维普资讯 http://www.cqvip.com

实时操作系统 VxWorks 下 I/O 设备驱动程序的编写技巧

Development of I/O Device Drivers under RTOS VxWorks

尚 涛 施 亮 吴智铭
Shang,Tao Shi,Liang Wu,Zhiming

摘要：近年来，嵌入式设备得到高度重视，在通信，自动化等领域的应用也越来越广泛。在开发嵌入式设备所用的实时操作系统中，VxWorks 是较受欢迎的的一种。开发嵌入式设备经常遇到的一个问题就是如何编写高效可靠的设备驱动程序。本文对 VxWorks 下设备驱动程序开发的几个主要方面进行了详细介绍，为初涉 VxWorks 的开发人员提供有参考。

关键词：**VxWorks; 驱动程序; 实时操作系统**

文章编号:1008-0570(2004)01-0052-04

Abstract: Today, great research efforts have been paid to the embedded devices which are employed in many fields including communication, automation engineering, and so on. As one of the RTOS (Real-Time Operating System) for developing embedded devices, VxWorks becomes very popular. In applying VxWorks in engineering area, the main difficulty that programmers are often confronted with is the development of efficient and reliable device drivers. This paper presents in detail the primary aspects in this regard. It will provide a useful reference for VxWorks beginners.

Key words: VxWorks; Driver; RTOS

1 前言

在工业自动化，通信等领域，国内系统开发很多是基于 Windows，Uinx 或 Dos 操作系统。但是这些操作系统本身不是设计用来开发实时嵌入式设备，因而导致一些无法避免的问题，比如无法快速响应中断，操作系统自身占用资源过多等。要克服这些问题，根本的方法就是选用开发实时嵌入设备的专用操作系统如 QNX, pSOS, VxWorks 等。

在嵌入式设备开发中所使用的众多实时操作系统里，VxWorks 以其卓越的性能占据了重要的地位。它具有小巧的内核、广泛的硬件支持以及第三方软件开发商的支持等优点。类 UNIX 风格和 POSIX 规范兼容性使得广大 UNIX 程序员可以轻松地掌握该系统。同时，VxWorks 所具有的良好开发环境极大地提高了程序开发的效率。在使用 VxWorks 开发嵌入式系统时，一个重要的问题就是如何为系统中的关键 I/O 设备写出高效的驱动程序。本文讨论了 VxWorks 下设备驱动程序的结构和编写设备驱动程序的一些关键问题。

2 驱动程序与系统的关系及其结构

2.1 驱动程序与 I/O 系统的关系

VxWorks 中，驱动程序仅提供几个低级的 I/O 函数来完成简单的数据输入输出操作，而由设备无关的 I/O 系统来处理具体的高级协议如面向字符设备的协议和面向块设备的协议。这种系统结构降低了驱动程序的复杂性，使得驱动程序的编写更容易。

I/O 系统相当于一个交换机，用于把 I/O 请求转发给合适的驱动程序以进行处理。同时，系统也提供了几个库来支持标准的字符设备和块设备。

字符设备的驱动程序和 I/O 系统直接作用，调用关系如图 1 所示。字符设备调用驱动程序安装函数 iosDrvInstall()在 VxWorks 中安装驱动程序。该函数有 7 个参数，分别指向驱动程序提供的标准 I/O 函数：Create, Delete, Open, Close, Write, Read 和 ioCtl。如果设备功能上不支持，有些函数不一定要实现，对应的参数可为 NULL。IosDrvInstall()只是为驱动程序在驱动程序表中分配了一个位置，要使用驱动程序还要调用设备安装函数 iosDevAdd()。IosDevAdd 把设备名和驱动程序号写到数据结构 DEV_HDR 中，并把它加到系统的设备列表中。

块设备驱动程序和文件系统作用，再由文件系统与 I/O 系统作用。块设备驱动程序不使用 iosDrvInstall ()来安装驱动程序，而是通过初始化块设备描述结构 BLK_DEV 或顺序设备描述结构 SEQ_DEV，来实现驱动程序提供给文件系统的功能。块设备驱动程序在系统中的调用关系如图 2 所示。类似的，块设备驱动程序不使用 iosDevAdd()来把驱动程序装人 I/O 系统，而

维普资讯 http://www.cqvip.com

是使用文件系统设备初始化函数如 dosFsDevInit()等来完成。实际上，文件系统把自己作为一个驱动程序装到 I/O 系统中，并把请求转发给实际的设备驱动程序。

