

基于 VxWorks 的网卡驱动程序开发

张宇坤,袁冬莉,黄鑫鑫,姚建勋

(西北工业大学 自动化学院,陕西 西安 710129)

摘要: VxWorks 是一种运行在目标机上的可剪裁的高性能嵌入式实时操作系统,集成了标准 TCP/IP 网络功能。但其支持的网卡型号有限,限制了网络资源的使用,网络驱动开发就显出其重要性。VxWorks 特有的增强网络驱动(END)包含有一个多路复用层模块(MUX)将驱动程序细化,这个模块赋予了驱动独立性,使得开发者专注于驱动本身的发展,因此选择使用这种驱动模式为 intel82567 网卡成功实现驱动。这种特有的驱动模式在应用的过程中确实体现出其高效性,不用关心底层的转换和协议处理,对需要进行短周期 END 型网络驱动的开发有参考价值。

关键词: VxWorks;END 网络驱动;intel82567

中图分类号:TP316.2

文献标识码:A

文章编号:1673-629X(2013)01-0018-03

doi:10.3969/j.issn.1673-629X.2013.01.005

Programming of Network Adapter Driver Based on VxWorks

ZHANG Yu-kun, YUAN Dong-li, HUANG Xin-xin, YAO Jian-xun

(School of Automation, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: VxWorks is a scalable embedded real-time operating system with great performance, which is run in the target. And the standard TCP/IP protocol is integrated within the system. As only a few network adapters support the system, which limits the use of cyber source, the designing of network driver in VxWorks systems becomes vital. The MUX included in END is able to refine the driver program, which helps the designer to focus on developing the driver and gives the drive independence, so use this driving mode for successful implementation of intel82567 NIC driver. This special driving mode reflects the efficiency in the application process with no care about the underlying drive conversion and protocol processing, and may have a reference value for the short period designing of END.

Key words: VxWorks; enhanced network driver; intel82567

0 引言

VxWorks 操作系统有着良好的可靠性和卓越的实时性,经常应用在航天航空等高精尖技术的领域中^[1],因此将其作为航空半物理仿真软件平台。对于平台的开发主要还是在 BSP(板级支持包)的配置和驱动程序的编写上,分析了 VxWorks 特有的 END 型网络驱动,用此方法解决实现平台硬件中未有支持的 intel82567 驱动程序。

1 VxWorks 网络驱动结构

VxWorks 是第一个集成标准 TCP/IP 网络功能的实时操作系统。支持两种形式的网络驱动:一种是

BSD 驱动,支持通用的 BSD4.4 网络、API 结构等,和大多数的 BSD 网络的驱动类似;另一种是 END 网络驱动,是 VxWorks 独有的,叫做增强型网络模型。它主要的特点就是加了一个 MUX 模块,管理所有的 END 设备,使得协议与硬件无关。END 驱动在底层也要转换成 BSD 的形式^[1,2]。

2 VxWorks 网络驱动流程

END 设备驱动程序的装载分三个步骤:指定 END 设备;装载设备;启动设备^[3]。

指定 END 设备要通过修改板级支持包(BSP)完成。BSP 中的 configNet.h 文件中定义着驱动程序入口 END_TBL_ENTRY 结构的数组 endDevTbl[],该数组描述了系统中的所有网络设备的装载函数入口点及相关参数。在 config.h 文件中加#define INCLUDE_END 以使系统准备初始化 MUX 并通过 MUX 装载网络设备驱动程序^[4,5]。

装载及启动 END 设备驱动程序通过系统调用完成。在系统引导过程中,将执行 tUsrRoot 来完成初始

收稿日期:2012-04-19;修回日期:2012-07-24

基金项目:陕西省自然科学基金计划(2011QG8005)

作者简介:张宇坤(1988-),男,陕西铜川人,硕士研究生,研究方向为现代控制理论及应用和机载机电系统控制;袁冬莉,硕士生导师,博士,副教授,主要从事控制系统理论与应用、飞行控制系统设计等方向的研究。

