VxWorks 下增强型网络驱动程序的设计与实现

Design and Implementation of Enhanced Network Driver on VxWorks

詹特伦　刘伟平　黄红斌　陈舜儿　高智强
（暨南大学 信息科学技术学院电子系 广州 510632）

摘要：VxWorks 下增强型网络驱动程序是一个数据链路层驱动程序，通过 MUX 函数与网络协议层通讯。本文首先从整体上分析 VxWorks 网络驱动程序的原理和结构，然后介绍了基于三星公司 ARM7TDMI 内核的 S3C4510B 增强型网络驱动程序的实现。本文对于 VxWorks 下的同类开发也有一定参考作用。

关键词：VxWorks　S3C4510B　ARM7TDMI　增强型网络驱动程序（END）

1　引言

VxWorks 是由 WRS（Wind River System Inc.）公司开发的一套具有微内核、高性能、可伸缩的实时操作系统，支持广泛的网络通信协议，并能够根据用户的需求进行组合。其开放式的结构和对工业标准的支持使开发者只需做最少的工作即可设计出有效的适合于不同用户要求的系统。VxWorks 的优秀特性为编写应用程序和设备驱动程序提供了极大的便利。本文描述的以太控制器驱动程序的设计方法是在完成了一个嵌入式系统中以太控制器驱动程序基础上产生的。在这个嵌入式系统项目中，所使用的 CPU 是三星公司的 S3C4510B，它是具有 ARM7TDMI 内核的 CPU。作者设计实现了以太网驱动程序并给出了软、硬件的实现方法，同时给出了实际的测试结果。

2　VxWorks 的网络模型[1,2,3]

如图 1 所示，VxWorks 支持两种类型的网络设备驱动程序：VxWorks BSD4.3 网络驱动程序和 VxWorks 可裁剪的增强型网络堆栈（SENS，Scalable Enhanced Networks Stack）。VxWorks BSD 4.3 协议栈符合 BSD4.3 标准，将网络设备驱动程序直接与 IP 协议紧密结合；VxWorks SENS 协议栈则提供了可替换的网络设备驱动程序，而且 Wind River 把专门为使用这个网络堆栈而编写的网络接口设备驱动程序称为增强的网络驱动程序（END，Enhanced Network Driver）。

SENS 模型包括三个部分：协议驱动程序，多元接口（MUX，multiplexer）层，增强的网络驱动程序（END driver）。SENS 模型独立于硬件设备接口，它将网络设备驱动程序细化，使开发者可以专注于驱动程序

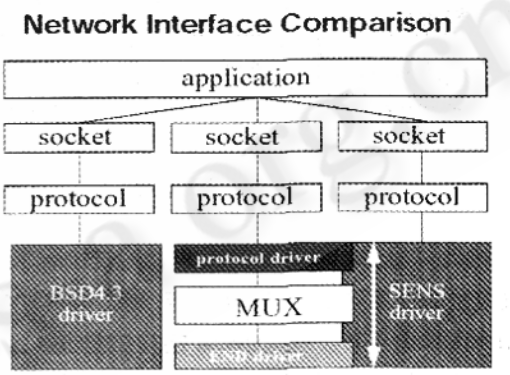

图 1　VxWorks 的网络系统

（END）本身的开发。在驱动程序与上层协议（例如 TCP/IP）之间，SENS 模型提供了协议驱动层。协议驱动层与网络驱动程序之间，SENS 模型则提供了 MUX 层作为注册接口，如图 2 所示，MUX 接口的设计使驱动程序接口独立于网络协议，可以让协议同时与多个网络设备接口交换数据。

通过 MUX 管理网络协议接口和底层驱动程序之间的通信，发送、接收数据的过程也就变得简单了，而不需要象在 BSD 中采用挂接钩子程序的办法解决。

2.1 END 设备驱动程序和 MUX 接口

增强型网络驱动程序(END)是一个专为使用网络堆栈而编写的网络接口驱动程序。网络堆栈的一个特征就是在数据链路层和网络协议层之间使用一个 API。此 API 为一个用来访问网络硬件以获得多种网络协议的多元接口,即 MUX 接口。MUX 接口在 OSI 网络模型中对应于数据链路层和协议层之间的接口并不存在所谓的 MUX 层协议来实现通信,如图 2 所示。MUX 接口只关心自身跟所在协议栈的协议层和数据链路层之间的标准化的通信。由于 MUX 接口的存在,协议或网络驱动程序无需了解相互之间的内部机制。MUX 接口的目的是分解协议和网络驱动程序,从而使他们彼此独立。