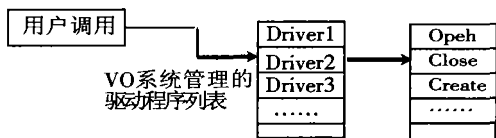

图 1

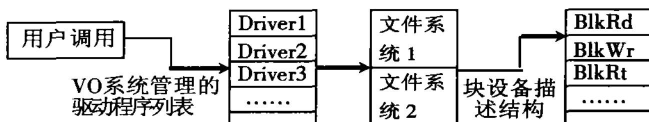

图 2

2.2 驱动程序的结构

驱动程序的结构包括三个部分：初始化部分，函数功能部分和中断服务程序 ISR。初始化部分初始化硬件，分配设备所需的资源，完成所有与系统相关的设置。如果是字符设备，首先调用 iosDrvInstall()来安装驱动程序，把中断向量和 ISR 挂上，然后调用 iosDevAdd()来把驱动程序加到 I/O 系统中；如果是块设备，首先把中断向量和 ISR 挂上，在内存中分配一个设备结构，然后初始化该结构。用户要使用该设备时，先调用设备初始化部分 myInit()(一般放在 sysLib.c 中)，再调用设备创建函数 myDevCreate()返回一个 BLK_DEV 结构的指针，供文件系统初始化函数如 dosFsDevInit()使用。

以下为块设备的初始化示例代码：

```
struct    MyDevice{  //自定义的设备结构
BLK_ DEV   dev; //必须在结构的开始
Int interrupt; //设备使用的中断号
........
}
int myInit(){    //初始化函数
MyDeviceInit(); //硬件初始化函数
MyDevice * Device =(MyDevice*)malloc (sizeof(MyDevice));  //为设备结构分配内存
Device->dev.bd_blkRd=Read; //初始化设备结构
Device->dev._blkWrt = Write;
......
intConnect(intToVec(9),my_ISR); //连接中断向量和中断服务程序 ISR
......
}
```

ISR 处理硬件中断，管理具体的硬件输入输出，同时和驱动程序的其它部分通信。ISR 中第一条指令用来读 APIC 的中断服务寄存器，以通知 CPU 已经接到中断请求。

```
my_ISR(int val){  //中断服务函数
sysOutByte(0xa0,0x02); //中断响应
......
semGive(my_sem); //通知其它程序中断处理完毕
}
```

函数功能部分完成系统指定的功能，对于字符设备，这些函数就是指定的 7 个标准函数；对于块设备，则是在 BLK_DEV 或 SEQ_DEV 结构中指定的功能函数。应当注意的是，系统在调用块标准函数时，传递的设备结构指针是设备结构中 BLK_DEV 结构的指针，由于 BLK_DEV 定义在设备结构的开始处，该指针实际上也就是设备结构的指针。

```
STATUS Read (MyDevice* pDev,int startBlk,int numBlk,char* pBuf){
SemTake(my_sem,WAIT_FOREVER);   //等待设备 I/O 执行完成
......
}
```

3 中断服务程序 ISR

中断服务程序用来处理来自硬件的中断，是设备驱动程序的重要组成部分。为及时响应外部中断，防止中断丢失，中断服务程序应该尽量的小，只把最必要的任务放在中断服务程序里面执行。

一般在系统启动，硬件设备成功初始化之后将 ISR 与中断向量挂上；也可以在系统启动后的任何时刻挂中断向量。调试中经常采用后一种方式。在 VxWorks 中有两个不同的函数可提供挂中断：intConnect 和 pciIntConnect。两者的区别是 intConnect 使用的中断向量是独占的，pciIntConnect 则可在各个不同的 ISR 之间共享中断向量。实际上 pciIntConnect 内部调用了 intConnect 函数，在内部使用一个链表来管理多个不同的 ISR。pciIntConnect 要求每次进入 ISR 都要检查硬件的寄存器，证实中断的确是由 ISR 服务的硬件产生。如果硬件的寄存器表明该硬件并未产生中断，则 ISR 立即退出，以让挂在同一个中断向量上的其它 ISR 有机会检查是否有中断产生。pciIntLib.c 中的代码清楚的说明了这个问题：

```
void pciInt ( int irq ) {
```

维普资讯 http://www.cqvip.com


```
PCI_INT_RTN *pRtn;
for (pRtn = (PCI_INT_RTN *)DLL_FIRST (&pciIntList[irq]); pRtn ! = NULL;
    pRtn = (PCI_INT_RTN *)DLL_NEXT (&pRtn->node))
  (* pRtn->routine) (pRtn->parameter);
}
```

