第 32 卷 第 14 期 计 算 机 工 程 2006 年 7 月
Vol.32 № 14 Computer Engineering July 2006
·开发研究与设计技术· 文章编号：1000—3428(2006)14—0271—03 文献标识码：A 中图分类号：TP 391

高度可移植嵌入式系统设备驱动体系结构

杨永志，唐玉华
(国防科学技术大学计算机学院，长沙 410073)

摘 要：设备驱动的编写是嵌入式系统软件设计的重要部分。当前的驱动开发通常是根据硬件设备，寻找其驱动然后进行移植工作。如果没有可用的驱动程序，将被迫重新编写设备驱动。这种模式不利于软件的重用，往往造成重复劳动。该文在现有驱动开发模型的基础上，提出了一个高度可移植的设备驱动编写模型，并定义出相应的驱动程序开发接口。这样，驱动程序的移植就被集中在这些接口的实现和适配上，从而大大提高了驱动程序的可移植性。

关键词：嵌入式系统；驱动程序；可移植性；接口

Architecture of Embedded Device Driver for High Portability

YANG Yongzhi, TANG Yuhua
(School of Computer, National University of Defense Technology, Changsha 410073)

【Abstract】 Implementation of device driver is an important part of embedded system development. Current way is to find some existing code according to the specified device, and analyze it thoroughly before porting. If no code is found, it has to write the code. It is time consuming and some kind of wasting. This article suggests a model of designing highly portable device driver and defines its interfaces, based on current models. So the porting of device driver is limited to implementation and adapting of those interfaces. The portability is built.

【Key words】 Embedded system; Device driver; Portability; Interface

1 驱动程序设计的抽象模型

嵌入式操作系统、用户程序和设备驱动是嵌入式系统设备驱动编写中的 3 大基本模块。用户调用驱动程序的服务与硬件进行数据交换；设备驱动访问硬件并在操作系统提供的一些基本服务辅助下完成用户指定的操作。因此，可以建立两个框架模型。

1.1 直接模型

图 1(a)中，设备驱动和用户程序、操作系统以及硬件设备直接交互。很明显，它只用于驱动的早期开发，并不适合用于构建可移植的驱动。相对于下面的间接模型，将该模型称为直接模型。

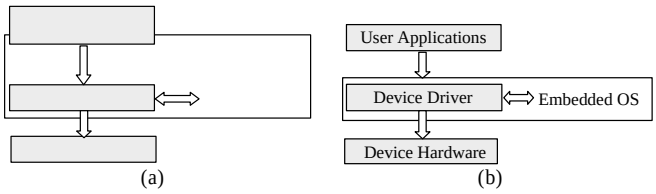

图 1 两个框架模型

1.2 间接模型

为了隔离用户程序和驱动之间的直接联系，由直接模型派生出图 1(b)所示的间接模型。图 1(b)中的用户程序不再直接与驱动程序交互，而是通过操作系统提供的接口来访问驱动程序。该模型将用户、驱动和操作系统 3 者之间的接口标准化。正确实现这些标准接口，就可以设计出具有良好可移植性的设备驱动，从而大大提高生产率。

2 可移植设备驱动解决方案

基于上述两个模型，抽象出具备高度可移植性的驱动程序体系结构。只要能够实现其中的接口，设备驱动的移植将变得非常简单。

2.1 体系结构

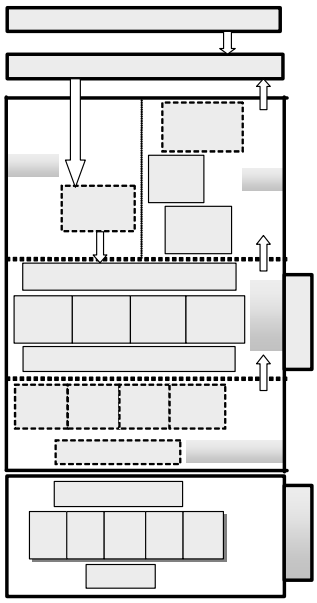

图 2 可移植设备驱动接口模型

图 2 中体系结构综合了两种模型，因此，涵盖了可移植驱动的开发和具体的移植工作。图中虚框模块为接口，即可移植驱动编写中用于移植而定义的接口函数等。后续 3 节将

基金项目：国家“863”计划基金资助项目(2003AA115130)
作者简介：杨永志(1980—)，男，硕士生，主研方向：嵌入式系统软件设计，嵌入式并行计算；唐玉华，教授
收稿日期：2005-10-08 **E-mail：**iamfoolberg@163.com