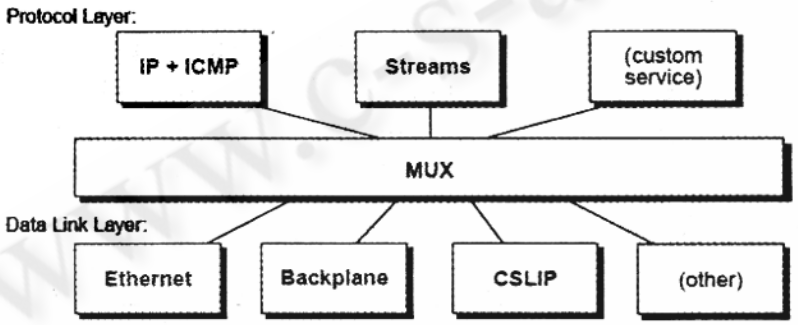

图 2　VxWorks 的 MUX 接口

由此可见,MUX 接口就像桥梁。在协议层一方,协议或服务程序可以通过调用系统函数 muxBind( ) 或者 muxTkBind( ) 来实现与 MUX 接口的通信。这些函数将协议或服务通过 MUX 接口绑定一个网络驱动程序。在绑定的操作中,网络协议或服务程序提供其函数指针给 MUX 接口。

在数据链路层一方,操作系统通过调用 muxDevLoad( ) 来向 MUX 接口添加 END 设备驱动程序。在系统内部,驱动程序装载函数必须创建并初始化几个数据结构,比如 END_OBJ 和 NET_FUNCS。这些数据结构能够为 MUX 接口提供设备的描述以及提供指向驱动程序中标准的与 MUX 接口函数的指针。

2.2 内存管理[4]

VxWorks 定义的三种统一的存储结构分别是 cluster(簇)、clBlk(簇模块)和 mBlk(存储模块)。其中 cluster 是存储数据的最基本单元,长度上是灵活可变的,在驱动程序中一般使用同样长度的簇,即最大的以太包长 1520 字节。这三种结构在驱动程序初始化时产生。当数据向上层传送时,在本层中可以释放已经不再使用的 mBlk 链表,但是这并不表示将 cluster 中的数据释放掉了,上层复制的链表仍然控制着这些数据,直到 clBlk 中的 mBlk 计数为 0 时才真正的将数据占用的簇释放掉。而最后一步是不需要我们来考虑的,VxWorks 提供的系统函数已经很好的处理了,只需要调用即可。创建这三级结构来接收数据报一般要遵循这样四步(接收的数据已经放到 cluster 中):

(1) 调用系统函数 netClBlkGet( ) 从内存池中取 clBlk 结构;

(2) 用系统函数 netClBlJoin( ) 将 clBlk 与存有数据的 cluster 联结起来;

(3) 调用系统函数 netMblkGet( ) 从内存池中取 mBlk 结构;

(4) 调用系统函数 netMblkClJoin( ) 将 mBlk 与 clBlk – > cluster 结构联结起来。

这样,就可以使用这些数据了。数据发送完时要释放内存,只需要调用系统函数 netMblkClChainFree( ) 释放存有数据的 mBlk 链表,当 clBlk 中的 mBlk 计数为 0 时就将数据占用的内存空间释放、归还给系统将来使用。

3　END 驱动程序的设计

3.1 硬件设计

本设计中的硬件核心是三星公司设计的一款基于 ARM7TDMI 核的 S3C4510B 微控制器。该芯片集成度高、功耗低、通用性强,而且它内部集成有以太网控制器,主要应用于基于以太网通信的嵌入式系统中。因此能够缩短网络驱动程序的开发周期以及降低程序设计的负责程度。

S3C4510B 微控制器内部集成了一个以太网控制器(MAC)。该控制器可以运行于 10Mbps/100Mbps 数据速率,半双工/全双工工作模式。但由于没有提供物理层接口,因此,基于 S3C4510 微控制器的系统进行网络通信还需要物理层(PHY)芯片的支持。

我们选用 RTL8201BL 这个 PHY 芯片来实现

S3C4510B 与 100M 以太网的接口，由于 S3C4510B 有两种接口：SNI 和 MII。在系统中我们采用了 MII。接口的选择可以同设置 MII/SNIB 引脚为高电平和正确的设置 ANE，SPEED，DUPLEX 引脚来实现。MII 能够运行在两个频率上，即 25MHz 和 2.5MHz 分别为 100M 以太网和 10M 以太网支持。当数据传输时，MAC 将首先判定 TXEN 信号并且改变 8 位字节数据为 4 位数据，再通过 TXD[0…3]传到物理层上，在 TXEN 信号有效期间，PHY 将通过传输时钟信号 TXCLK 对 TXD[0…3]上的数据进行同步采样；当接受数据时，PHY 将判断接受使能信号，来接受 RXD[0…3]上的数据。

