嵌入式技术丛书

VxWorks下设备驱动程序及BSP开发指南

周启平 张 杨 编著

- 本书以实用为目的，内容全面，凝聚了作者多年丰富实际开发经验
- 集VxWorks下设备驱动程序与BSP的开发于一体
- 采用循序渐进的手法，深入介绍了VxWorks下设备驱动程序的开发，并给出了与设备相应的驱动程序模板，以便于读者理解和学习
- 以图解的方式详细介绍了VxWorks映像的启动流程
- 深入讲解了VxWorks下BSP的配置和开发，以及与体系结构相关的注意事项

中国电力出版社
www.infopower.com.cn

嵌入式技术丛书

VxWorks下设备驱动程序及BSP开发指南

周启平 张 杨 编著

中国电力出版社
www.infopower.com.cn

内容提要

本书详细、深入地介绍了 VxWorks 下设备驱动程序及 BSP 的开发等内容。全书共 17 章，主要内容包括：外部设备及设备驱动程序概述、VxWorks 下设备及设备驱动程序、VxWorks 下设备驱动程序的分析、驱动程序的轮询和中断处理、编写字符设备驱动程序、编写网络设备驱动程序、BSP 概述、VxWorks 预内核初始化、BSP 的配置、BSP 开发等内容。

本书语言流畅、条理清晰、内容全面且深入浅出，以示例源代码加文字说明并结合编者多年实际开发经验编写而成，实用性强。适用于以 VxWorks 操作系统为基础的嵌入式系统开发、设计人员，也可供其他相关技术人员及爱好者参考。

图书在版编目（CIP）数据

VxWorks 下设备驱动程序及 BSP 开发指南 / 周启平，张杨编著. —北京：中国电力出版社，2004
ISBN 7-5083-2481-1
I.V... II.①周...②张... III.实时操作系统，VxWorks IV.TP316.2
中国版本图书馆 CIP 数据核字（2004）第 067158 号

责任编辑： 张娇
书　　名： VxWorks下设备驱动程序及BSP开发指南
出版发行： 中国电力出版社
地址：北京市三里河路 6 号　邮政编码：100044
电话：（010）88515918　　传　　真：（010）88518169
本书如有印装质量问题，我社负责退换
印　　刷： 北京丰源印刷厂
开本尺寸： 185×233　　**印　　张：** 19.5　　**字　　数：** 446 千字
书　　号： ISBN 7-5083-2481-1
版　　次： 2004 年 9 月北京第 1 版
印　　次： 2004 年 9 月第 1 次印刷
印　　数： 0001—4000
定　　价： 30.00 元

序　言

学习 VxWorks 的几年来，大部分的精力都花在 BSP 的配置和驱动程序的编写上，期间遇到过很多问题，也得到过一些经验。在不断深入的学习过程中，总是希望能有一本中文的资料可以参考，这个想法最终促成了这本书的诞生。

这是一本合作的书籍，周启平在其中花费了大量的心血。大部分的联络工作和大部分章节的编写都是由他完成的。由于工作中，我更多地涉及驱动程序的编写，于是参与了第 6 章、第 7 章、第 8 章和第 10 章的编写以及全书各章节的修改、校对工作。其余工作，包括与编辑的联系、其他章节的编写、修改等，都是由周启平完成的。

同时要感谢的是关心和支持我们的人：我们的家人、我们的直系领导、同事们和出版社的编辑们，感谢他们为这本书所付出的艰辛劳动。

本书注重实际的应用和经验，同时也非常关注基本的原理。全书共分为两个部分：第一部分是驱动程序的编写。驱动程序的编写涉及到很多的内容，从设备的访问方法到设备工作的方式等分别在各章节有所讨论。在总体概论之后，分章节讨论了各种类型设备驱动程序的编写方法并给出了相应的模板。最后为了让读者有更深入的了解，在第 10 章简单介绍了 PCI 设备、ISA 设备和文件系统的编写，这也是编写设备驱动程序中典型的应用。

第二部分以 x86 架构为例，基本上对 BSP 的各个部分都有所描述：描述了预内核的初始化过程、BSP 工作流程、BSP 相关组件、BSP 开发等。最后描述了不同体系结构上需要注意的问题。

在编写过程中，我们参考了一些公司的讲义、Wind River 公司的培训资料以及新上市的部分书籍。我们尽量使文章有自己的特色，但也难免会重复一些他人的东西。在修改过程中，我们尽量使语言更平易近人、更本地化，但水平有限，可能会弄巧成拙。总之，希望读者可以对此多包涵，也希望读者能给我们提出反馈意见，使我们的工作可以得到改进。