分三层逐个解释模型中除用户程序外的元素，重点阐述设备驱动模块。(图 2 中没有表示出用户程序和操作系统在硬件访问上的接口，那不属于本文讨论的内容，并且不影响本文的讨论。)

2.2 嵌入式操作系统

嵌入式操作系统是整个软件系统的核心，用户程序和驱动程序实际上被包含在其中，这里为了体系结构的叙述方便而在图中将其独立出来。

2.3 设备驱动

设备驱动主要由 3 个层次共 4 大部分组成，分别描述底层硬件访问(硬件访问层)、内部结构(核心层)和用户服务(基础/服务层中服务模块)以及与操作系统的接口(基础/服务层中基础模块)。

2.3.1 基础服务层

基础服务层根据扇入扇出可分两大模块：基础模块和服务模块。前者用于从操作系统获取基本的服务，后者向用户和操作系统提供硬件访问函数。基础服务层是驱动和操作系统的接口层。

(1)操作系统服务函数

此接口用于标准化设备驱动对操作系统的调用，使得驱动程序能够获得诸如中断安装、信号量等必需的系统服务。

一般情况下，设备驱动需调用操作系统的中断安装函数。在异步传输模式中，可能会用到操作系统的信号量(函数)。

某些情况下，驱动程序需要延时一段时间后再继续执行操作，所以需要提供一个 sleep 接口。由于操作系统可能提供自己的基本数据类型，定义设备驱动使用的基本数据类型来映射操作系统的数据类型对于可移植性是必要的。

较为少见的情况下，设备驱动可能需要和操作系统的内部数据(通过操作系统提供的一些服务函数)进行交互，以正确完成用户服务。例如使用 1394 或者 USB 实现两台或者多台 PC 机网络互连时，需要操作系统接收并处理 1394 或者 USB 的请求报文，并将应答报文发送到请求者。要满足这种需求，可以定义事件处理接口来实现设备驱动和操作系统之间更为灵活的交互，从而避免定义繁多的接口函数。另外，还需要内存分配函数。在概念上至少需要两种内存分配函数。

(2)操作系统接口函数

操作系统接口函数是操作系统定义的、需要由驱动程序实现的接口。这包括对硬件的高层控制、数据传输和其他特殊资源的控制与访问，以及不同的操作系统要求的功能。例如 RTEMS 要求硬件驱动提供 initialization、open、close 等 6 个函数，从而给用户统一接口。同时，通过实现这套接口而不是直接向用户程序提供访问接口，可以确保驱动与系统的结合不会出现诸如共享互斥之类的问题。

(3)模式 i 访问函数

硬件设备与操作系统或者用户程序的交互，其最终目的可以抽象成数据的读写两种操作。而大部分硬件可以工作在不同的模式下，例如 1394 设备的等时和异步传输。因此，对于某种模式下的操作，可以抽象为 read、block_read、write、block_write 等函数。

由于硬件与 CPU 是异步工作的，因此应该区分 read 和 write 是同步操作还是异步操作，设备驱动开发者应该提供同步和/或者异步读写函数。

(4)总控服务函数

总控服务函数用于提供对硬件的高层控制。包括打开、关闭设备，设备软复位，模式的选通、关闭等操作。另外，某些设备还需要一个控制函数用于实现特殊的功能。

2.3.2 核心层

设备驱动核心层将实现大部分的设备相关的函数。它们包括内存管理、底层操作和内部函数以及这些功能的基础——核心数据。实际上，整个设备驱动几乎都在这里实现。

设备驱动内的上下两层都是为可移植性而设立的。基础/服务层和硬件访问层将核心层包围在其中，从而使得在驱动的移植过程中可以绕开对核心层的修改——甚至完全不需要涉入其中。

由于核心层随设备的不同而变化各异，因此不可能抽象出一套通用的接口。另外，该层几乎不涉及到移植，故没有必要要求使用某种统一的接口。

(1)核心数据结构。核心数据包含了设备驱动中绝大部分的数据，它通常由一个数据结构涵盖几乎所有其他的数据。例如在 Linux 的 1394 驱动中，数据结构 hpsb_host*　host 实际上总括了所有的数据；同样的在 DDC 1553B 的驱动中，BuConf_p BuConf 起到了同样的作用。