化网络任务的工作队列、创建 tNetTask 任务来处理网络任务工作队列的条目、调用 muxDevLoad 和 muxDevStart 装载和启动指定的网络驱动。并根据 endDevTbl[] 中的定义逐一调用具体设备的装载函数和设备启动函数,至此 MUX 的初始化以及 END 驱动程序的初始化工作已经完成。但是网络设备还不可用,还需要调用 muxBind 将协议绑定到指定的设备上^[6]。图 1 为网络初始化顺序^[7,8]。



图 1 网络初始化顺序

3 intel 82567 驱动具体实现

Intel 82567 是一个千兆网卡,并没有 VxWorks 的驱动支持。从零开始开发网卡驱动程序比较费时,此时通过查看 intel 82567 网卡的 datasheet 发现有个相似的千兆网卡 intel 82543 是有 VxWorks 驱动支持的,两个型号的网卡有相同的工作模式,可以用 intel 82543 的驱动为基础进行修改,用此方法可以大大缩短开发周期。

前面对 VxWorks 的 END 型网络驱动做了简单说明,接下来按照 END 驱动加载流程对 intel82567 驱动进行具体分析。将网卡驱动具体实现分为几个步骤:驱动头文件修改,网络定义和 PCI 初始化,xxLoad 函数修改,xxStart 函数实现。

3.1 头文件修改

对两个网卡的 datasheet 进行仔细分析,得到不同之处:大部分为寄存器定义,以及少部分协议定义和宏定义,对应修改驱动头文件。下面给出头文件部分修改:

```
#define PRO1000_567_BOARD 0x10cb
/* 82567v MAC */

#define GEI_END_FORCE_FLUSH_CACHE 0x1000 /*
force flush data cache */

#define GEI_END_FORCE_INVALIDATE_CACHE 0x2000 /*
* force flush data cache */

#define IP_HDR_CKSUM_OFFSET 10
/* Offset within IP header */

#define TCP_HDR_CKSUM_OFFSET 16
/* Offset within TCP header */

#define UDP_HDR_CKSUM_OFFSET 6
/* Offset within UDP header */
```

```
#define INTEL_82567GC_EERD 0x14
.....
```

还有协议定义、EERD 寄存器、中断寄存器的扩展控制、发送接收描述符定义、EEPROM reload done、扩展控制寄存器、发送控制寄存器。注意统一定义格式,保证程序通用性。

3.2 网络接口和 PCI 初始化

BSP 包中的 config.h 文件中有着 END 网络驱动定义,在此添加网卡定义,以最初声明网卡。在 config-Net.h 文件中定义着网卡装载函数的所有接口函数和装载参数的数组 endDevTbl[],对驱动的正常调用有重要作用。以 intel 82567 为例给出代码说明:

```
#ifndef INCLUDE_GEI82567_END
#define GEI82567_LOAD_FUNC sysGei82567EndLoad
#define GEI82567_BUFF_LOAN TRUE
#define GEI82567_LOAD_STR ""
IMPORT END_OBJ * GEI82567_LOAD_FUNC (char *,
void *);
#endif /* 包含 intel82567 网卡 */
END_TBL_ENTRY endDevTbl[] =
{.....
#ifdef INCLUDE_GEI82567_END
{0, GEI82567_LOAD_FUNC, GEI82567_LOAD_STR,
GEI82567_BUFF_LOAN, NULL, FALSE},
#endif
.....}; /* 程序入口数组定义 */
```

sysLib.c 中有硬件网卡设备的 PCI 初始化函数声明,在系统硬件初始化的时候运行,查找到设备并分配相应的资源。写 PCI 初始化函数的时候有个方法,先引导系统到 DOS 环境下,用 RU 小工具来查看硬件的六个基地址寄存器,以确定设备所需要的存储空间和 I/O 空间及相应寄存器号,在编写 PCI 初始化程序时有目的性,不必做循环查找函数,影响系统启动时间。另外还要确定硬件类型,为之后装载函数做参数准备。

