

可扩展固件接口的网络设备驱动

董丽丽, 张富凯, 廖福燕

(西安建筑科技大学 信息与控制工程学院, 陕西 西安 710055)

摘要: 提出了基于EFI的网络设备驱动设计方法. 通过研究EFI固有的驱动模型, 分析通用网络驱动接口和简单网络协议, 为网络设备驱动的实现提供了基本的接口和服务. 分析了EFI网络设备的结构和组成, 根据其特点, 提出了3层结构模型的EFI网络设备驱动设计方案, 阐述了如何以EFI驱动形式实现EFI网络设备驱动. 通过设计并实现网络设备的基本功能, 提出了EFI网络设备驱动的基本流程.

关键词: 可扩展固件接口; 驱动模型; 网络设备; 驱动程序

中图分类号: TP316; TP393

文献标识码: A

文章编号: 1000-7180(2011)05-0152-05

Network Device Driver of Extensible Firmware Interface

DONG Li-li, ZHANG Fu-kai, LIAO Fu-yan

(School of Information & Control Engineering, Xi'an University of Architecture & Technology, Xi'an 710055, China)

Abstract: A design method of network device driver based on EFI is presented. To begin with, the inherent EFI driver model is studied and the Universal Network Driver Interface and Simple Network Protocol are analyzed. It provides the basic interface and services for the implement of network device drivers. It analyses the structure and the component of EFI network device and puts forward a 3-layer model for EFI network device driver scheme according to the features of EFI network device. How to implement EFI network device driver in the form of EFI driver is introduced. Finally, by designing and implementing the basic functions of network device, a basic process of network device drivers based on EFI is presented.

Key words: extensible firmware interface (EFI); driver model; network device; driver

1 引言

可扩展固件接口(Extensible Firmware Interface, EFI)是由模块化的C语言, 藉由参数堆叠传递, 并以动态连结形式架构的简易型操作系统, 它定义了操作系统与平台固件之间的接口规范. EFI是全新的下一代BIOS架构, 它规定了一个统一的驱动程序^[1], 这些驱动程序是专门应用于EFI系统下的机器指令, 使得不同厂商开发的代码可以一起工作.

网络设备驱动已被广泛地应用于各种Windows和Linux平台. 但是, 现有操作系统中网络设备驱动的开发在EFI上并不能顺利运用, 这主要是

由于EFI只是提供了统一的驱动模型, 并没有实现具体的网络设备驱动. 和目前常见的操作系统网络设备驱动相比, EFI缺少网络设备开发支持的工具, 大大限制了EFI的应用, 因此, 有必要根据EFI的实际情况进行专门的网络设备驱动设计.

2 EFI驱动模型

EFI下的网络设备通过一个特定的驱动被加载到系统, 因此, 它应该遵循EFI驱动模型, 以实现驱动动态加载的设计目的. 每个符合EFI驱动模型的驱动都要实现EFI驱动绑定协议EFI_DRIVER_BINDING_PROTOCOL^[2]. 该协议是允许驱动程序和控制器被管理的最主要的部分, 它提供了检测、启

动和停止驱动程序的服务,是 EFI 能够管理硬件的前提.要实现该协议,必须实现如下三个接口函数^[3]:

(1) Supported 接口:检测给定的驱动程序是否支持相应的控制器.对于 EFI 网络设备驱动程序,当启动服务连接到控制器和映像句柄时,该接口会检测驱动程序是否能够加载到实现了网络 I/O 协议的控制器的上.

(2) Start 接口:启动控制器上的驱动程序.在 EFI 网络设备驱动中,该接口的主要任务是初始化控制器所对应的网络设备,为网络设备分配其所需要全局数据结构;同时,它将安装 EFI 服务绑定协议 EFI_SERVICE_BINDING_PROTOCOL 到驱动映像处理上,以供使用者使用不同配置的该驱动实例. EFI 服务绑定协议的接口实现如下^[4]:

```
typedef struct
_EFI_SERVICE_BINDING_PROTOCOL{
EFI_SERVICE_BINDING_CREATE_CHILD CreateChild;
EFI_SERVICE_BINDING_DESTROY_CHILD DestroyChild;
}EFI_SERVICE_BINDING_PROTOCOL.
```

(3) Stop 接口:停止驱动程序.该接口实际上是 Start 接口的反过程,它的主要任务是释放 Start 接口中分配的资源,关闭已经打开的协议.其中,关闭协议的顺序应该和 Start 接口中打开协议的顺序相反.

