嵌入式操作系统驱动程序开发

张晓辉*
(西南交通大学电气工程学院, 四川 成都 610031)

[摘 要] 嵌入式系统的应用越来越广、越来越复杂, 嵌入式操作系统在嵌入式系统中的作用也日显重要。嵌入式操作系统设备驱动程序的开发也是嵌入式系统开发中的一个重要的方面。本文通过对现在比较流行的四种嵌入式操作系统μCOS- II、嵌入式 Linux、VxWorks 和Windows CE 的简单介绍与比较, 讨论并给出了这几种嵌入式操作系统驱动程序的开发方法。并给出了开发标准 RS- 232 串行通信接口驱动程序的实例。

[关键词] 嵌入式操作系统; 驱动程序; RS- 232
[中图分类号] TP316 [文献标识码] A [文章编号] 1672- 9706(2005)01- 0074- 04

Device Driver Development on Embedded Operation System

ZHANG Xiao- hui
(Electrical Engineering of Southwest Jiaotong University, Chengdu 610031, China)

[Abstract] Action that embedded operation system is living in the Embedded System more and more important with the application of Embedded System is more and more vast and complex. The development of device driver on embedded operation system is also the significant respect in the development of the embedded system. The paper introduces and compares four embedded operation systems: μCOS- II, embedded Linux, VxWorks and Windows CE. The important content is the device driver development on these embedded operation system. In this paper the author develops the device about the RS- 232 serial communication interfaces.

[Key words] embedded operation system; device driver; RS- 232

嵌入式系统的应用越来越广、越来越复杂, 嵌入式操作系统在嵌入式系统中的作用也日显重要。嵌入式操作系统的主要任务之一是控制所有的输入/输出设备, 它必须向设备发布命令, 捕获中断并进行错误处理, 它还提供一个设备与系统其余部分的简单易用的界面, 该界面应该对所有的设备尽可能的一致, 从而将系统硬件设备细节从用户视线中隐藏起来。具体细节都是由设备驱动程序来完成的。

设备驱动程序完成的功能主要有: ①设备的初始化和设备的释放; ②完成内核与设备之间数据(或命令)的传递; ③完成应用程序与设备之间数据的传递; ④检测、处理设备出现的错误。

在开发嵌入式系统设备驱动程序的时候, 一般遵循如下的步骤进行编写:

(1) 读懂硬件规格书, 并且能够存取每一个装置的管脚。

(2) 向系统注册中断的服务函数, 让操纵系统了解中断产生的时候系统要用哪些函数。

(3) 将驱动程序界面加到一般用户存取界面函数中, 让用户通过一定的函数调用来存取外围装置。

其中(2)、(3)必须依照操作系统本身的规定来做, (1)依据该硬件平台的硬件说明书来做。下面就几种比较常用的嵌入式操作系统进行介绍、比较并给出其驱动程序的开发。

* 收稿日期: 2004- 11- 17
作者简介: 张晓辉(1979-), 女, 山东潍坊人, 西南交通大学电气工程学院硕士研究生。

1 μCOS- II

μCOS- II 是一个完全免费的实时操作系统。其源码公开化; 内核可移植性强、可固化、可裁减; 且为占先式实时内核; 多任务且每个任务都有自己的任务栈; 提供很多系统服务(例如邮箱、消息队列、信号量、块大小固定的内存申请与释放、时间相关函数等)。μCOS- II 从 1992 年至今已用于几百个商业运用, 稳定和可靠性强。

对μCOS- II 这种微内核操作系统, 其设备驱动一般都是由应用程序在内核外部实现的。不过由于不分系统空间和用户空间, 这些设备驱动程序仍在系统态执行。从实现方式上看, 设备驱动程序的实现可以分为两种: 一种是轮询(Polling) 方式, 另一种是中断方式。

(1) 轮询式设备驱动

最简单的设备驱动方式是轮询。在这种方式中, 对设备的操作完全由 CPU 掌握, 外部设备则完全处于被动的状态。在采用轮询方式的设备驱动中, 当 CPU 要对某一外部设备进行某项操作, 但是因为该设备尚未准备好而暂时不能进行时, CPU 有两种对策: 一种是睡眠一段时间(或先进行其他操作) 以后再试; 另一种是锲而不舍地试探下去, 直至完成, 称为“ 连轴转(Spin)”。显而易见, 在实时系统中, 如果采用轮询方式的设备驱动, 势必会影响系统的实时性能。虽然不能排除轮询在嵌入式系统中的应用, 但是它毕竟不可能成为主要的设备驱动方式。

(2) 中断式设备驱动

毫无疑问, 中断是嵌入式系统主要的设备驱动方式。

μCOS- II 提供了两个 ISR 与内核的接口函数: OSIntEnter 和 OSIntExit。OSIntEnter 通知μCOS- II 内核, 中断服务程序开始了。而另一个接口函数 OSIntExit 则通知内核中断服务已结束。然后根据相应情况, 返回被中断点或由内核做任务调度。用户编写的 ISR 必须被安装到某一位置, 以便中断发生后, CPU 根据相应的中断号运行准确的服务程序。

