles.html>, 2004-08-01.
- [5] 赵慧斌, 李小群, 孙玉芳. 改善 Linux 核心可抢占性方法的研究与实现[J]. 计算机学报, 2004, 27(2): 244-251.
- [6] Ismael Ripoll. RTLinux Versus RTAI[EB/OL]. http://bernia.disca.upv.es/rtportal/comparative/rtl_vs_rtai.html, 2002-10.
- [7] Herman Bruyninckx. Real-time and Embedded Guide[M/OL]. <http://people.mech.kuleuven.ac.be/~bruyinck/rthowto/rtHOWTO.pdf>, 2002-12-11.
- [8] 毛德操, 胡希明. Linux 内核源代码情景分析(上册)[M]. 杭州: 浙江大学出版社, 2001.
- [9] USB 开发网. 嵌入式操作系统 $\mu\text{C}/\text{Linux}$ [EB/OL]. http://www.usbing.net/Article_Show.asp?ArticleID=114, 2004-8.
- [10] 袁爱君. 基于 ARM 的嵌入式 $\mu\text{C}/\text{Linux}$ 系统设计与 Web 服务器应用的实现[D]. 杭州: 浙江大学硕士学位论文, 2003.
- [11] 万晨妍. 基于 ARM 的嵌入式系统及 SNMP 的设计与实现[D]. 杭州: 浙江大学硕士学位论文, 2003.
- [12] Jean J. $\mu\text{C}/\text{OS}$ 源码公开的实时嵌入式操作系统(第 2 版)[M]. Labrosse. 邵贝贝, 等. 北京: 北京航空航天大学出版社, 2003.
- [13] 蒋句平. 嵌入式可配置实时操作系统 eCos 开发与应用[M]. 北京: 机械工业出版社, 2004.
- [14] Anthony J Massa. Embedded Software Development with eCos[M]. New Jersey: Prentice Hall PTR, 2002.
- [15] 秦怀峰. 面向嵌入式 PC 的 eCos 支持与完善[D]. 西安: 西北工业大学硕士学位论文, 2002.

作者简介:

季志均(1977-), 男, 江苏泰兴人, 硕士研究生, 研究方向为嵌入式操作系统、生物医学信号的检测和处理; 马文丽, 女, 教授, 研究方向为生物医学基因诊断; 陈虎, 男, 讲师, 研究方向为嵌入式操作系统; 郑文岭, 男, 教授, 研究方向为生物医学制药技术。