由于知识有限、时间仓促，不免会在编写的过程中出现这样或者那样的错误，还要请读者不吝赐教。下面是我们的电子邮箱：

张　杨：zhangyang@first.com.cn

周启平：qpzhou@sina.com

编　者

目　录

第 1 章　外部设备及设备驱动程序概述

在嵌入式系统开发过程中，对于编写设备驱动程序的人员来说，每写一个设备驱动程序就代表一个挑战。因为不仅需要开发人员具备软件编程能力和编写设备驱动程序的基本知识，同时还需要了解硬件设备的工作原理以及熟悉整个系统的体系结构。快速而有效地开发设备驱动程序，经验是一方面，背景知识和相关概念也是必不可少的。但是所有设备驱动程序的基本原理和功能都有相通的一面，有其规律可循。

刚接触编写设备驱动程序时，不要急于编写程序，而应该先熟悉硬件设备、了解它的工作原理及功能。然后从设备共性上来看待设备结构和功能，即所谓透过设备的表面现象看“本质”，以数据流为中心，了解设备与 CPU 及外界通信的方式。

本章内容不涉及到具体某类设备和编程，主要讨论一些概念、方法、技巧等知识，希望为读者实际编写设备驱动程序打下一定基础。

1.1　外部设备

在广泛使用的微型计算机系统中，外部设备就是以实现人机交互和机间通信为目的的一些机电设备。在对外部设备的控制过程中，系统不可避免地，有时候甚至要频繁地对设备接口进行控制和访问，因此，在编写设备驱动前对硬件设备本身的认识是非常必要的。

输入/输出设备都要通过一个硬件接口或控制器与 CPU 相连，例如终端显示器通过显卡与 CPU 连接起来；网络设备通过网卡与 CPU 连接起来；软盘驱动器通过软盘控制器与 CPU 连接起来。CPU 与接口板及设备之间的关系如图 1.1 所示。

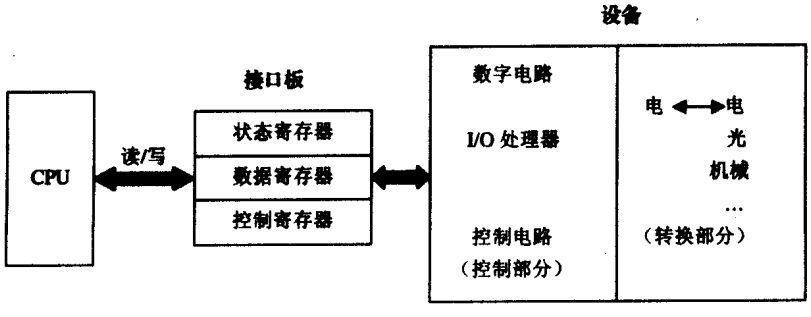

图 1.1　CPU 与接口板及设备之间的关系

通常接触到的外部设备有显示器、键盘、鼠标、硬盘等。设备具有以下共性：

（1）是一个硬件；

（2）具有输入/输出功能；

（3）通常是核心处理器的扩展部分；

（4）与 CPU 进行数据交换通常是靠设备接口板完成；

（5）系统为每一个连接到 I/O 总线上的设备分配独自的 I/O 地址。

1.2 外部设备的分类

在微型计算机系统中外部设备通常被划分成三种类型：字符设备、块设备和网络设备。下面分别讨论三种类型的设备。

1. 字符设备（character device）

字符设备是能够像字节流一样被访问的设备。在数据传输中以字符为单位进行传输，由字符设备驱动程序实现这种特性。字符设备驱动程序通常需要实现打开、关闭、读和写系统调用。像字符终端、串口、鼠标、打印机等就是字符设备的代表。字符设备可以通过文件方式（如/tyCo/0）来访问，它和普通文件之间的惟一差别在于，对普通文件的访问可以前后移动访问指针，而多数字符设备只能顺序访问数据通道。