当 PCI 总线上有中断发生时，系统调用 void pciInt(int irq)函数，再由 pciInt 使用内部的链表来依次调用挂在该中断上的 ISR。如果某个 ISR 不能正常退出，就会影响到其它 ISR 的运行。在调试时为了检查中断向量是否已经和 ISR 可靠的连接上，可以在命令行上或程序中直接调用 pciInt 来查看 ISR 是否被触发。

在硬件确定的情况下，可以小心设计保证各个硬件使用不同的中断，这样对 PCI 上的设备也可直接使用 intConnect 来挂中断。

需要说明的是，ISR 挂上中断向量的过程不是简单的在向量表中设置中断向量值。VxWorks 除了设置中断向量值以外，还在与中断向量相连的 ISR 加上了一层薄薄的包装，包括 ISR 执行前保存寄存器值，设置堆栈以及 ISR 执行后恢复寄存器和堆栈。在中断频繁的场合，系统中中断堆栈有可能被耗尽而溢出。为了避免上述情况发生，必须修改系统的中断堆栈大小，即在 config.h 中加入以下代码：

```
#define INCLUDE_KERNEL
#define ISR_STACK_SIZE 0x1000 //表示系统中中断堆栈的大小为 4k
```

由于中断处理程序的特殊性，中断处理程序中不能使用可能导致阻塞的函数，如 printf，semTake 等，具体不可使用的函数列表可以在<<VxWorks Programmer Guide>>中查到。有时候为了调试方便，希望在 ISR 中打印一些信息，系统提供了一个与 printf 等价的函数 sysLog，该函数可接受 7 个参数。它是非阻塞的。比较而言，printf 函数要在打印任务完成后才返回，sysLog 只把打印任务放到系统的打印队列中就返回。在 ISR 中虽然不可以使用 semTake，但可以使用 semGive (互斥类型的除外)。一般使用 semTake 和 semGive 在 ISR 和普通程序间通信：当一个中断产生，ISR 完成必要的任务后，调用 semGive 通知另外一个使用 semTake 等待 ISR 信号的任务，该任务收到 semGive 释放的信号后，继续完成 ISR 中不便处理的任务。

4 关于卡上 I/O 地址和内存地址的映射

在系统开发中，X86 体系使用较多。X86 体系中经常遇到的问题就是如何把 PCI 总线上设备的资源映射到系统中，准确的说是：把 PCI 总线上的物理资源映射成 CPU 的本地资源。有两种处理方式：动态寻找和静态指定。静态指定适合硬件完全可以确定的情况，动态寻找主要用在系统中硬件不能确定的情况。

静态指定。系统中使用卡的数目已知，卡上的 I/O 地址和内存地址固定已知，可以直接在 syslib.c 文件中修改。比如卡上 I/O 空间的基地址为 0x3000，大小为 0x100，内存空间基地址为 0xee000000，大小为 0x100000，在 syslib.c 文件中找到 PHYS_MEM_DESC 结构，在其中加上对卡 I/O 空间和内存空间的说明如下：

```
#define INIT_STATE_MASK (|\
        VM_STATE_MASK_WRITABLE |
VM_STATE_MASK_ CACHEABLE)
#define INIT_STATE (VM_STATE_VALID |
VM_STATE_ WRITABLE |\
    VM_STATE_CACHABLE_NOT)
PHYS_MEM_DESC sysPhysMemDesc [] ={
  ......
  {
  (void *) 0xee000000, //映射到 cpu 端的基地址

  (void *) 0xee000000, //卡上内存的基地址
  0x1000, //卡上内存的大小为 4k
  INIT_STATE_MASK, //卡上内存初始化状态的掩码
  INIT_STATE //卡上内存的状态
  },
  ........
}
```

这样系统在每次启动后都会把该结构中指定的 I/O 空间和内存空间放到内存映射表中，驱动程序可以直接使用这些地址读写。

动态寻找。系统中使用的卡的数量不确定，需要在系统启动时才可确认，这样的情况下使用动态寻找。动态寻找要确定资源的类型 (I/O 或内存)，大小和基地址。一般把这段代码放到一个文件里，然后在 syslib.c (位于$(HOME)/target/config/bsp_type/下)中把这段代码包含进去。在该文件中的硬件初始化函数 void sysHwInit (void)调用该代码。

