vxWorks 串口驱动程序设计

曹桂平

（中国科学技术大学 近代物理系 快电子实验室 合肥 230026）

摘要：串口驱动程序由于涉及硬件寄存器较少，所以是外围器件驱动程序编写中相对较为容易的一类，但其与内核操作系统的联系上和其它驱动程序都具有相同的层次结构。论文着重阐述了 vxWorks 操作系统下串口驱动程序与其上层虚拟驱动层的交互，并详细解释了虚拟驱动层和串口驱动程序的功能以及初始化过程。论文最后结合代码示例给出了串口驱动程序的架构。

关键词：vxWorks 虚拟驱动层 串口驱动

Serial Driver Design in VxWorks Real Time System

CAO Guiping

(University of Science and Technology of China, Department of Modern Physics, Fast Electronics Lab, Anhui, Hefei, 230026, China)

Abstract: Serial I/O Driver is relatively simpler than other peripheral drivers for its corresponding less register operations, but from the view of architectural relationship with kernel, it has the same organization with those other drivers. This paper firstly expounds the contacts between serial driver level and upper virtual driver level, and then in detail explains the functions of these two levels and their corresponding initialization processes. At last, the paper, combining with concrete codes, gives the architecture of a serial driver.

Keywords: vxWorks, virtual driver, serial driver

串口驱动程序编写应该是所有外围器件驱动中较为容易的一类，因为一般串行设备控制寄存器较少，所以思路较易理清，代码编写上较为容易。设备驱动程序编程都遵循一种约定，即系统提供中间层（对底层驱动而言，即系统提供回调函数），从而将底层驱动与上层（一般为用户接口层）隔离开来。这种隔离使得底层驱动程序员可以编写针对具体硬件的驱动程序并加入到内核中去，这样的中间层一般被称为虚拟驱动层或者接口层。硬件驱动程序员需要清楚了解驱动程序与系统的接口所在。对系统接口层了解的越清楚，编写出的驱动程序将更具健壮性和稳定性。

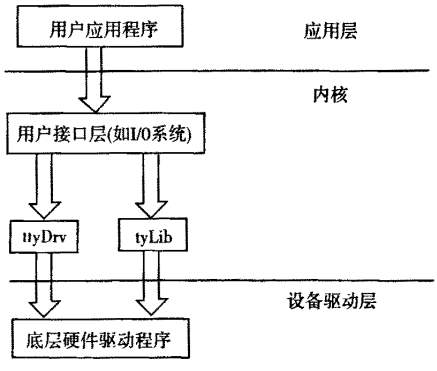

图1 vxWorks 下串口驱动层次结构

串口驱动既然作为底层驱动程序的一个方面，系统不例外的也提供了其与上层的接口层。对于 vxWorks 操作系统而言，这个接口层由 ttyDrv，tyLib 所代表。接口层本质上就是系统内核自定义的一套操作函数和数据结构，底层设备驱动程序将使用其中某些函数和结构与接口层进行交互，接口层经过进一步的处理后，继续与上层进行交互。如上文所述，所谓的上层一

本文于 2008－08－25 收到。

般即指用户接口层,如 I/O 层。之间关系如图 1 所示。

ttyDrv 和 tyLib 被称为虚拟驱动层,二者协同工作共同管理设备驱动层。虚拟驱动层提供的功能主要有:

(1)以字符设备的形式向上层(I/O 系统层)注册其读写和控制函数。注册是在 ttyDrv( )函数中完成的。而 ttyDrv 则调用 iosDrvInstall 完成其注册功能。其对上层注册的读写和控制函数如下:(以下所谓"对应上层 XXX 函数"是指上层 XXX 函数将针对 tty 设备进一步调用此处提供的函数,如上层 read 函数针对 tty 设备将调用 ttyRead 函数进行进一步的处理)