3 通用网络驱动接口和简单网络协议

通用网络驱动接口(Universal Network Driver Interface)和简单网络协议(Simple Network Protocol)是 EFI 网络设备驱动需要实现的两个重要部分.通用网络驱动接口用来生成简单网络协议,并获取底层网络接口的类型和版本信息.简单网络协议给网络适配器提供了一个信息包一级的接口,通过这个接口便可以获得初始化网络接口、发送包、接收包和关闭网络接口等服务.

通用网络驱动接口提供了大量的函数接口,对于上层使用者来说,通过这些接口屏蔽了具体设备的差异.当初始化通用网络驱动接口时,需要打开它所依赖的 PCI I/O Protocol 和 Device Path Protocol,之后利用 PCI 总线的 IO 功能读取网络设备的信息,并初始化全局控制结构.通用网络驱动接口定义如下:

```
VOID UNDI_GetState();
```

```
VOID UNDI_Start();
VOID UNDI_Stop();
VOID UNDI_GetInitInfo();
VOID UNDI_GetConfigInfo();
VOID UNDI_Initialize();
VOID UNDI_Reset();
VOID UNDI_Shutdown();
VOID UNDI_Interrupt();
VOID UNDI_RecFilter();
VOID UNDI_StnAddr();
VOID UNDI_Statistics();
VOID UNDI_Ip2Mac();
VOID UNDI_NvData();
VOID UNDI_Status();
VOID UNDI_FillHeader();
VOID UNDI_Transmit();
VOID UNDI_Receive();
```

简单网络协议提供了 packet 层面的网络接口.简单网络协议是通用网络驱动接口上很薄的一层,它了解通用网络驱动接口的 CDB(Command Descriptor Block)结构,主要负责把上层调用封装成为通用网络驱动接口定义的 CDB 格式来操作网络设备.简单网络协议是全局唯一的实例,不会因配置不同而生成不同的实例,因此它不支持 EFI_SERVICE_BINDING_PROTOCOL.简单网络协议定义如下:

```
typedef struct
_EFI_SIMPLE_NETWORK_PROTOCOL_ {
UINT64 Revision;
EFI_SIMPLE_NETWORK_START Start;
EFI_SIMPLE_NETWORK_STOP Stop;
EFI_SIMPLE_NETWORK_INITIALIZE Initialize;
EFI_SIMPLE_NETWORK_RESET Reset;
EFI_SIMPLE_NETWORK_SHUTDOWN Shutdown;
EFI_SIMPLE_NETWORK_RECEIVE_FILTERS ReceiveFilters;
EFI_SIMPLE_NETWORK_STATION_ADDRESS StationAddress;
EFI_SIMPLE_NETWORK_STATISTICS Statistics;
EFI_SIMPLE_NETWORK_MCAST_IP_TO_MAC MCastIpToMac;
EFI_SIMPLE_NETWORK_NVDATA NvData;
EFI_SIMPLE_NETWORK_GET_STATUS GetStatus;
EFI_SIMPLE_NETWORK_TRANSMIT Transmit;
EFI_SIMPLE_NETWORK_RECEIVE Receive;
EFI_EVENT WaitForPacket;
EFI_SIMPLE_NETWORK_MODE * Mode;
```

} EFL_SIMPLE_NETWORK_PROTOCOL.

4 EFI 网络设备驱动设计方案

4.1 系统结构

EFI 网络设备驱动设计的目的是与网络中远程平台建立数据连接. 由于 EFI 并没有实现多任务的管理和调度, 因此忽略了网络设备的一些高级特性, 所支持的设备操作, 主要是通用网络驱动接口规范所定义的接口.

根据网络设备的特点, 可把整个驱动体系分为 3 层: 协议栈、网络设备驱动功能层、设备媒介层. 系统结构如图 1 所示.

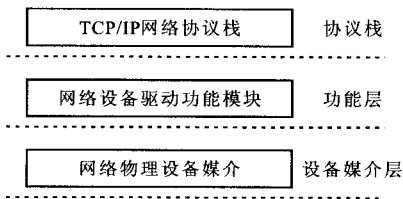


图 1 系统结构

(1) TCP/IP 网络协议栈 (TCP/IP Network Stack) 是设备启动管理实现的一个端口, 它包括了大量的 Internet 国际互联网协议和服务. 该层的主要任务是调用网络设备各个模块的接口, 实现协议栈所定义的接口. TCP/IP 网络协议栈组织结构如图 2 所示.