由于 S3C4510B 芯片内部集成了 MAC 控制器，由芯片内部寄存器进行控制，因此我们在使用网络功能时不需要为网卡模块分配外设端口号或映射寄存器地址。虽然在外观上看 S3C4510B 芯片和 PHY 芯片时两个部分，但从开发角度看 S3C4510B 芯片中的 MAC 控制器和 PHY 芯片是一个统一的整体。它们同 S3C4510B 芯片内部通过 BDMA 总线交换数据，由 MAC 控制器和 BDMA（带缓冲的 DMA）控制器统一管理。

3.2 软件设计[4,5]

系统启动时，VxWorks 创建一个系统任务 tUsrRoot，该任务会执行下列操作：

（1）初始化网络任务的工作队列；

（2）创建任务 tNetTask 来处理网络任务工作队列中的项；

（3）调用 muxDevLoad( ) 加载网络驱动程序；

（4）调用 muxDevStart( ) 开始驱动程序。

tUsrRoot 通过查找 endDevTbl 并将表中的相应参数传给 muxDevLoad( )，该函数进而根据传入的参数调用驱动程序入口点 endLoad( )，endLoad( ) 必须处理任何硬件初始化，同时设置 END_OBJ 的大部分成员。当控制从 endLoad( ) 返回到 muxDevLoad( ) 时，函数将设置 END_OBJ 的 receiveRtn 域。在控制从 muxDevLoad( ) 返回后，驱动程序已经加载。接着 tUsrRoot 调用 muxDevStart( )，该函数调用 sysIntConnect( )，ISR 挂接在某个中断向量上，至此，驱动程序就开始工作。

当上层任务需要接收数据时，它将调用 muxReceive( )，但是驱动程序并没有一个叫做 endRecieve( ) 这样的入口点，这时网络有数据过来时，网卡将发出中断，于是 ISR 得到执行，ISR 在对硬件做出相关的设置后，将通过 netJobAdd( ) 提交一个函数指针并放入 tNetTask 的工作队列中。tNetTaks 循环地从工作队列取出函数指针并调用它来执行真正的数据包接收工作。一切都做好以后，该函数将使用一个函数指针将数据包传给 MUX，这个函数指针就是先前由 MUX 设置的 END_OBJ 的 receiveRtn 域。

当上层任务需要发送数据时，它将调用 muxSend( )，muxSend( ) 接着会调用 endSend( ) 向网络发送数据包。上层任务也可以使用 MUX 的其它的入口点，这些入口点均会调用相应的 END 入口点来改变驱动程序的状态。如果调用了 muxDevStop( )，相应的 endStop( ) 被执行，驱动程序将处于非活动状态，但并没有被卸载，我们可以重新调用 muxDevStart( ) 使之处于活动状态。只有在调用了 muxDevUnLoad( ) 时，相应的 endUnLoad( ) 执行后，驱动程序才被真正卸载。以下给出 BSP 中 config.h 和 configNet.h 的关键宏定义和网卡驱动程序中装载函数的部分代码。

```
#define INCLUDE_NETWORK
#define INCLUDE_END
#ifdef INCLUDE_END
#define INCLUDE_SNGKS32C_END /* 根据网络设备的类型，加入指定的网络设备组件（本程序中的网络设备为 sng）*/
#endif
#define INCLUDE_ARP /* 加入所需的网络协议栈组件 */
#define INCLUDE_DNS_RESOLVER
#define INCLUDE_IP_FILTER
#define INCLUDE_FTP_SERVER
#define INCLUDE_HTTP
# define SBCARM7_LOAD_FUNC sngks32cEndLoad /* 定义驱动程序装载函数 */
# define SBCARM7_LOAD_STRING_0 "100:1:1"
END_TBL_ENTRY endDevTbl [ ] = /* 定义 endDevTbl [ ]数组，该数组结构是一张系统网络设备表。它记录了可以使用的网络设备信息，包括数量、相应的驱动程序名称、工作参数等 */
```