在μCOS- II 下开发驱动程序要自由得多, 因为它没有用户态、核心态等特权级别的区分, 所有的软件都连接在一起, 成为一个二进制映像。或者说, 应用软件、操作系统以及设备驱动之间没有什么明显的区别, 这些界线的划分纯粹是软件开发者的责任, 每个人都可以根据他的思路习惯和具体的应用情况对整个系统作出合理, 但不尽相同的结构划分。

在μCOS- II 上编写设备驱动程序, 可以归结为下面 5 个要素: (1) 把访问设备需要的相关数据信息封装成合适的数据结构, 便于管理。(2) 定义一组跟踪目前硬件和设备驱动状态的变量, 这些变量的定义是与第一点对应的。(3) 编写设备初始化函数。(4) 为用户端的有关操作编写接口函数, 仔细合理地定义接口函数的形式、参数和返回值, 最主要的用户程序操作就是读和写。(5) 编写中断服务程序, 这是最困难的工作。在决定 ISR 具体做些什么之前, 需要对整个设备驱动方案有个全局的把握和深入的分析。

RS- 232 串行通信接口设备驱动程序实例:

因为串行设备或者其他字符型设备都存在外设处理速度和 CPU 速度不匹配的问题, 需要一个缓冲。这里用环形队列数据结构来描述缓冲区, 用 UART 中的中断来控制 ISR, 只对缓冲的一端进行读写, 而用户任务则在缓冲的另一端作读写操作。引入读、写信号量分别对缓冲两端的操作进行同步。这里主要给出串口接收与发送程序的设计。

(1) 带缓冲和信号量的串口接收程序: 接收到中断到来后, ISR 从 UART 控制器 FIFO 中读入接收的字节, 放入接收缓冲, 然后通过接收信号量唤醒用户任务端的读操作。

(2) 带缓冲和信号量的串口发送程序: 接收中断是由外界被动触发的, 而发送中断在 UART 的发送寄存器为空时产生, 所以这里用计数型信号量。发送信号量初始值为发送缓冲的大小, 并且关闭发送中断。发送数据时, 用户任务在信号量上等待。如果发送缓冲未满, 用户任务向发送缓冲写入数据。如果

写入的发送缓冲中取出最早写入的字节输出至 UART。这个操作又触发了下一次的发送中断, 如此循环确定发送缓冲中最后一个字节被取走, 重新关闭发送中断。在 ISR 向 UART 输出的同时, 给信号量发信号, 发送任务据此信号量计数值来了解发送缓冲中是否有空间。

除了上述函数外, 还需要编写一些其他的接口函数: 与 CPU 架构或运行模式有关的接口函数; 对 UART 操作的接口函数; 与操作系统相关的读写及其他操作接口函数。

2　嵌入式 Linux

嵌入式 Linux (Embedded Linux) 是指对标准 Linux 经过小型化裁剪处理之后, 能够固化在容量只有几 kB 或者几 MB 的存储器芯片或者单片机中, 适合于特定嵌入式应用场合的专用 Linux 操作系统。

作为一个宏内核操作系统, Linux 的设备驱动都是在内核中, 即系统空间实现的。实现的方式有两种: 一种是将有关的驱动程序和数据结构静态地连接在内核映像中; 另一种是将具体设备的驱动程序和数据结构独立加以编译, 成为“可安装模块”, 需要时才由应用程序通过系统调用动态地予以安装或拆卸。这当然与采用微内核的操作系统形成鲜明的对照。由于所有设备驱动都实现于内核中, 并且只能实现于内核中, 对设备的操作就最终必须进入内核才能完成。而系统调用又是进程主动进入内核的唯一途径, 于是对设备的操作就只有通过系统调用才能完成。

在嵌入式 Linux 操作系统下主要有 3 类设备: 字符设备、块设备和网络设备。在 Linux 系统下将硬件设备当作文件看待。

嵌入式 Linux 的设备驱动程序与外接的接口与 DDI/DKI 规范相似, 可以分为三个部分: (1) 驱动程序与内核的接口, 这是通过数据结构 file_operations 来完成的。(2) 驱动程序与系统引导的接口, 这部分利用驱动程序对设备进行初始化。(3) 驱动程序与设备的接口, 这部分描述了驱动程序如何与设备进行交互, 这与具体设备密切相关。

根据功能驱动程序的代码可以分为如下几个部分: (1) 驱动程序的注册和注销; (2) 设备的打开与释放; (3) 设备的读和写操作; (4) 设备的控制操作; (5) 设备的中断和查询处理。

RS- 232 串行通信接口(UART) 属于字符设备, 下面给出其驱动程序:

1) 头文件、宏定义和全局变量: 一个典型的设备驱动程序一般都包含有一个专用头文件, 这个头文件中包含一些系统函数的声明、设备寄存器的地址、寄存器状态位和控制位的定义以及用于此设备驱动程序的全局变量的定义。