sysLib.c 中 intel 82567 PCI 初始化函数声明具体代码说明:

```
#ifndef INCLUDE_GEI82567_END
IMPORT STATUS sys567PciInit (void);
#endif /* 声明 PCI 初始化函数 */
#ifdef INCLUDE_GEI82567_END
#include "sysGei82567End.c"
#endif /* 声明代码位置 */
sysHwInit 函数由系统调用初始化系统硬件,在此函数中的 PCI 初始化代码后添加如下代码:
#ifdef INCLUDE_GEI82567_END
sys567PciInit (void);
#endif /* 执行网卡 PCI 初始化 */
```

3.3 装载函数

前面的 endDevTbl[] 数组中定义了驱动装载函数

sysGei82567EndLoad,它会被 muxDevLoad 函数调用两次。第一次返回设备名称,第二次调用为设备分配结构体 END_DEVICE 和内存,设置基寄存器,并利用 PCI 初始化程序所确定的硬件类型进行板特定硬件初始化。以 intel 82567 为例,要在 EEPROM 的校验程序上加上限制条件,避免 ICH8(南桥芯片)访问 EEPROM:

```
if (pRsrc->boardType == PRO1000_546_BOARD)
{
    .....
    EEPROM 校验程序
    .....
    并重新设计校验程序的读字函数
    sys567eepromReadWord(int unit,UINT32 index)
    {
        int count;
        UINT16 val;  UINT32 tmp;
        tmp = EERD_START_BIT | (index << 2);
        GEI_SYS_WRITE_REG ( unit, INTEL_82567GC_EERD,
        tmp);
        for ( count = 0; count < 10000; count++)
        {
            taskDelay(1);
            tmp=GEI_SYS_READ_REG(unit,INTEL_82567GC_EERD);
            if (tmp & EERD_567_DONE_BIT)break;
        }
        if (count == 10000)
        {
            printf("gei%d: EEPROM read timed out\n", unit);return
(0);
        }
        tmp=GEI_SYS_READ_REG(unit,INTEL_82567GC_EERD);
        val = (tmp >> 16) & 0xFFFF;
        return (val);
    }
}
```

装载函数接着建立基寄存器、设备所需结构体,分配内存,关芯片中断,关系统中断,关收发操作(在启动函数中会再次打开),释放分配的内存。装载函数至此执行完毕。

3.4 启动函数

设备装载完毕后,网卡准备工作完成,需要用启动函数 gei82567EndStart 来连接系统中断,初始化收发器,并使能网卡工作。

网卡工作在 OSI(开放式系统互连)的最后两层,即物理层(PHY)和数据链路层(MAC)。PCI 连接 MAC,MAC 连接 PHY,PHY 连接网线,构成网卡基本工作线路。每一层除了本层的特定功能外,都需要向上层提供接口^[9,10]。而设备驱动就是处理硬件与上次协议之间的接口程序,因此,涉及 PHY 操作的启动函数是驱动中最困难的部分^[11]。