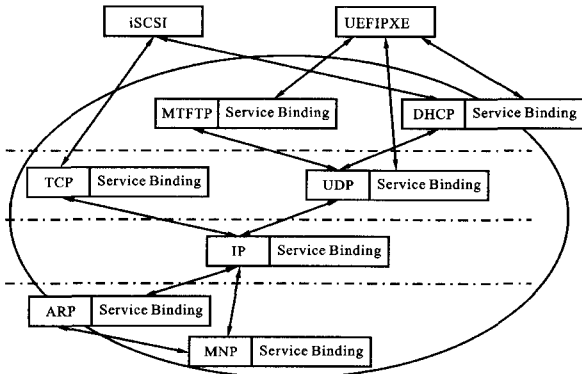


图 2 TCP/IP 网络协议栈

(2) 功能层具体处理网络设备驱动逻辑. 它包括通用网络驱动接口和简单网络协议, 需要处理网络设备定义的各种设备数据结构. 由于功能层处理的数据逻辑比较复杂, 因此, 该层的实现是整个网络设备驱动设计的重点和难点.

(3) 网络物理设备媒介层提供具体的物理设备, 有网卡、交换机、集线器、Modem 防火墙等.

4.2 网络设备驱动功能模块

网络设备驱动功能模块, 主要完成对网络设备

所定义的主要函数和数据结构的操作. 在 EFI 的启动阶段中, 设备中断(时钟中断除外)是被关闭的, 系统通过轮询方式获知数据的状态, 即数据是否到达网络设备和数据是否已经被网络设备成功发送. 通用网络驱动接口和简单网络协议为网络数据的传输提供了基本的信息和服务, 因此, 设备驱动的实现得到了简化. 根据网络设备驱动系统结构, 可以把设备驱动分为几个设计部分: 设备驱动注册, 设备驱动注销, 网络设备初始化, 数据发送流程, 数据接收流程, 打开网络设备和释放网络设备. 这些部分完成各种网络数据结构的操作. 其中, 主要对数据发送和数据接收这两个部分进行处理. 网络设备驱动功能模块如图 3 所示.



图 3 网络设备驱动功能模块

数据包的发送和接收^[5]是网络设备驱动要实现的核心部分. 在 EFI 中, 简单网络协议提供了基本的网络抽象接口, 如 Transmit 和 Receive 等, 并采取轮询的方式实现网络数据传输. 数据包发送流程的功能函数如下:

```
typedef
EFI_STATUS (EFI_API * EFL_SIMPLE_NETWORK_
TRANSMIT) {
    IN EFL_SIMPLE_NETWORK_PROTOCOL * This
    IN UINTN HeaderSize,
    IN UINTN BufferSize,
    IN VOID * Buffer,
    IN EFL_MAC_ADDRESS * SrcAddr OPTIONAL,
    IN EFL_MAC_ADDRESS * DestAddr OPTIONAL,
    IN UINT16 * Protocol OPTIONAL
}
```

发送数据时^[6], 上层调用 Transmit() 函数, 该操作代表了一次发送需求. 发送函数 Transmit() 填充发包事件 Event 中的数据, 并将其放入到 Transmit Packet Queue 队列中. 然后, 系统读取 Packet 空间内的数据, 确定其发送标志, 即设备处于准备发送状态. 此时, 系统将要发送的数据循环写入发送包中. 完成这些操作后, 网卡会将数据组织为以太网帧并添加填充位和校验信息, 然后将数据转化为比特流传送到网络媒介. 图 4 是数据包发送流程示意图.

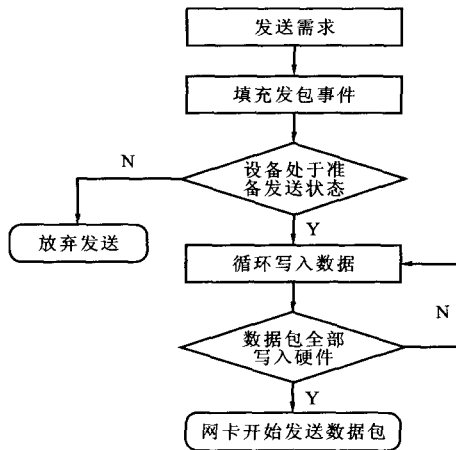


图4 数据包发送流程

对于接收来说,每个接收实例中有两个队列:Received Queue 和 Delivered Queue.前者放置从网络设备上拷贝的数据,而后者放置已经找到上层交付者的数据.当网卡接收到一帧数据时,就会触发接收事件.数据包接收流程的功能函数如下:

```

typedef
EFL_STATUS (EFI_API * EFL_SIMPLE_NETWORK_RECEIVE) {
    IN EFL_SIMPLE_NETWORK_PROTOCOL * This
    OUT UINTN * HeaderSize OPTIONAL,
    IN OUT UINTN * BufferSize,
    OUT VOID * Buffer,
    OUT EFL_MAC_ADDRESS * SrcAddr OPTIONAL,
    OUT EFL_MAC_ADDRESS * DestAddr OPTIONAL,
    OUT UINT16 * Protocol OPTIONAL
}
  