ttyOpen:tty 设备打开函数,对应上层 open 函数。

ttyClose:tty 设备关闭函数,对应上层 close 函数。

tyRead,:tty 设备字符读取函数,对应上层 read 函数。

tyWrite:tty 设备字符发送函数,对应上层 write 函数。

ttyIoctl:tty 设备控制函数,对应上层 ioctl 函数。

(2)向下层(设备驱动层)注册回调函数。作为对下层的接口,针对是读还是写稍有不同。对于写而言,在上层 I/O 层调用 write 函数写设备时,ttyWrite 函数需要主动调用底层驱动程序相关函数发送字符。而对于读而言,ttyRead 函数则直接从该层(虚拟层)读缓冲区中读取字符,当缓冲区为空时,等待(阻塞方式)或者立刻返回错误(非阻塞方式)。而此时读缓冲区的填充是由底层驱动主动进行的,当底层驱动接受到一个字符时,其调用虚拟层提供的机构函数将字符写入缓冲区中以供上层读取。

换句话说,从虚拟层的角度:当写字符时,在将字符填入写缓冲区后,其需要直接调用底层驱动函数发送写缓冲区中的字符。

而当读字符时,则不会直接调用底层驱动函数,而只是检查读缓冲区是否有可读字符。读缓冲区中字符的填入是由底层驱动异步进行的。

从底层驱动的角度:对于写字符时,如果发送完一个字符后,其需要从写缓冲区中(如果非空)取下一个字符继续发送,由于底层驱动不被允许操作虚拟层中的写缓冲区,所以虚拟层即以回调函数形式提供一个函数(tyITx)供底层驱动从写缓冲区中取字符进行发送。当写缓冲区为空时(即目前所有字符已被发送完),底层驱动将停止发送,处于空闲状态。此时如果 I/O 层调用 write 函数,继而到达虚拟层的 ttyWrite 函数,tty-Write 函数在将字符填入写缓冲区后,其就需要启动底层驱动从空闲状态转入发送状态。这种启动即通过调用底层驱动的某个函数完成。而这个函数的地址在底层驱动程序初始化时将被提供给虚拟层。

对于读字符,由于字符到达的随机性,其不能等待上层调用 write 函数,才进行接收,而是只要有字符,就进行接收,并将接收到的字符上传给虚拟层,至于如何对这些字符进行处理将由应用程序决定。此处底层驱动将接收到的字符上传给虚拟层即是通过虚拟层提供的回调函数(tyIRd)完成的。

综上,虚拟层提供给底层驱动的回调函数有两个:①发送:底层驱动从写缓冲区中取字符函数:tyITx;②接收:底层驱动向读缓冲区中写字符函数:tyIRd。

(3)完成其内部初始化(如读写缓冲区分配)和其他各种与硬件无关的辅助功能。

内部初始化主要有:①读写缓冲区分配;②互斥信号量分配;③重要变量和函数指针初始化。

作为底层驱动程序,向上其与虚拟层交互,向下其与具体硬件交互。所以其主要完成的功能也可以分为三个方面:

(1)向上层提供回调函数。底层驱动需要提供一些函数供上层(虚拟驱动层)调用,如启动发送函数,硬件配置函数等等。一般虚拟层为了完成其功能,对于其需要调用的底层功能实现函数会专门以一个数据结构的形式进行封装,底层驱动程序需要对这个数据结构进行初始化后调用虚拟层提供的函数将信息通报给虚拟层。针对串口而言,这个数据结构即为 SIO_DRV_FUNCS 结构,该结构中所有成员均为函数指针,底层驱动必须编写对应的完成相应功能的函数,用这些函数地址对 SIO_DRV_FUNCS 结构进行初始化并通知给虚拟驱动层。

```
/*sioLib.h 文件*/
typedef struct sio_drv_funcs SIO_DRV_FUNCS;
typedef struct sio_chan  /*内核使用 SIO_CHAN 结构表
      示一个串口设备*/
{
  SIO_DRV_FUNCS * pDrvFuncs;
} SIO_CHAN;
struct sio_drv_funcs  /*驱动函数指针定义 */
{
  int (*ioctl)(SIO_CHAN *pSioChan, int cmd, void *
    arg);
  int (*txStartup) (SIO_CHAN *pSioChan);
  int (* callbackInstall) (SIO_CHAN * pSioChan, int
    callbackType,
STATUS (* callback) (void *, ...), void * callback-
    Arg);
  int (*pollInput) (SIO_CHAN *pSioChan, char *in-
    Char);
  int (*pollOutput) (SIO_CHAN *pSioChan, char out-
    Char);
};
```