核心数据包含两部分：静态预分配的数据和动态申请的数据。静态预分配的数据由编译器自动进行内存分配，而动态内存则需要操作系统提供内存分配和释放函数。因此，需要定义驱动所需内存的申请和释放函数。

(2)交换内存管理和访问。交换内存是指直接和卡上内存进行数据交换的内存区域。一般情况下，普通的内存区域可以作为交换内存，与卡上内存进行数据交换。但是，在许多情况下，必须分配特别的内存用于数据交换。例如，在一个如图 3 所示结构的系统中，进行 DMA 传输时，要求交换内存必须在共享内存中。

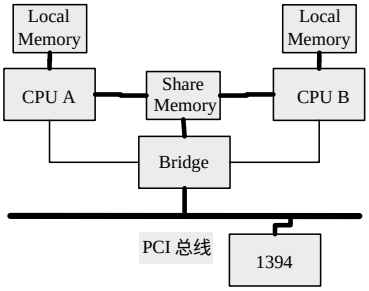

图 3 双处理器共享内存结构

图 3 中粗线表示允许的数据流向。当 1394 和 CPU 进行 DMA 传输时，桥会转发 DMA 请求到 CPU，但是，由于桥无法直接访问局部存储器，因此 DMA 数据只能在共享内存和 1394(卡上内存)中流动。这就要求数据交换区的分配不能在局部存储器中进行。

有了交换区内存的分配和释放这两个基本的函数，就可以进行交换区的管理了。如果操作系统没有提供对交换内存区的管理，那么设备驱动程序可能不得不自己管理这一片内存。应该尽量使用简单的内存管理方法，比如静态划分各个区域(可以参考 RTEMS 中基于共享内存的消息传递中，对共享内存区的管理方法)。

(3)卡上内存管理与访问。一般情况下，设备提供的内存都需要按一定的格式进行组织才能够正确使用。因此需要提供卡上内存的组织和管理函数。卡上内存的管理因设备而异。

(4)底层操作函数。底层操作主要是提供对硬件访问层和基于核心数据等的简单函数封装。如软复位、忙状态判定等。

(5)底层中断管理。很多情况下，设备驱动需要利用中断服务程序实现 CPU 与设备之间的异步通信。因此，需要提供中断管理。一般地，这些中断管理函数可以直接取自上一层的操作系统服务函数，但是需要进行必要的封装才能满足移植性的要求。

另外，中断发生时，设备驱动可能需要通知用户程序，作为可选，可以提供用户事件接口使得用户程序可以获知事件的发生。但是这些函数应该在服务模块中实现或者封装，以确保层次性不被破坏。

2.3.3 硬件访问层

硬件访问层提供基本的硬件访问函数给核心层。硬件访问函数是基于硬件资源的，硬件资源可以抽象为以下几类：卡上寄存器和内存，配置空间，特殊资源以及中断请求(线)。

一般情况下，中断请求的配置都包含在配置空间中。

(1)配置空间访问。配置空间访问主要是为驱动程序核心层提供寄存器和存储器的映射，将卡上的寄存器和存储器映射到 CPU 的内存或 IO 空间中，使得 CPU 可以通过自己的地址空间进行访问。因此，需要提供寄存器和卡上存储器的映射函数。这两个函数大部分情况下是通过对配置空间的直接读写来完成的。

另外，配置空间访问还可能包含其它一些内容。比如常见的 PCI 总线设备中的 PCI 配置头。一个较为重要的配置项是中断号的设置。有些嵌入式系统的外部设备中断号是静态分配的，即使如此，提供中断设置接口函数也是保险的。

(2)寄存器访问和存储器访问。寄存器和内存在被映射之后，即可被 CPU 直接访问。但是，这里可能涉及到诸如 big endian 和 little endian 等问题。因此，需要提供它们的访问接口函数。

需要注意的是，在寄存器访问和存储器访问的接口函数中，某些数据指针指向的数据区可能需要在交换内存中分配。

另外，还可能需字节的位转换函数(这通常发生在网络传输所用的硬件驱动程序中)。

(3)信息区访问和特殊资源访问。信息区通常包含一些设备描述信息。可以参考设备存储器的办法设置接口函数。由于硬件设备的特性，有些硬件设备的功能可能无法使用上面的接口进行描述，因此保留特殊资源这个模块是明智的。

2.4 硬件控制器

对于硬件，驱动程序设计者基本上不需要了解其内部的实现细节。就像汇编语言程序设计者只需要知道汇编指令集，而不需要了解 CPU 内部实现细节一样。因此，表 1 只是概要性的叙述。