字符设备通常具有如下的特点：

（1）在数据传输中可以传输任意大小的字符数据。像串口既可以一次传输一个字节，也可以通过数据流的方式来控制传输数据的大小。

（2）通常传输一组字符数据。

2. 块设备（block device）

块设备是指以“块”为单位对数据进行存取的设备。和字符设备一样，系统通过设备表访问块设备，但块设备上能够容纳文件系统。块设备包含整数个“块”，而每块包含 1KB 或 2 的多次幂字节的数据，它与字符设备的区别主要是内部数据管理的方式不同。块设备除了给操作系统提供接口以外，还要提供专门面向块设备的接口。如硬盘、光驱、软驱、磁带等就是块设备的代表。

块设备通常具有如下特点：

（1）在数据传输中传输固定大小的数据块。

（2）可以随机访问块设备中所存放的数据块。

3. 网络设备（network device）

任何网络通信都要经过网络设备，即能够和其他主机进行数据交换的设备。网络设备由网络子系统驱动程序负责发送和接收数据包，它与普通 I/O 设备不同，没有对应的设备文件。所以应用程序和网络接口之间的数据通信不是基于标准的 I/O 系统接口，而是基于 socket()、bind()、listen()、accept()、send()、connect()等系统调用。

1.3 I/O 设备的数据传送方式

控制 I/O 设备与 CPU 相连的接口和控制器都能支持 I/O 指令与外部设备交换信息。这些信息包括控制、状态和数据三种不同性质的信息，必须为它们分配不同的端口地址分别进行传送。

控制信息输出到 I/O 接口，告诉接口和设备要进行什么处理；从接口获得的状态信息表示 I/O 设备当前的状态；数据信息则是 I/O 设备和 CPU 真正要交换的信息。不同的 I/O 设备要求传送的数据类型是不同的，例如与字符型终端交换的数据必须是 ASCII 码，而不是二进制表示的数。

输入/输出方式通常包括三种方式。

（1）直接控制 I/O 方式：通过指令直接对端口进行输入输出操作称之为直接控制 I/O 方式。

（2）中断方式：采用中断实现数据的输入/输出。

（3）DMA 方式：采用 DMA 控制器实现数据传输。

在外部设备输入/输出方式选择中，通常依据硬件本身而采用不同的方式。例如字符设备通常采用中断方式；而对于一些传输大量数据的 I/O 设备（如磁盘、网卡等设备），通常采用 DMA 方式，以减少 CPU 资源占用。程序直接控制输入/输出方式和中断传送方式在后面的第 4 章中将会专门进行描述，而 DMA 方式主要是由硬件 DMA 控制器来实现数据的传送功能，所以仅在这一小节里简单介绍一下。

DMA 方式即直接存储器访问（Direct Memory Access）方式，也称为成组数据传送方式。DMA 方式主要适合于一些传输大量数据的 I/O 设备，例如磁盘、数模转换器等设备。由于这些设备本身传输字节或字的速度非常快，通过执行输入/输出指令的方法或中断方法来传输数据，将会造成这些数据的丢失。而 DMA 方式能使 I/O 设备直接和存储器进行成批数据的快速传输。数据一旦到达端口，就直接由 DMA 控制器送到存储器；同样，DMA 控制器也能直接从存储器取出数据并把它们送到 I/O 设备中去。

DMA 控制器或接口一般包括四个寄存器：状态控制寄存器、数据寄存器、地址寄存器和字节计数器。这些寄存器在数据传送前要进行初始化操作；每个字节传送后，地址寄存器加 1，字节计数器减 1。

系统完成 DMA 传送的步骤如下所示：

（1）DMA 控制器向 CPU 发出 HOLD 信号请求使用总线；

（2）CPU 发出响应信号 HLDA 给 DMA 控制器，并将总线让出，之后 CPU 不再使用总线，而 DMA 控制器获得总线控制权；

（3）传输数据的存储器地址（在地址寄存器中）由 DMA 控制器通过地址总线发出；

（4）传输的数据字节通过数据总线进行传送；

（5）地址寄存器加 1；

（6）字节计数器减 1；

（7）如字节计数器非 0，转向步骤（3）；

（8）否则，DMA 控制器撤消总线请求信号 HOLD，数据传送结束。

1.4 设备驱动程序

设备驱动程序是直接控制设备操作的那部分程序，也是设备上层的一个软件接口。设备驱动程序的功能是对 I/O 进行操作，实际上从软件角度来说就是对 I/O 端口地址进行读写操作。只要系统访问设备就会调用驱动。从这一点可以看出，驱动程序不能自动执行，只能被系统或应用程序调用。