系统通过 muxDevStart 调用启动函数。先关系统中断,初始化硬件并建立和 PHY 设备的连接,为千兆媒体独立接口做连接的准备工作,接着为收发器分配

工作所需结构体,配置网卡工作模式,连接系统中断,最后使能收发操作,至此网卡启动,系统还会继续为指定设备绑定协议,重新编写收发函数后,网卡就能正常工作了。启动函数中关于 PHY 的部分容易出错,在此对于 PHY 的具体操作可以借鉴 linux 驱动,先对 PHY 操作有些基础的了解,然后编写 VxWorks 下的 PHY 操作函数就稍微容易一点,实践中也证明这一方法行之有效。以下给出 PHY 写函数示例:

```
LOCAL STATUS gei82567PhyWrite(
END_DEVICE * pDrvCtrl, /* driver structure */
UINT8 phyAddr, /* PHY bus number */
UINT8 phyReg, /* PHY's register to write */
UINT16 data /* data to write */) {
    int count = 10; /* counter */
    volatile UINT32 mdicRegVal;
    mdicRegVal = (MDI_WRITE_BIT | phyAddr << MDI_PHY_
SHIFT | phyReg << MDI_REG_SHIFT | data);
    GEI_WRITE_REG(INTEL_82567GC_MDI,mdicRegVal;
    gei82567Delay (pDrvCtrl, 32000);
    /* wait 32 microseconds */
    while (count-- /* wait max 96 microseconds */) {
        GEI_READ_REG(INTEL_82567GC_MDI,mdicRegVal;
        if (mdicRegVal & MDI_READY_BIT)break;
        gei82567Delay (pDrvCtrl, 16000);
    }
    if ((mdicRegVal & (MDI_READY_BIT | MDI_ERR_BIT)) !=
MDI_READY_BIT)
        {
            DRV_LOG (DRV_DEBUG_LOAD, "Error: MII write PhyA-
ddr=%d, phyReg=%d\n...",phyAddr, phyReg, 3, 4, 5, 6);
        }
}
```

做完所有修改就可编译 BSP 得到 VxWorks 镜像,启动系统对驱动进行循环测试,直到网卡正常工作。

4 结束语

本驱动是通过修改有相似工作模式的网卡驱动来实现的,开发周期较短。在修改的过程中要对修改部分进行测试,在此介绍下测试经验:在网卡 PCI 初始化程序中不能使用 printf() 函数,因为此时 PC 控制台还没有初始化,可以编写小测试函数代替^[12]。具体测试方法:编译 BSP 建立一个可引导工程,并添加 PING client、Telnet server 和 FTP server 组件,编译 VxWorks 镜像,进行 ping 测试、ping 大包测试、Telnet 测试和 FTP 测试,验证网卡驱动的正确性。在实际测试中,ping 测试无丢包,延迟时间正常,Telnet 和 FTP 可以正常登陆并传输数据,验证驱动的正确性。此驱动开发的流程对 END 型网卡,尤其是需要短周期开发的网卡有借鉴意义。

(下转第 24 页)

(2)将队列长度和阈值进行比较,若队列长度是 40 及以上,那么 counter+1,若队列长度是 10 及以下,则 counter 值-1。

(3)如果 counter 值是 15,建立生成标签,增加一个服务器连接,并重置 counter 和标签值。

(4)如果 counter 值是-15,建立删除标签,删除一个服务器连接,并重置 counter 和标签值。

(5)返回 1。

在该实验中,如果队列长度是 40 及以上,便认为服务器已经过载,需要减少连接数。同理,若队列长度是 10 及以下,认为服务器还有很大的负载量,可以增加一些连接。最大连接数设为 10,最小连接数设为 5。

4.3 实验结果

使用 TPC-C 标准,2 小时内 TPM 吞吐量和平均应答时间如表 2 所示。在本实验中使用了 40 个数据仓库,400 个客户连接来测试,同时将服务器连接数分为固定和动态变化两种,固定的设置为 20,10 和 5。初始阶段服务器连接数设置为 20,并且在运行时比较固定连接数和动态改变连接数之间性能的不同。从结果来看,吞吐量提高了将近 4%,应答时间减少了大约 10%。

表 2 实验结果

连接数	20(固定)	10(固定)	5(固定)	动态变化
吞吐量(bps)	530.4	530.6	529.2	553.1
平均响应时间(秒)	5.98	6.03	6.01	5.39

5 结束语

文中通过改进 TPM 来提升 Postgresql 的性能,检

(上接第 20 页)

参考文献:

[1] 陈智宇,温彦君,陈 琪. VxWorks 程序开发实践[M]. 北京:人民邮电出版社,2004.