2) OPEN() 函数: 无论一个进程何时试图去打开这个设备都会调用这个函数。

3) RELEASE() 函数: 当一个进程试图关闭这个设备特殊文件的时候调用这个函数。

4) READ() 函数: 当一个进程已经打开此设备文件以后并且试图去读它的时候调用这个函数。这里从 UART 的控制器中读取数据并发送至缓冲区, 进程从缓冲区中读取数据。由于缓冲区不是内核空间而是用户空间, 所以必须将内核空间的数据拷贝到用户空间。

5) WRITE() 函数: 当试图将数据写入这个设备文件的时候, 这个函数被调用。这里先将数据发送至缓冲区, 进而写入设备。

6) 这个设备驱动程序提供给文件系统的接口。

7) 模块的初始化和模块的卸载。

3　VxWorks

VxWorks 是美国 Wind River System 公司推出的运行在目标机上的高性能、可裁减的嵌入式实时操作系统。它以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中。它的特点有:

(1) 它具备一套实时操作系统开发环境—Tornado, 该环境与操作系统紧密结合。

(2) 它的开放式结构和对工业标准的支持使开发者只需做最少的工作即可设计有效的适合于不同

用户要求的实时操作系统。

(3) 支持 POSIX 1003.1 b 用户接口实时扩展标准。

(4) 微内核结构(最小结构< 8 kB), 使得整个系统易于定制。

(5) 网络支持上满足 TCP/IP 标准, 可以和 Internet 连接, 符合发展潮流。

在 VxWorks 中, 设备驱动程序也有查询和中断两种方式, 无论采用哪种方式, 设备驱动程序的基本流程都是相同的, 如图 1 所示。

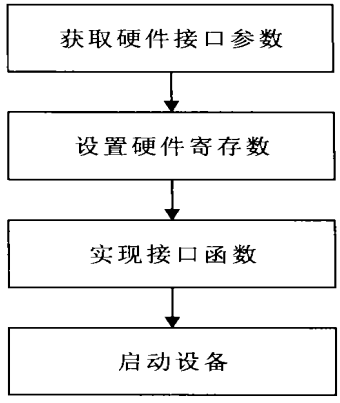

图1 驱动程序的基本流程

获取硬件接口参数包括内存映射地址、I/O 端口和系统中断控制器的输入(IRQ)。获取这些参数由硬件的接口方式决定, 硬件的行为和特性由内部寄存器控制。对于采用中断方式的硬件设备, 在接口函数中必须实现中断服务程序。中断程序的编写必须遵循三条规则: 代码的运行时间不能太长, 并尽量保持能被其他中断程序中断的能力; 不能独占共享资源以避免死锁; 程序结束后尽可能快速返回。

4 Windows CE

Windows CE 是一个简洁的、高效率的多平台操作系统。它是从整体上为有限资源的平台设计的多线程, 抢占式多任务的操作系统, 并有强大的通信能力。它的模块化设计允许它对于从掌上电脑到专用的工业控制器等电子设备进行定制。但是它的基本核心需要至少 200k 的 ROM, 而且在实时性方面的表现也不尽如人意。

但是它支持 Win32API 的一个子集, 为开发人员提供熟悉开发平台, 使得有经验的 Windows 开发人员可以很快的掌握 Windows CE 的开发。

从 Windows CE 驱动程序的复杂性角度, 设备驱动可以分两种: 分层的驱动和单体结构的驱动, 绝大多数实际的 Windows CE 设备驱动程序都是分层的驱动。

小结

设备驱动程序所完成的功能主要是设备的初始化, 操作系统内核与设备之间数据(或命令)的传递等, 但是在不同的嵌入式操作系统中, 其设备驱动程序的开发还是有差异的。所以在进行嵌入式操作系统设备驱动程序开发的过程中, 应该根据不同嵌入式操作系统, 如上所述编写相应的设备驱动程序。另外, 在编写驱动程序时候必须注意下面的事项:

(1) 驱动程序所占用的内存空间不能太大; (2) 驱动程序架构所需的计算资源要低; (3) 驱动程序对其他模块的反应要及时; (4) 中断程序的执行时间要足够短; (5) 驱动程序内对类似设计的移植性要高; (6) 驱动接口与硬件规格相关性低; (7) 驱动程序内各模块的相关性低及处理要具备独立性; (8) 与浮点运算无关; (9) 减少状态变量; (10) 减少状态常数所要表达的意义; (11) 基底地址常数声明的建立; (12) 遮照常数声明的建立; (13) 状态常数声明的建立; (14) 常用功能宏的建立。

参考文献:

[1] Jean J Labrosse. 嵌入式实时操作系统 μCOS- II[M]. 北京: 北京航空航天出版社, 2003.

[2] 陈莉君. 深入分析 Linux 内核源代码[M]. 北京: 人民邮电出版社, 2002.

[3] 孔祥营. 嵌入式实时操作系统 VxWorks 及其开发环境 Tornado[M]. 北京: 中国电力出版社, 2001.

[责任编辑: 程 蓓]