vxWorks 操作系统对每个串口用一个 SIO_CHAN 结构表示,SIO_DRV_FUNCS 结构作为 SIO_CHAN 结构中一个字段而存在。串口驱动程序必须定义一个 SIO_CHAN 结构并对其进行初始化,实际上即对 SIO_DRV_FUNCS 结构字段进行初始化。在实现上,为了同时保存其它信息,驱动程序定义一个自定制数据结构。如下所示。

```
/*底层设备驱动程序文件:arm926_uart.c*/
  typedef struct _ARM926_CHAN{
/* SIO_CHAN 类型字段必须是自定制结构的第一个
      成员 */
  SIO_CHAN  sio;  /*内核定义的 SIO_CHAN 结构
    类型 */
  ……  /*其他信息定义*/
} ARM926_CHAN;
  /*定义一个自定义 ARM926_CHAN 结构类型变
      量*/
  ARM926_CHAN pChan;
  /*初始化一个 SIO_DRV_FUNCS 结构*/
LOCAL SIO_DRV_FUNCS arm926UartDrvFuncs =
  arm926UartIoctl,
  arm926UartTxStartup
  arm926UartCallbackInstall,
  arm926UartPollInput,
  arm926UartPollOutput,
};
/*在初始化函数中对 SIO_CHAN 结构字段进行初始
    化*/
void arm926UartDevInit(ARM926_CHAN * pChan)
{
  ……
  pChan->sio.pDrvFuncs = &arm926UartDrvFuncs;
  ……
}
```

arm926UartTxStartup 函数即是底层驱动提供的供虚拟层调用的发送启动函数。

arm926UartCallbackInstall 函数也是由底层驱动程序提供,虚拟层调用该函数向底层驱动注册 tyITx,tyIRd 函数地址,即虚拟层提供给底层驱动的其(虚拟层)内部读写缓冲区操作函数,也即前文中所述的虚拟层向下层提供的两个回调函数。之所以提供 arm926UartCallbackInstall 函数,原因在于:①底层驱动并不知道虚拟层提供的回调函数究竟为何?②虚拟层并不清楚底层驱动将自己提供的回调函数安放在何处?

所以虚拟层提供另外一个函数指针 callbackInstall,由底层驱动实现,虚拟层提供回调函数地址作为参数传入,至于这些回调函数地址被保存在何处由底层驱动决定。

此处底层对 callbackInstall 的具体实现为 arm926UartCallbackInstall。对于串口,网口,一般存在两种工作模式:中断和查询。中断模式为硬件正常工作模式,查询模式一般使用在系统调试时。由于查询模式需要明确的由上层进行控制何时进行接收和发送,所以底层驱动程序必须提供对应的函数供上层调用,而上层提供函数指针以获取函数地址,这就是 SIO_DRV_FUNCS 结构中 pollInput,pollOutput 函数指针的目的所在。

(2)控制下层硬件进行字符接收和发送及其他硬件配置工作。底层驱动程序完成的主要功能即控制具体硬件进行字符的接收和发送。在软件实现上,也即读写相关硬件寄存器内容。正常方式下,对于串口,网口这样与外界进行数据交换的设备,一般通过中断方式进行数据的接收和发送,中断方式特别对于数据接收具有重要的意义,因为数据接收时刻对于本设备而言完全不可知,或者说具有相当的随机性,内核或者应

用程序无法预知何时有数据到达,从而正好对其进行读取,所以需要底层驱动程序的异步接收功能:即无论数据何时到达,底层驱动都进行接收,并上传给虚拟驱动层进行缓冲,当应用程序需要读取数据时,直接从缓冲区中读取即可。