其中，endDevTbl [ ]数组的每行开始的数字表示设备的单元数。如第一行规定单元数为 0。每行末尾的 FALSE 表示此行没有被处理。当系统成功装载了一

个驱动程序后，该值将变为 TURE。我们是要将设备驱动程序嵌入到内核中随系统一起启动，因此这里编写时一定不能设为 FALSE。

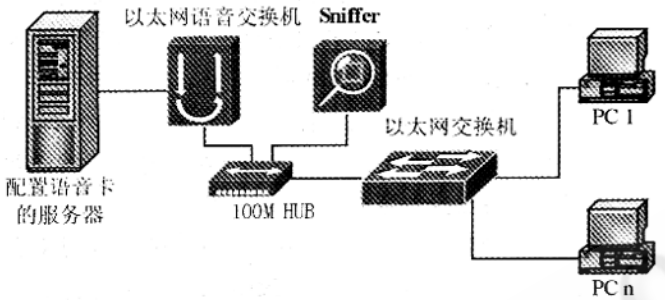

图 3　测试系统网络结构

No.	Source Address	Dest Address	Len (Bytes)	Rel. Time	Delta Time
1	Realtk44726C	Essent888800	144	0 00 00 000	0 000 000
2	Essent888800	Realtk44726C	144	0:00:00.002	0.002.299
3	Realtk44726C	Essent888800	144	0:00:00.015	0.013.147
4	Essent888800	Realtk44726C	144	0:00:00.017	0.002.303
5	Realtk44726C	Essent888800	144	0:00:00.031	0.013.768
6	Essent888800	Realtk44726C	144	0:00:00.033	0.002.312
7	Realtk44726C	Essent888800	144	0:00:00.046	0.012.861
8	Essent888800	Realtk44726C	144	0:00:00.048	0.002.298
9	Realtk44726C	Essent888800	144	0:00:00.062	0.013.272
10	Essent888800	Realtk44726C	144	0:00:00.064	0.002.290
11	Realtk44726C	Essent888800	144	0:00:00.077	0.013.422
12	Essent888800	Realtk44726C	144	0:00:00.080	0.002.279
13	Realtk44726C	Essent888800	144	0:00:00.093	0.013.304
14	Essent888800	Realtk44726C	144	0:00:00.095	0.002.324
15	Realtk44726C	Essent888800	144	0:00:00.109	0.013.300
16	Essent888800	Realtk44726C	144	0:00:00.111	0.002.300
17	Realtk44726C	Essent888800	144	0:00:00.124	0.013.312
18	Essent888800	Realtk44726C	144	0:00:00.127	0.002.340
19	Realtk44726C	Essent888800	144	0:00:00.141	0.013.896

图 4　PBX 与 PC1 之间通信

0	Address	In Packets	In Bytes	Out Packets	Out Bytes	Total Packets	Total Bytes
+	This station	302	44696	302	44696	604	89392
+	Essent888800	302	44696	302	44696	604	89392

图 5　PBX 与 PC1 之间数据包统计

需要注意的是，本 END 程序设计中的数据报发送使用轮循方式。这样，控制器将按照系统设定的频率定期查询发送 FIFO 队列中是否有数据，若有则将之发出；或由系统指定不等下一次轮循到来便将数据发出，遇到系统设定的空发送描述符则表示发送完毕，而数据发送完毕时并不触发中断，继续保持轮循状态。因此，发送函数最后步骤中不使能发送完中断，而且在发送函数的最后要确认本 VxWorks 次发送完毕、允许下一次向发送缓存作 DMA、并释放本次发送占用的缓存给下一次发送使用。

4　实验测试

本文描述的 END 程序的设计方法是在完成了一个嵌入式以太网语音交换机系统基础上产生的，该系统主要是对 PSTN 与局域网之间的语音数据进行处理。因此测试的环境使用了一台配置了语音卡的服务器来模拟 PSTN 的 E1 信号，同时使用 NA 公司的 sniffer 软件进行抓包分析，主要分析双方语音数据的收发情况。测试的网络结构如图 3 所示。

我们用 sniffer 分析了其中一路的语音数据收发情况，即在以太网语音交换机（MAC 地址：00－A0－88－88－88－00）同 PC1（MAC 地址：00－E0－4C－44－72－6C）之间抓取了 600 个数据包，如图 4 所示。

从图中可以看到 PBX 往 PC1 方向的语音数据和 PC1 往 PBX 之间的语音数据速度基本一致，而且保持一收一发的均衡关系，完全符合测试的理想要求。图 5 给出了数据包的统计情况。

5　结束语

本文分析了基于 S3C4510B 在 VxWorks 操作系统上 END 驱动程序的实现，并在实验测试的基础上验证了其可行性。由于 MAC 的内部差异，其他网卡在初始化、接收数据以及发送数据方面会有不同，但是在其他方面，都可以参考本驱动程序的实现。

参考文献

1　WindRiver Inc. “VxWorks Programmer's Guide”
2　WindRiver Inc. “VxWorks Network Programmer's Guide”
3　WindRiver Inc. “Tornado API Reference”
4　全成斌、任秀丽等，“嵌入式系统以太网驱动程序的设计方法”，小型微型计算机系统 2002.9。
5　吕佳彦、杨志义等，“VxWorks 增强型网络驱动程序 END 的分析与实现”，计算机应用研究，2005.4