```

接收数据时,上层调用 Receive()函数查询接收标志.一次 Receive()调用在返回的过程中,用来检测某个 Received Queue 上是否已经存在收到的数据,若是,系统则将数据转移到 Delivered Queue 上,并依次读取接收到的状态标志和数据.判断数据包读取完毕后,驱动程序进行数据包的处理并将数据向上层传递.图5是数据包接收流程示意图.

其他各个部分相互协调共同完成数据传输的具体操作.网络设备驱动的注册与注销使用成对出现的 Register()和 Unregister()函数完成,设备驱动注册需要在网络设备驱动程序的模块加载函数中进行,而设备驱动的注销则需要在模块卸载函数中完成.网络设备的初始化需要做多方面的准备工作.检查网络设备硬件是否存在,如果存在,则检测设备所使用的硬件资源;进行软件接口上的准备工作,分配适当的数据结构并对其数据和函数指针成员赋值;

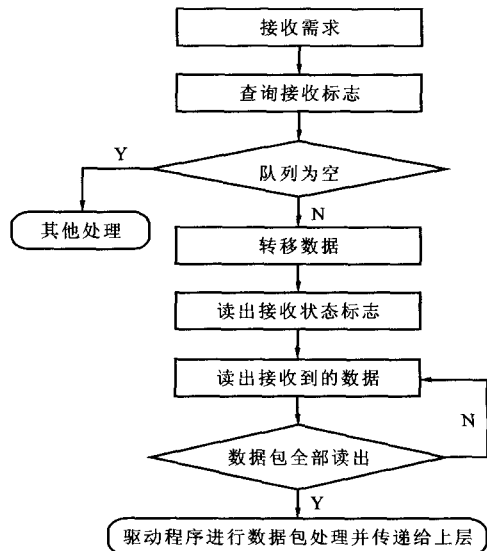


图5 数据包接收流程

获得设备的私有信息指针并初始化其各成员的值.网络设备的打开函数使能设备使用的硬件资源,申请 I/O 区域,并激活设备发送队列.网络设备的关闭函数停止设备传输包,并释放设备所使用的 I/O 区域等.

5 接口与协议的实现

网络设备驱动功能模块负责处理网络设备的具体逻辑,而通用网络驱动接口和简单网络协议为该功能模块的实现提供了必要的接口和服务基础.

通用网络驱动接口规范是 EFI 中网络设备驱动的一个标准,它的主要接口实现如下^[6]:

(1) GetState 接口:获取网卡的操作状态.

(2) Start 和 Stop 接口:Start 接口将通用网络驱动接口的状态从 Stopped 变换为 Started;而 Stop 接口则将通用网络驱动接口的状态从 Started 改变到 Stopped.

(3) GetInitInfo 和 GetConfigInfo 接口: GetInitInfo 接口用来获取通用网络驱动接口的初始化信息,这些信息包括 MemoryRequired, FrameDataLen, LinkSpeeds[0-3], MediaHeaderLen, HWAddrLen, MCastFilterCnt, TxBufCnt, TxBufSize, RxBufCnt, RxBufSize, NvCount, NvWidth, IFtype, Duplex 和 LoopBack 等.而 GetConfigInfo 接口获取通用网络驱动接口控制的 NIC 的配置信息,这些信息包括总线类型,总线句柄和设备句柄等.

(4) Initialize 接口:该接口使用参数来重启和初始化 NIC,成功后网卡即可进行收包和发包操作.