[2] 孔祥营,张保山,俞烈彬. VxWorks 驱动及分布式编程[M]. 北京:中国电力出版社,2007.

[3] 周启平,张 杨,吴 琼. VxWorks 开发指南与 Tornado 实用手册[M]. 北京:中国电力出版社,2004.

[4] 周启平,张 杨. VxWorks 下设备驱动程序及 BSP 开发指南[M]. 北京:中国电力出版社,2004.

[5] 曹桂平. VxWorks 设备驱动开发详解[M]. 北京:电子工业出版社,2011.

[6] 张 杨,于银涛. VxWorks 内核、设备驱动与 BSP 开发详解

测标准是 TPC-C,解决了现存 TPM 中三层数据库系统的问题,增加了连接数的动态变化和监测服务器状态的功能。

参考文献:

[1] 许宏松. PostgreSQL7 数据库开发指南[M]. 北京:机械工业出版社,2001:1-38.

[2] 林河水,程 伟,孙玉芳. PostgreSQL 存储管理机制研究[J]. 计算机科学,2004(12):76-80.

[3] 古 锐,元 伟,叶晓俊. 表空间存储策略 PostgreSQL 中的研究与实现[J]. 计算机工程,2006(16):38-40.

[4] Stonebraker M. The Design of the Postgres Storage System [C]//Proceedings of 13th International Conference on Very Large Data Bases. Brighton,England:[s. n.],1987.

[5] 陈文星,付继宗. Linux 下数据库 PostgreSQL 分析与应用[J]. 电脑开发与应用,2006(11):56-57.

[6] 吴 亮. 基于 PostgreSQL 的海量数据存储管理[D]. 长沙:中南大学,2005.

[7] 王 博,李 波,高振铁. 基于 TPM 的嵌入式可信计算平台设计[J]. 单片机与嵌入式系统应用,2011,11(1):13-16.

[8] 刘长浩,孙玉芳. PostgreSQL 请求优化机制研究[J]. 计算机科学,2005,32(4):163-167.

[9] 郭龙江,李金宝. PostgreSql 分析器研究[J]. 黑龙江大学自然科学学报,2001(4):50-52.

[10] Pachev S. Understanding MySQL Internals[M]. [s. l.]: O'Reilly Media,Inc. ,2007.

[11] 胡巧巧,王建名,叶晓俊. PostgreSQL 数据库预取算法研究[J]. 计算机科学,2006(6):138-139.

[12] 朱 江,沈庆国. 开放源码数据库 PostgreSQL 的特点及其应用实例[J]. 军事通信技术,2003(2):59-62.

[M]. 北京:人民邮电出版社,2011.

[7] VxWorks Reference Manual[M]. USA:Wind River System Inc,1997.

[8] VxWorks 5. 5 Migration Guide 6. 6[M]. USA:Wind River System Inc,2007.

[9] 周启平,张 杨. VxWorks 程序员速查手册[M]. 北京:机械工业出版社,2005.

[10] 李方敏. VxWorks 高级程序设计[M]. 北京:清华大学出版社,2004.

[11] 罗国庆. VxWorks 与嵌入式软件开发[M]. 北京:机械工业出版社,2003.

[12] Wind River. VxWorks Programmer's Guide[M]. USA:Wind River System Inc,1997.

基于 VxWorks 的网卡驱动程序开发

作者: [张宇坤](#), [袁冬莉](#), [黄鑫鑫](#), [姚建勋](#)
作者单位: [西北工业大学 自动化学院, 陕西 西安 710129](#)
刊名: [计算机技术与发展](#)
英文刊名: [Computer Technology and Development](#)
年, 卷(期): 2013(1)

本文链接: http://d.g.wanfangdata.com.cn/Periodical_wjtz201301007.aspx