表 1 硬件控制器资源

Configure Space		配置空间控制着诸如板上内存和寄存器的映射、中断的分配等必须的初始化项目。典型的是 PCI 设备的配置头
Registers		寄存器用于控制设备的工作或者返回设备的状态、错误码等，大致可以分为如下 3 类
	Config	用于配置设备的属性如工作模式等
	Control	驱动程序可以通过控制寄存器控制硬件进行工作
	State	状态寄存器返回硬件的工作状态或者错误代码等反馈信息
Memory		板上存储器多作为带格式数据缓冲区，常见的是控制信息块或者数据报文
IRQ		中断用于异步的通知 CPU 硬件事件的发生
FunctionalUnits		功能部件最终执行设备驱动程序指示的各种任务
SpecialResources		特殊资源用于表述未被上述条目描述的硬件资源

3 小结

作为嵌入式系统开发中较为常见的驱动移植问题，需要硬件和嵌入式系统软件方面的大量背景知识。驱动程序开发人员希望为某个硬件提供通用的驱动程序，嵌入式软件设计人员也期望能够以最小的代价移植现有的设备驱动。但是驱动开发人员不可能考虑到所有的嵌入式操作系统以及它底层的硬件实现，而嵌入式软件开发人员也不希望阅读和维护庞大的设备驱动源码并保证其可靠性。因此，定义一套二者之间的接口规范是很有必要的。它不仅可以使得驱动开发人员尽可能少考虑平台，也使软件设计人员无须过多涉入硬件驱动内部。更为重要的是，驱动移植人员通过实现间接模型，可以确保整个软件系统的可靠性不被破坏。

因此，高度可移植的三层设备驱动开发模型可以作为大部分硬件驱动开发的框架。

参考文献

1 Burns A, Wellings A. 王振宇, 陈　利译. 实时系统与编程语言(第 3 版)[M]. 北京: 机械工业出版社, 2004.

2 孙天泽, 袁文菊, 张海峰. 嵌入式设计及 Linux 驱动开发指南——基于 ARM 9 处理器[M]. 北京: 电子工业出版社, 2005.

3 Labrosse J J. 袁勤勇, 黄绍金, 唐　青等译. 嵌入式系统构件(第 2 版)[M]. 北京: 机械工业出版社, 2002.

(上接第 213 页)

和 J2EE 平台对协同设计环境下规范检验系统的实现技术进行了研究，实践证明该系统在一定程度上降低了结构设计过程数据处理的工作量，提高了设计效率，改善了设计质量。其不足之处在于系统仅能在知识库的辅助下进行简单的推理，不具备机器学习等高级智能性，其功能还有待在未来的工作中进一步完善。

参考文献

1 Fenves S J, Garrett J H, Kiliccote H, et al. Computer Representation of Design Standard and Building Codes [J]. The International Journal of Construction Information Technology, 1995, 3(1): 13-34.

2 Yang Q, Li X. Representation and Execution of Building Codes for Automated Code Checking[C]. Proc. of CADD Futere, The Netherlands, 2001: 315-329.

3 孙林夫. 基于知识的智能 CAD 系统设计[J]. 西南交通大学学报, 1999, 34(6): 611-616.

(上接第 229 页)

务器和 Web 服务器，集中管理数据，统一提供诊断服务。虽然这两种部署方式存在差异，但是各类人员对信息的需要却是相同的，因此系统从用户角色出发，进行功能和界面的设计，在这两种管理方式上都取得了成功的应用。

本文研发的设备状态点检管理系统，为企业点检设备管理提供了完整的解决方案，促进了中小型设备的信息化管理发展，目前已在武钢、马钢和酒钢等钢铁企业中大规模推广应用，其中仅在武钢大型厂的应用过程中，借助系统的诊断分析功能，就发现故障隐患 13 起，直接减少经济损失 5000 多万元[4]。

参考文献

1 Koc M, Lee J, 邓军民. 新一代电子维护系统的系统框架[J]. 中国机械工程, 2001, 12(5): 531-534.

2 徐　敏. 设备故障诊断手册[M]. 西安：西安交通大学出版社, 1998.

3 屈梁生, 何正嘉. 机械故障诊断学[M]. 北京: 机械工程出版社, 1985.

4 武钢公司大型轧钢厂.《设备状态点检管理系统》应用证明[Z]. 2005-03.