第33卷 第21期 计 算 机 工 程 2007年11月
Vol.33 No.21 Computer Engineering November 2007
·开发研究与设计技术· 文章编号：1000—3428(2007)21—0273—03 文献标识码：A 中图分类号：TP311

基于嵌入式实时操作系统的驱动框架

李 健[1,3]，李明禄[2]，张 激[3]，施 刚[3]
（1. 上海交通大学软件学院，上海 200030；2. 上海交通大学计算机系，上海 200030；3. 华东计算技术研究所，上海 200233）

摘 要：嵌入式设备随着应用领域的扩展呈现出多样性以及复杂性，而嵌入式实时系统普遍不具有良好的设备驱动体系结构，造成驱动开发困难、可移植性差。该文给出了一种嵌入式实时操作系统下的驱动框架，支持各种设备及总线接口，设计了一种快速的设备中断机制，符合嵌入式实时系统的强实时、高可靠的特点，能够满足开发各种嵌入式系统。

关键词：实时操作系统；驱动框架；中断处理

Device Driver Framework Based on Embedded Real-time Operating System

LI Jian[1,3], LI Ming-lu[2], ZHANG Ji[3], SHI Gang[3]
(1. School of Software, Shanghai Jiaotong University, Shanghai 200030; 2. Department of Computer Science, Shanghai Jiaotong University, Shanghai 200030; 3. East-China Institute of Computer Technology, Shanghai 200030)

【Abstract】The widespread usages of embedded applications have led to a new level of functional variety and complexity in embedded hardware. Because of the lack of a common framework for developing device drivers, implementing or porting drivers for real-time embedded systems is a difficult task for developers. This paper presents a common framework for driver development based on real-time operating systems, various devices and bus interfaces will be supported with ease under this scheme. A technique of implementing a high speed interrupt system required to meet the hard timing constraint and high reliability of real-time systems is also explained.

【Key words】real-time embedded operating system(RTOS); driver framework; interrupt processing

嵌入式设备呈现出多样性及复杂性，需要支持网络接口卡、显示卡(LCD)、串行设备(串口)、存储卡(SD, FLASH)等设备类型，涉及到的设备挂接总线包括：IEEE1394, USB, 1553, CPCI等，并且还必须支持CAN, USB, IrDA, Bluetooth等通信接口，伴随着的是其设备驱动的开发日趋复杂。

同时，嵌入式实时系统具有强实时性的基本要求，反映强实时系统的最重要指标是设备中断响应时间，即中断延迟时间。为方便开发各种设备驱动、缩短设备中断延迟时间、保证系统核心的健壮性，嵌入式实时操作系统需要提供一种基于安全关键的设备驱动框架，既能够支持各种设备的接口，又能够监测、隔离、恢复外部设备引发的故障，并且能够确保各分系统之间的互连互通、音视频等信息的采集、信息的存储及快速访问。

本文设计了一种基于嵌入式实时操作系统的设备驱动框架，通过屏蔽底层硬件处理器、规范化总线接口、抽象设备访问接口等技术手段，支持接入、访问各种设备；利用快速的中断处理机制，支持设备操作的快速响应。

1 设备驱动的基本框架

通过研究I/O系统的基本抽象接口，结合设备驱动表、设备表、文件描述符表、总线驱动支持、设备中断处理机制，可以得出设备驱动的基本框架，如图1所示。在该体系结构中，主要包括：

(1)I/O标准接口。为确保应用的可移植性，I/O接口应符合POSIX标准，即create, open, close, read, write, ioctl, select等；网络接口应符合Berkeley socket标准，即socket, bind, listen, accept, send, recv等。

(2)文件描述符管理。系统中每个任务动态维护一张文件描述符表fd[]，文件描述符是I/O标准接口与驱动表之间的桥梁，创建设备文件时从表中取出一个空闲项，同时映射相应的设备表、驱动表，随后对设备的访问控制都可以通过该文件描述符来进行。通过文件描述符的统一管理，可以方便地支持I/O系统的输入输出重定向、多路输出等功能，提高了设备管理的动态灵活性。

(3)设备表、驱动表。各种I/O设备在访问控制、数据传输方面千差万别，设备驱动框架通过设备表、驱动表及其对应关系实现了独立于具体设备的统一设备访问机制。通过将每个设备映射到文件，对设备的访问控制就可以集中到与文件相关的的数据操作，相对应的即是驱动表项中的抽象接口(create, open, close, read, write, ioctl, select)，另外，驱动表中还设置主设备号、次设备号，主设备号区分不同种类的设备，次设备号区分相同种类的不同设备，即同一种设备通过定义相同的主设备号可以共享对应的设备驱动程序。同时设备驱动框架还支持设备的可动态配置性，设备可通过设备表中的表项实现动态添加、删除。