在嵌入式系统中调用设备驱动通常有三种方式：应用程序直接调用、应用程序通过操作系统内核调用、应用程序通过操作系统的扩展模块进行调用。如图 1.2 所示为应用程序调用设备驱动。

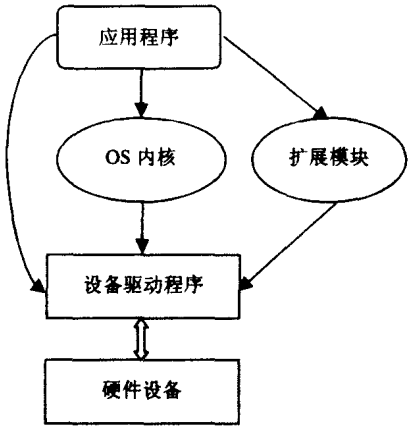

图 1.2 应用程序调用设备驱动

不同的调用方式各有自己的优缺点，下面分别描述它们的优缺点。

（1）应用程序直接调用设备驱动程序。

优点：可以高效访问设备（因为绕过操作系统，而直接访问设备，减少了系统开销时间），减少了系统代码开销，适合简单的嵌入式系统。

缺点：所编写的应用程序移植性差（因为应用程序直接与设备驱动关联），且用户自己来管理设备，不适合复杂的嵌入式系统。

（2）应用程序通过操作系统内核调用设备驱动程序。

优点：所编写的应用程序移植性好（统一的标准接口函数），且设备由操作系统管理，适合复杂的嵌入式系统。

缺点：操作系统管理设备，增加了系统开销。

（3）应用程序通过操作系统的扩展模块调用设备驱动程序。

扩展模块是嵌入式操作系统可供用户选择的模块。当用户需要某个扩展模块时才把该模块加入到系统中，如网络模块、文件模块等。它的优缺点类似第二种调用方式，但这些扩展模块依赖于操作系统内核。

至于采用哪种方式，视用户系统的复杂性及系统本身要求而定。在嵌入式应用开发中，用户通常采用第二种和第三种方式。

1.5 设备驱动程序的主要功能

设备驱动程序通常包含六个主要功能。

（1）对设备进行初始化。初始化的目的是使设备处于某种工作状态，以便用户程序访问该设备。譬如串口初始化包括设置串口波特率、数据位、奇偶校验位、停止位等。

（2）打开设备操作。打开设备操作实际上是查询用户指定的设备，并查看用户是否可以使用该设备。因为设备是共享资源，当设备正在被使用时，系统要对它进行保护，禁止其他任务对设备进行操作，直到设备资源被释放。

（3）关闭设备操作。关闭设备操作就是释放设备资源。任务对设备完成操作后，必须进行关闭设备操作，否则设备总是处于被占用状态，其他任务无法使用。与打开设备操作相对应，有打开操作就应该有关闭操作。

（4）从设备上接收数据并提交给系统。这项功能通常就是所说的读操作，接收外部传输来的数据。接收数据采用的方式有查询方式、中断方式和 DMA 方式。

（5）把数据从主机上发送给设备。这项功能对应通常的写操作，把主机上的数据传送给外界。通常系统主动调用该操作进行数据发送，有时也采取中断方式发送数据。

（6）对设备进行控制操作。在使用设备过程中，有时根据应用的需要对设备进行控制（例如改变设备某个状态），而控制操作就能提供这种功能。例如，在使用串口过程中，有时需要改变串口波特率的设置，这就需要用到这种控制操作。

1.6 设备驱动程序的组成部分

根据设备驱动的主要功能，设备驱动程序大致由下面几部分组成。

（1）设备驱动程序的注册函数。驱动程序开发人员需要编写一个函数把相关的驱动程序（打开、关闭、读/写等）注册到系统设备驱动程序表中，以便系统对其管理及使用。

（2）设备驱动程序的卸载函数。当系统不需要使用某设备时，为了节省资源开销，需要从驱动程序表中卸载该设备驱动程序并释放设备占用的资源。这些操作由设备驱动程序的卸载函数完成。

（3）设备的打开和关闭函数。这两个函数的功能在设备驱动程序主要功能部分已经作过描述，需要注意的是有打开设备操作，必须相应地有关闭设备操作。

（4）设备的读/写操作函数。设备与外界的通信主要由这两个函数完成，主要功能是完成设备与 CPU 之间的数据传输。

（5）设备控制函数。在对设备的操作中，用户有时根据需要对设备进行控制。如寄存器设置、设备相关操作等。

（6）中断服务函数。该函数通常在对设备进行读/写操作时使用，当设备上接收到数据或数据发送结束时，通过触发硬件中断信号，向系统报告这一状态，系统便执行中断服务函数进行相应的处理。