而对于数据发送而言,中断同样起着至关重要的作用,通过中断,底层驱动才可获知此次数据是否发送出去,从而继续发送下一波数据。

总而言之,无论是串口驱动还是网口驱动,中断起着枢纽作用,是中断将具体硬件与驱动软件联系起来。所以在软件架构上,无论是串口还是网口,其都是围绕一个中断处理程序而进行,由中断处理程序再调用其他相关处理函数进行进一步的处理。

(3)内部控制结构初始化和其他硬件相关辅助功能。对于驱动程序而言,其必须维护一个自定义数据结构,这个结构中将既包括上层需要的结构信息,也包括对应硬件的具体信息,如硬件控制寄存器基地址,中断号等等。

至于其他辅助功能,包括硬件的统计信息收集和其他一些辅助配置功能。

以上从层次上介绍了虚拟驱动层和设备驱动层的基本功能。了解层次关系对于下文中各层次之间的交互至关重要。下文即从串口初始化开始介绍串口驱动所关联的内核部分。

对于 vxWorks 而言,串口初始化是在 usrInit 函数中(定义在 usrConfig. c 文件中)进行的,调用路径如下:(以下采取"文件名:函数名"形式说明,底层驱动程序文件名为 arm926_uart. c,该文件中定义的函数名均以 arm926 打头,以与内核函数区分)

usrConfig. c:usrInit( )→sysLib. c:sysHwInit( )→sysSerial. c: sysSerialHwInit( )→

arm926_uart. c: arm926UartDevInit( )

sysHwInit 是对所有外围硬件进行初始化的函数,对于串口而言,具体的初始化函数在 sysSerial. c 文件 sysSerialHwInit 函数中进行。而 sysSerialHwInit 函数实现即与具体硬件平台相关,对于某个具体的平台而言,需要对 sysSerialHwInit 函数进行自实现,该实现继续调用具体硬件驱动程序初始化函数,如前文中对于 ARM926EJS 开发环境,其对应初始化函数即为 arm926UartDevInit。

此时进行的初始化只是底层驱动程序的初始化:

(1)完成相关数据结构初始化,包括虚拟层提供的相关结构的初始化从而提供给虚拟层相关回调函数地址。

(2)完成相关硬件寄存器的配置。配置后硬件处于非工作状态,但所有的硬件准备工作已经完成,只是相关工作使能位尚未开启。

此时除了开启硬件正常进行工作,已完成前文所述底层驱动程序所需完成的三个任务中所有方面。

内核相关初始化在 usrRoot 函数中完成。从代码实现看,如果需要在内核中包括串口组件,必须在 config. h 文件中定义如下常量:

```
#define INCLUDE_SERIAL
#define NUM_TTY     N_SIO_CHANNELS     /*串口通道数*/
#define CONSOLE_TTY     0
#define CONSOLE_BAUD_RATE     115200
```

INCLUDE_SERIAL 变量作为控制是否包含串口设备的总开关。对于需要使用串口的平台,必须定义该变量。

NUM_TTY 为串口通道数,这也是平台相关的。

CONSOLE_TTY 表示控制终端使用的通道号,一般定义为 0。

CONSOLE_BAUD_RATE 即控制终端所使用通道的波特率。

usrRoot 函数种对串口组件初始化涉及到的关键函数有三:

(1)ttyDrv( )内核实现。该函数实现在前文中已给出。该函数主要完成的功能是虚拟驱动层向上层(I/O 层)注册读写和控制函数:诸如 ttyWrite,ttyRead,ttyIoctl 等等,以供上层调用。

(2)ttyDevCreate()内核实现。该函数完成对下层相关函数的注册以及内核相关缓冲区和信号量等其它辅助功能初始化工作。

经过以上调用,此时虚拟驱动层已完成前文中所述该层所需完成的三个主要任务,内核串口组件已处于正常工作状态。

(3)sysSerialChanGet()内核提供函数原型,平台相关实现,由底层设备驱动程序员完成编写,与底层串口驱动数据结构相关。该函数的实现读者可参考 templateARM 目录下(或者其它体系结构下对应的目录)sysSerial.c 文件。