(4)文件系统管理。传统的设备分为字符设备和块设备，

作者简介：李 健(1978－)，男，工程师，主研方向：系统软件，实时系统实现及应用；李明禄，博士、教授、博士生导师；张 激，研究员；施 刚，高级工程师

收稿日期：2006-12-26 **E-mail：**lijian@ecict.com.cn

块设备中的数据管理涉及到大量数据的获取、存储、修改、传输等，一般需要借助于文件系统，文件系统介于 I/O 应用接口与具体设备驱动之间，对相应设备的操作是通过驱动表先调用文件系统服务，再调用底层的设备驱动例程来完成的。

(5)通信协议层。其处理过程和文件系统类似，都是通过驱动表先调用相应的协议服务，再结合协议服务调用底层的设备驱动例程，通过将协议服务从驱动中分离开来，可以较好地在同一种协议上支持多种设备(典型的是网络协议栈、终端结构中的行规则协议)，从而有效地减轻了驱动开发的难度并提高了可移植性。

(6)协议适配层。实现通信协议抽象层，将某种协议独立于实际传输的介质，同时数据交换也不考虑硬件平台的类型，由抽象的数据交换处理接口屏蔽硬件的差异，典型应用是在嵌入式中间件应用系统中支持基于 CAN 总线、1553 总线、CPCI 总线、串口等总线传输的 IP 协议封装，协议适配层的访问也和文件系统管理层类似，相当于 I/O 标准接口调用底层硬件硬件驱动例程的钩子(hook)服务。

(7)底层驱动层。包括总线驱动、设备驱动层，设备一般挂接在不同的总线之上，总线驱动往往需提供其特有的总线访问控制接口，如 PCI 总线驱动中的接口 pci_find_by_devid，可通过设备 vendorId 获取挂接在 PCI 总线上的设备状态信息，进一步完成设备的初始化。

(8)实时操作系统支持接口。包括设备的中断管理机制，提供设备中断挂接、中断优先级排序等功能，是访问控制设备必不可少的环节；操作系统还提供中断处理例程与任务、访问设备的不同任务之间的同步、共享机制，便于数据传输以及互斥访问控制等。

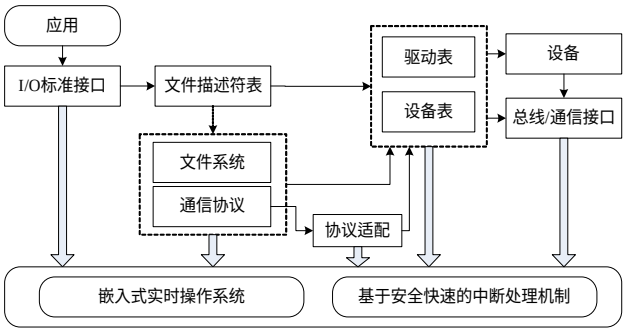

图 1　设备驱动框架结构

2　层次化的设备驱动接口

层次化的设备驱动接口见图 2。

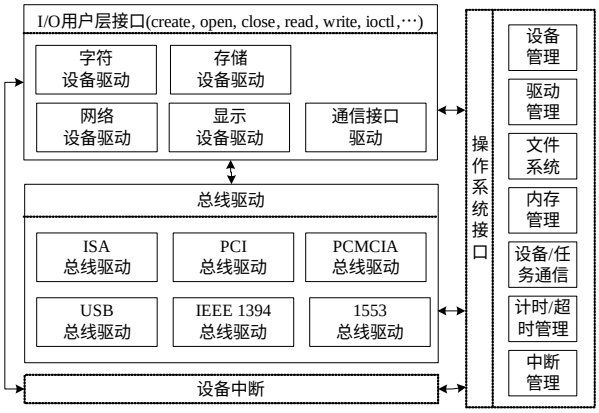

图 2　层次化的设备驱动接口

本节主要是基于上述基本框架的一个 I/O 系统的基本实现，提炼出设备开发过程中涉及到的各种接口。基于可移植、模块化的设计思想，设备驱动接口可分为 4 个层次：底层基于快速的设备中断响应处理层，总线/设备驱动层，操作系统接口层，上层用户接口层。如图 2 所示。