1.7 设备驱动程序的相关概念

1.7.1 设备驱动程序表

应用程序对设备进行操作是通过操作系统的 I/O 管理子系统来进行的。而 I/O 管理子系统对设备的管理在系统中是通过一张表来操作，通常把这张表称为“设备驱动程序表”。设备驱动程序表实际上是一个数组，它的每一个数组项是一个设备驱动程序地址表结构体，这个结构体包含初始化、打开、关闭、读、写、控制等操作函数的入口地址。如果哪一个操作函数空缺，那么它的入口地址

为NULL。

下面是一个设备驱动程序表的示例。

```
/* 设备驱动程序结构 */
typedef struct {
  xx_device_driver_entry initialization;     /* 初始化程序 */
  xx_device_driver_entry open;               /* 打开设备处理程序 */
  xx_device_driver_entry close;              /* 关闭设备处理程序 */
  xx_device_driver_entry read;               /* 读处理程序 */
  xx_device_driver_entry write;              /* 写处理程序 */
  xx_device_driver_entry control;            /* 特殊功能处理程序 */
}   xx_driver_address_table;

/* 系统设备驱动表 */
xx_driver_address_table Device_drivers[5] =
{
/* 控制台驱动 */
{ console_initialize, console_open, console_close,
     console_read, console_write, console_control },
/* 时钟驱动 */
{ Clock_initialize, NULL, NULL, NULL, NULL, Clock_control },
...
};
```

1.7.2 设备名称

操作系统通常提供了给设备命名的功能，系统可以调用相关的系统服务给设备注册一个名字。应用程序可以根据这个名字获得对应设备的主从设备号以及对设备进行打开、关闭、读、写等操作。不过系统分配给设备的设备号是一个整数，这对用户而言非常不直观明了。而设备名克服了这方面的缺点，一看名称一般就知道使用的是什么设备，否则用户在应用程序中使用设备时会带来诸多不便。

下面是一个注册设备名的函数示例。

```
status xx_io_register_name(char *name, int major, int minor);
```

1.7.3 设备驱动接口

当应用程序调用一个操作系统的I/O管理子系统所提供的服务时，操作系统将调用对应的设备驱动程序，应用程序传输给I/O管理子系统的信息将被传输给对应的设备驱动程序。

通常，操作系统I/O管理子系统调用的设备驱动程序具有如下的类似原型：

```
xx_device_driver io_xx(
xx_device_major_number     major,        /* 主设备号 */
xx_device_minor_number     minor,        /* 从设备号 */
void                       *argument     /* 传递给设备驱动程序的参数 */
);
```

值得注意的是“argument”参数通常传送给设备驱动程序，是设备驱动程序的输入参数项。因此，该参数的格式和内容与设备驱动程序有关。

1.7.4 设备驱动程序的运行环境

应用程序开发者和驱动程序开发者必须注意以下与操作系统 I/O 管理子系统相关的几点内容。

（1）设备驱动程序在调用它的任务上下文中执行，因此如果驱动程序阻塞了，那么调用它的任务也就阻塞了。

（2）尽管驱动程序会把调用它们的任务恢复到调用前的原始状态，但设备驱动程序还是可以自由地改变调用它的任务的模式。

（3）设备驱动程序可能被中断服务程序调用。

（4）I/O 管理通常只能调用本地的设备驱动程序。

（5）设备驱动程序可以调用所有的对本地或全局对象操作的系统服务，包括 I/O 管理系统的服务。

（6）I/O 管理只提供了一个设备驱动程序的框架，而不关心设备驱动的结构和具体的操作。

小　结

这一章主要从总体上对外部设备和设备驱动程序进行了描述，其中包括设备的分类、设备的数据传送方式等。在设备驱动程序方面，则主要描述了驱动程序的主要功能、驱动程序的组成以及相关的概念（例如设备驱动程序表、设备名称等）。这些描述不是针对某种操作系统或某种设备，而是从宏观上来说明设备及设备驱动程序，以便读者能够对其有所了解。

第 2 章　VxWorks 下设备及设备驱动程序

前一章的内容，从宏观的角度讨论了硬件设备结构和设备驱动程序的一些知识，这为在 VxWorks 操作系统下编写设备驱动程序奠定了基础。VxWorks 操作系统下的设备驱动程序同样遵循了基本设备驱动程序的原理、功能以及组成部分，同时它也具有自己的特色和特点。下面将一一讨论。