(5) Reset 和 Shutdown 接口:Reset 接口重启,

并使用参数中的配置来初始化通用网络驱动接口。而 Shutdown 接口用来重启网卡,并使得它可被其他的驱动再次初始化。

(6) Interrupt 接口:该接口可以用来读取或者改变当前外部中断设置。

(7) Status 接口:该接口函数返回当前的中断状态或者已经发送到缓冲区的地址。

(8) Transmit 和 Receive 接口:Transmit 接口函数用来将发送包放入到发送链表中去。当发送包交给发送函数后,就被认定已经锁定,此时应用程序没有对缓存的所有权,因此,不能擅自释放它。数据发送成功后会产发送中断,即使中断被关闭,中断状态位还是会被设置,可以在轮询中查到。而当网卡接收到一帧数据后,Receive 接口用来将该数据拷贝到驱动的缓存中,拷贝完成后,这一帧数据就会从接收队列中删除。

简单网络协议为网络接口的初始化提供服务,主要包括启动与停止网络接口,发送与接收数据包等操作。该协议主要接口的实现如下:

(1) Start 和 Stop 接口:在调用 Start 接口之前,其它的 EFI_SIMPLE_NETWORK_PROTOCOL 接口都不可操作,它为进一步的操作命令提供了网络接口。而 Stop 接口与之相反,调用它将停止命令的进程。

(2) Initialize 接口:该接口将重置网络适配器,并且分配发送和接收缓冲区。

(3) Transmit 和 Receive 接口:发送接口将发送包放入发送队列中,而接收接口则将数据包从队列中取出。要使用这些接口,就必须加载协议。在通用网络驱动接口的驱动绑定函数 Start 中,需要打开它所依赖的 PCI I/O Protocol 和 Device Path Protocol,并读取网络设备信息,初始化全局控制结构。而简单网络协议是在网络设备驱动加载时加载的,在 EFI 驱动绑定协议的 Start 接口中,调用特定的函数完成简单网络协议的加载。整个网络设备驱动的框图^[8]如图 6 所示。

6 结束语

基于完全模块化的设计思路,依据 EFI 固有的驱动模型,介绍了 EFI 网络设备驱动的设计和实现方案。但鉴于 EFI 自身的复杂性,在 EFI 网络设备

驱动中并没有涉及到一些高级特性,只是完成了网络设备的一些基本操作。对网络设备的一些高级特性,如 TCP/IP 网络协议栈,还有待进一步的研究。

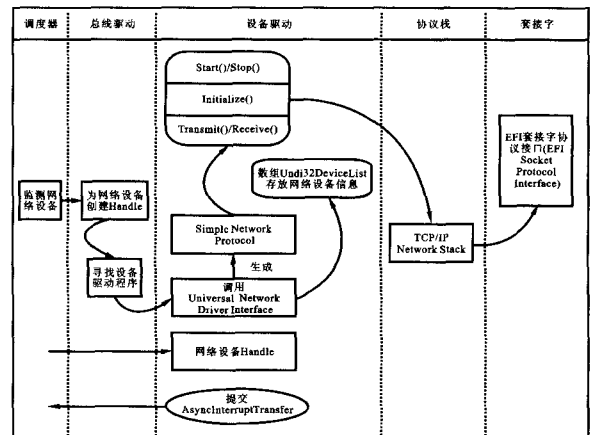


图 6 EFI 网络设备驱动图

参考文献:

- [1] Intel Corporation. Unified Extensible Firmware Interface Specification Version2. 3[S]. 2009.
- [2] Intel Corporation. EFI 1. 10 Driver Writer's Guide Draft for Review Version0. 9[S]. 2004.
- [3] Intel Corporation. Extensible Firmware Interface Library Specification Version1. 10. 1[S]. 2004.
- [4] 魏春娟,郑喜凤,丁铁夫. 基于 ARM 的嵌入式以太网通讯的实现[J]. 微电子学与计算机, 2008, 25(8): 232—235.
- [5] 殷薇. 基于 EFI 的 TCP/IP 协议栈的研究与实现[D]. 成都:电子科技大学, 2007.
- [6] 石浚菁. EFI 接口 BIOS 驱动体系的设计、实现与应用[D]. 南京:南京航空航天大学, 2006.
- [7] 潘登,刘光明. EFI 结构分析及 Driver 开发[J]. 计算机工程与科学, 2006, 28(2): 115—117.
- [8] 刘伟,胡平. 基于文件系统过滤驱动的文件加密系统设计[J]. 微电子学与计算机, 2009, 26(4): 114—117.

作者简介:

董丽丽 女, (1960—), 教授, 硕士生导师. 研究方向为分布式系统与计算机网络应用、数据挖掘。

张富凯 男, (1986—), 硕士研究生. 研究方向为分布式系统与计算机网络应用。

廖福燕 男, (1985—), 硕士研究生. 研究方向为数据仓库及数据挖掘。