2.1　与设备相关的操作系统接口层、用户接口层

包括设备管理、驱动管理、文件系统、设备处理/任务间通信机制、计时/超时管理、中断管理、Cache 与 DMA 管理等，总体上讲，除了通用的设备加载/卸载，驱动加载/卸载，即基本的 I/O 操作外，同时支持嵌入式实时系统中开发驱动的一些特用方法或接口。

2.1.1　设备驱动的加载方式

通常，嵌入式实时操作系统中的设备驱动加载方式为直接加载，即在系统初始化时调用设备驱动的初始化接口。本文针对某些嵌入式系统容量小以及某些设备总线的特性，通过系统服务支持运行时进行设备驱动的加/卸载，主要有 2 种应用场景：

(1)设备的热插拔

比如基于 PCMCIA 总线的 SD 卡、网卡等，当对应设备进行热插拔时，通过中断或轮询，设备加载服务得到感知，再进行相应设备驱动的加/卸载，同时还支持插入设备操作相关的后置处理(卸载时设备相关的文件描述符关闭等)。典型应用是地理信息导航系统中基于 SD 卡实现地图数据的动态切换，通过驱动的动态加载服务，可以确保在不关机的情况下进行。

(2)驱动初始化的“即用即插”

某些嵌入式系统的容量比较小，特别是内存资源，设备驱动的加载必然会引起内存、中断号等资源的消耗，通过设备加载服务，支持在该设备进行打开的同时进行加载(以及在设备关闭的同时进行驱动卸载)，降低了内存等资源的消耗。另外，某些嵌入式系统中，多种设备的初始化有加载次序的要求，通过设备加载服务可以将驱动的加载进行延后等待处理，有效地提高了系统设计的灵活性。

2.1.2　设备驱动中的内存分配方式

设备中断处理中一般是不允许调用任何会引起阻塞的系统调用，同时对于内存的访问除了 CPU/Cache 进行访问外，DMA 也会在同一时刻进行访问，其内存分配/释放接口与通用的内存接口有不同之处：

(1)DMA 访问的内存地址必须是物理地址，内存分配时必须是“超级模式”，即进行物理内存的直接分配，系统支持一段物理内存的分配/释放管理。同时，还满足 DMA 的访问地址边界对齐、最大大小等要求，分配时能够进行自动调整。

(2)由于 CPU Cache 以及 DMA 的同时存在，Cache Invalidate/WriteThrough 也是开发驱动常调用的函数，在向设备传递数据时调用 Cache WriteThrough(更新物理内存中的数据，与 Cache 同步)，读取设备数据时调用 Cache Invalidate 接口(DMA 将数据直接传递到物理内存中，需同步 Cache)。

2.1.3　设备驱动中的计时支持

(1)超时处理。一般有以下应用场景：

1)某些设备发生了故障，需要进行设备的重新初始化，典型的是网络接口卡需要使用一种检测方法，在长时间得不到响应的情况下进行自动重启。

2)某些设备在进行读写数据时需要设置超时处理，即特

定的操作需要在一段时间内完成，否则进行重新数据的重新读写。

(2)忙等待。许多设备芯片中一般包括发送/接受缓冲区，在发送时，需要等待缓冲区中的数据全部发送出去，而在中断处理中又不允许调用会引起阻塞的系统调用，这种情况下就需要进行一段时间的忙等待操作。

2.1.4 设备驱动的健壮性支持

系统与外部环境主要通过各种总线/通信接口、存储设备等相联系，在外部环境多样性的前提下，设备驱动的不健壮会引起系统无法预料的结果，甚至崩溃。

从本质上来讲，健壮的设备驱动必需以完善的设备体系结构、健壮的操作系统、健壮的总线驱动为基础，在设备驱动的各个环节进行故障的检测、隔离及恢复，相关的几个重要原则如下：

(1)驱动应该相对独立，相互间的联系尽量减少。

(2)对于 I/O 的操作通过规范化的操作接口来进行。

(3)数据的读写必须检测其完整性及有效性。

(4)对于可能发生的各种故障能够提前预测并加以控制，包括各种总线驱动。

(5)驱动卸载的同时必须释放所有占用的资源。

(6)启用系统的日志功能，记录设备驱动的运行状况。

(7)通过轮询定时器，支持设备的健康性检测。

2.1.5 设备驱动中的其他系统支持接口

(1)I/O 端口读写接口。提供 8/16/32/64 位的 I/O 读写操作，一般 I/O 空间有两种访问方式，X86 中的 I/O 空间与内存空间是两种映射方式，需要特殊指令进行访问，ARM 中 I/O 空间贡献内存空间，直接访问内存即可，系统提供统一的抽象 I/O 访问接口。