2.1　VxWorks 下的设备

2.1.1　VxWorks 下设备的分类

在 VxWorks 下，设备分为如下几类：

- 字符设备，大多数的传输设备以及显示终端；
- 随机存储块设备，这种设备主要是指磁盘；
- 虚拟设备，比如管道、套接字都属于这种设备；
- 控制监视设备，一般是指用于控制数/模变换的 I/O 设备；
- 网络设备，那些与网络协议挂接的通信设备。

这些设备通常包括网络设备、磁盘设备、串并口、PCI 桥、VME 桥等，如图 2.1 所示。

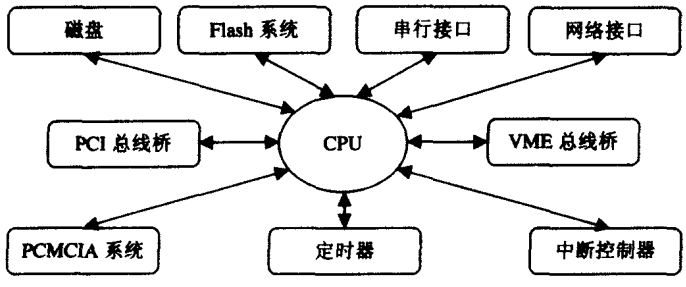

图 2.1　VxWorks 下的设备

根据操作方式的不同，VxWorks 操作系统中 I/O 又分为基本 I/O 系统和缓冲 I/O 系统，操作系统为两种不同的 I/O 操作方式分别提供了不同的 C 语言函数库。基本 I/O 系统使用了与 UNIX 标准相兼容的 C 语言函数库，而缓冲 I/O 系统则使用了与 ANSI-C 标准相兼容的 C 语言函数库。为了获得更快的速度和更好的兼容性，VxWorks 系统中的 I/O 都是经过优化设计的，而速度和兼容性对于 VxWorks 这样的实时系统来说都是至关重要的。

还有一些特殊类型的设备（如网络设备），由于其自身的特性，虽然不是通过标准 I/O 来进行存取的，但是也都有它们各自相关的规范。

2.1.2 VxWorks下设备的命名

在VxWorks系统中，设备是被当作文件来管理，而这种机制实际上就是仿照UNIX操作系统的设备管理机制。打开设备操作，是通过打开指定的文件来操作I/O设备。这个指定的文件可以表示：一个非结构化的原始设备或者是具有文件系统的结构化的随机存储设备上的一个逻辑文件。在对设备命名时通常有一些约定的标准格式，大部分的设备名会以“/”开头，并使用“/”将设备名和文件名分开表示；但是不基于NFS的网络设备会采用远程主机名加冒号的形式，例如host:download；而使用dosFs格式的文件系统设备名经常以大写字母、数字的组合形式加一个冒号构成，例如DEV1:。

在表2-1中列举了VxWorks系统中可能出现的设备名。

表2-1 VxWorks常用设备说明

设备名	设备
/tyCo/0	串口设备
/fd0	软盘驱动器
/ide0	IDE接口设备
/ata0	ATA接口设备
/pipe/0	管道设备
/sd0	SCSI设备
/tffs0	Trueffs闪存设备
/pcmcia0	PC存储卡块设备

注意

不能以单独的“/”作为设备名，否则将会引起系统的混乱。

2.1.3 VxWorks下设备相关概念

1. 设备数据结构（DEV_HDR）及设备描述符

大部分的设备驱动程序都可以为同类型的设备的多个实例提供服务，例如系统中串口设备的驱动程序可以为不同的通道服务。

在VxWorks I/O系统中，设备驱动程序需要定义一个被称为设备头（DEV_HDR）的数据结构。在这个结构中包括了设备名称和为设备所服务的驱动程序索引号。I/O系统中的所有设备的设备头都保存在一个叫做设备列表的链表结构中。实际中，每一个设备都会有更多的数据存储在更大的数据结构中，设备头就是这个数据结构的起始部分。而这个更大的数据结构则称为设备描述符，如图2.2所示。它包括了很多和设备相关联的数据。

设备头数据结构——DEV_HDR原型如下：

```
typedef struct              /* DEV_HDR - 所有设备结构的设备头 */
```