到此,只剩下向内核注册中断程序(调用 intConnect 函数完成)和开启硬件使能位。这是在 sysClkConnect 函数中完成的。函数调用路径如下:(采用"文件名:函数名"方式说明,底层串口驱动程序名为 arm926_uart.c)。

usrRoot:sysClkConnect()→sysLib.c:sysHwInit2()→sysSerial.c:sysSerialHwInit2()→

arm926_uart.c: arm926UartDevInit2()

arm926UartDevInit2 函数的实现较为简单:①调用 intConnect 完成中断处理程序的注册。②调用 intEnable 使能该中断。③写具体硬件相关寄存器使能位开启硬件工作。

所有串口相关准备工作完成,串口进行正常的数据接收和发送。

底层驱动程序框架总结如下:

(1)自定义描述设备的数据结构,形式如前文所述。

(2)初始化一个虚拟驱动层定义的 SIO_DRV_FUNCS 结构,如前文所述。

(3)编写中断处理核心函数。具体函数架构可参考 Wind River 提供的串口驱动模版文件。

(4)提供底层驱动初始化函数,修改 sysSerial.c 中相关函数实现,调用这些初始化函数。注意由 sysSerial.c 中相关初始化函数对底层驱动程序中初始化函数的调用通过修改 sysSerial.c 中相关函数实现完成的,而对于 sysSerial.c 文件中初始化函数的调用是由内核自动完成,这从前文的分析中也可看出。所以底层驱动程序员可以而且应该改变 sysSerial.c 文件中相关函数实现,但一定不可修改该文件中函数定义的原型:函数名称和参数类型与个数均不可进行改变。

(5)编写其他具体处理数据接收和发送函数以及其它硬件配置和控制函数,这些函数由中断处理核心函数调用。

结束语

为使内核上层结构独立于具体底层驱动程序实现,几乎所有的操作系统都会在设备驱动层之上提供接口层,一般我们将该层称为虚拟驱动层。作为驱动程序编写者,除了要了解具体硬件功能外,还需要清楚了解底层驱动程序与虚拟驱动层之间的联系。本文以 vxWorks 下串口驱动程序为例,详细阐述了串口驱动程序底层模块和虚拟驱动层模块的功能以及基本实现,并从内核初始化代码进行分析,了解内核模块和底层驱动模块的初始化过程。文章最后附以具体代码给出了一个串口驱动程序的架构。串口驱动程序虽然是所有外围器件驱动程序中较为容易的一类,但要编写出一个稳定健壮的串口驱动程序还是需要清楚地了解其涉及的各个方面,希望本文可以在此方面助你一臂之力。

参 考 文 献

1 Wind River Systems Inc. vxworks kernel programmers guide. 2006.09

2 Wind River Systems Inc. vxworks device driver developers guide. 2006.09

3 Wind River Systems Inc. 内核示例源代码

4 陈智育等. VxWorks 程序开发实践. 北京:人民邮电出版社,2004.

5 http://www.gd-emb.org/detail/id-31347.html, 2008.08

作者简介

曹桂平,男,(1985—),研究生在读,研究方向:高速数据采集,网络通信。

vxWorks串口驱动程序设计

作者： 曹桂平， CAO Guiping
作者单位： 中国科学技术大学,近代物理系,快电子实验室,合肥,230026
刊名： 微计算机应用 ISTIC
英文刊名： MICROCOMPUTER APPLICATIONS
年，卷(期)： 2008,29(11)

参考文献(5条)

1. 查看详情 2008

2. 陈智育 VxWorks程序开发实践 2004

3. Wind River Srstems Inc 内核示例源代码

4. Wind River Systems Inc vxworks device driver developers guide 2006

5. Wind River Systems Inc vxworks kernel programmers guide 2006