(2)中断与任务的同步通信机制。设备驱动中使用的比较多的是事件管理，中断得到响应时通过发送事件通知对应的任务进行数据读写。

(3)数据环。由于中断处理和数据处理任务需要访问同一段内存空间，系统应该提供统一的内存环操作接口进行数据的读写。

(4)中断屏蔽字的操作。简单的打开/关闭设备中断方式。

2.2 总线/设备驱动层

该层主要是将目前嵌入式系统中主流的总线、设备进行分类，在基本框架的基础上进行分类别的接口抽象，总线主要分为 ISA 总线、CPCI 总线、PCMCIA 总线、USB 总线、1553 总线等，设备主要分为字符设备(UART, SPI 等)，存储设备(软驱、硬盘、DOC 等)，网络设备，显示设备(LCD 等)，通信接口(红外、蓝牙等)。

2.3 快速的中断处理机制

系统对设备的及时响应最好的办法就是对设备发出的中断信号的响应，再由中断处理函数进行设备的处理，操作系统中，设备中断处理中一般支持信号量、消息队列、事件、信号等线程间的通信、同步机制的调用。由于中断的异步突发性，需要对相关的临界区进行访问上锁，临界区保护强有力的方法是禁止中断，这种上锁保证了对 CPU 的独占访问，但同时，影响了设备中断的响应。

如何缩短设备中断的延迟时间，一直是嵌入式实时系统研究的内容之一，一般实时系统中断处理机制实现的基本原则有：中断处理程序尽量短，通过中断的后置任务处理大量的数据通信，不支持浮点操作等 CPU 执行周期长的指令，尽量不锁中断等。除了上述基本原则，本文提出了一种缩短中断延迟时间的中断处理机制：

(1)中断中进行直接的任务调度，系统中的上下文保护主要有两个地方：任务切换时的上下文保护以及中断被触发时的上下文保护，如将两种上下文保护方式进行统一，并且在任务上下文恢复时模拟中断中进行恢复的环境，就可以实现在中断中直接进行任务的调度。

(2)中断优先级的引入，在处理某中断时，屏蔽优先级与之相同或者低的中断，同时允许优先级更高的中断进入，即中断嵌套，从而可以确保最高优先级中断的顺利优先执行。

(3)中断与任务进行通信的相关模块的优化。

传统的实时系统主要有 2 种处理方式：

1)在通信机制中设置大量的临界区；

2)提供一种微内核状态，在重入微内核状态时将所要处理的接口打包成 Job，再在必要的时刻执行。

第 1)种方法不能有效降低中断的延迟时间。第 2)种方法会引起系统内存、CPU 等资源相对于任务、中断的归属问题。

通过在中断处理中重入微内核状态时，将中断处理函数升级为轻量级的任务，并纳入调度，如此既可以大量减少系统的临界区，又不存在 CPU 等资源的归属问题。

另外将中断处理函数映射到任务的同时可能会引起中断任务的优先级逆转问题，通过基于确保最高优先级中断的原则来进行实现，可以有效地避免。

3 结束语

本文设计了一种基于嵌入式实时系统的设备驱动框架，该框架容纳了目前主流的总线及设备，支持集成传统 I/O 系统的文件系统、网络协议栈等模块，提出了协议适配的概念，并且通过实现快速的中断处理机制来降低中断延迟时间，同时提炼出与设备驱动开发相关的一些通用接口，包括各种总线/设备接口、操作系统提供的支持接口，从而使用户能够快速地开发设备驱动。另外，该框架适用于不同的硬件体系结构，确保了系统的移植性。

参考文献

1 Kleidermacher D. Minimizing Interrupt Response Time[EB/OL]. (2006-01-30). http://www.ghs.com/download/articles/GHS_Minimixe_Interrupt_031405.pdf.

2 Wind River Systems Inc. VxWorks Programmer's Guide 5.4[Z]. 1997.

3 周正干, 李和平, 安振刚. 设备驱动程序通知应用程序的几种方法[J]. 单片机与嵌入式系统应用, 2003, (11): 16-17.

4 杜永文, 何华灿, 陈 榕. 基于灵活内核的构件化驱动程序[J]. 小型微型计算机系统, 2004, 25(4): 587-590.