映射 PCI 资源的一般过程大概如下：扫描 PCI 总线，找到总线上的感兴趣的硬件，然后查询该硬件上的资

源信息，最后培植该硬件的资源信息让 VxWorks 系统知道。具体设置 I/O 空间和内存空间的代码如下所示：

```
int sysMyPciInit()
{
 ULONG dwBaseOAddr,deMemRang, deTmpAddr;
  int pciBusNO, pciDevNo, pciFuncNo;
  if (pcifindDevice (VENDOR_ID,DEVICE_ID,0,
pciBusNo,pciDevNo,&pciFuncNo) ! = OK)
    return ; //查找总线上感兴趣的硬件
  pciConfigInLong (pciBusNo,pciDevNo,pciFuncNo,
PCI_CFG_BASE_ADDRESS_0,\
      &dwBaseOAddr); //得到该硬件的 I/O 基地址,保存给后面用
  pciConfigOutLong(pciBusNo,pciDevNo,pciFuncNo,
PCI_CFG_BASE_ADDRESS_0,\
      0xffffffff);
  pciConfigInLong (pciBusNo,pciDevNo,pciFuncNo,
PCI_CFG_BASE_ADDRESS_0,\
      &dwMemRange); //得到硬件上的内存大小
  pciConfigOutLong(pciBusNo,pciDevNo,pciFuncNo,
PCI_CFG_BASE_ADDRESS_0,\
      &dwBaseAddr);  //恢复硬件上的原来内存基地址值
  if(dwMemRange == 0)
      return;
  dwMemrange = ~ (dwMemrange& PCI_MEM-
BASE_MASK) + 1;
  dwBaseOAddr = dwBaseOAddr & PCI_MEM-
BASE_MASK ;
  if (sysMmuMapAdd ((void*)deBaseAddr,deMem-
Rang,INIT_STATE_MASK,\
       INIT_STATE) != ok) //将 PCI 总线上硬件的资源映射成 CPU 本地资源
       Return;
}
```

PCI_CFG_BASE_ADDRESS_0 是 pci 头文件中定义的常量，实际上就是 PCI 规范中定义的 PCI 桥的 I/O 寄存器。PCI_MEMBASE_MASK 是在 pci 头文件中定义的常量。INIT_STATE 和 INIT_STATE_MASK 用来设置 PCI 设备上内存的存取性质，一般设置该内存区为合法，可写，不缓存。对这些性质的具体说明可参考<<VxWorks Programmer Guide>>中的说明。

5 支持 SELECT()

在驱动程序中支持 select()机制可以让任务同时等待从多个设备来的输入并设置等待某个设备输入的超时时间。在驱动程序中添加 select 机制很容易，selectLib 库提供了这方面的支持。驱动程序要做的工作如下：首先，在设备描述结构里声明一个 SEL_WAKEUP_LIST 结构并在设备创建函数里调用唤醒表初始化函数 selWakeupListInit()来初始化该结构；当函数 ioctl()被调用时(参数为 FIOSELECT)，应该用节点增加函数 selNodeAdd () 把函数 ioctl () 中的参数 SEL_WAKEUP_LIST 加到链表 SEL_WAKEUP_LIST 中；然后用唤醒类型函数 selWakeupType 检查任务是读设备还是写设备；最后当设备准备好以后，调用唤醒函数 selWakeup 来确定调用任务目前是否阻塞，这样可避免设备准备好而调用任务由于其它原因而挂起，或者调用 selWakeupAll()函数来唤醒所有等待该设备的任务。

6 结束语

嵌入式系统中 I/O 设备是关键的一环，为 I/O 设备编写高效无误的驱动程序是开发嵌入式系统中的重要问题。本文结合我研究室的开发实践，对 VxWorks 系统下驱动程序编写中的几个主要方面进行了介绍，为初涉 VxWorks 下驱动程序开发的人员提供参考。

参考文献

[1] Wind River System Inc. VxWorks Programmer Guide, Edition 1, 1999.
[2] Scott Johnson Geurt Vos. How to translate a PCI device physical address into CPU local. comp.os.VxWorks,1999,8.
[3] PCI Special Interest Group. PCI Local Bus Specification. Product version Revision 2.1. June 1 1995

作者简介：尚涛，男，1975 年生，上海交通大学自动化所硕士研究生，研究方向为网络存储设备。施亮，男，上海交通大学副教授。吴智铭，男，上海交通大学教授。电话：(021)62932070，E-mail: woshishangtao@263.net
(200030 上海交通大学自动化系浩然大厦 1416 室)
尚 涛 施 亮 吴智铭

单片机开发设计 IC解密
专业 IC 解密 AT89CXX、W78EXX、PIC12/16CXX、HD647180 、PSD313、813……
承接各式单片机程式修改
欢迎合作开发新旧产品
地址：广州市番禺区市桥富华东路番禺电脑城 50 铺
电话：020-84666224 34517676 传真：020-34805890
E-mail:CHU1943@HOTMAIL.COM 邮编：511400