第24卷第2期　　山东科技大学学报(自然科学版)　　Vol.24 No.2
2005年6月　Journal of Shandong University of Science and Technology(Natural Science)　Jun.2005
文章编号:1672-3767(2005)02-0062-04

Linux下USB主机控制器驱动的设计实现

武甲东[1,2],陈新华[2],张志敏[1]

(1. 中国科学研究院 计算技术研究所,北京 100080; 2. 山东科技大学 信息科学与工程学院,山东 青岛 266510)

摘　要:Linux操作系统具有良好的开放性和较强的可移植性,在当前嵌入式操作系统中被广泛采用;同时USB也具有极佳的通用性,是当前最为主流的通用外设接口。中科SOC系统开发USB主机控制器,并采用了Linux操作系统,本文首先介绍Linux驱动程序的架构,重点说明基于中科SOC系统的USB主机控制器驱动的开发实现。

关键词:Linux;USB;USB驱动程序;嵌入式系统

中图分类号:TP316　　**文献标识码**:A

Implementation of Driving of the USB Host Controller in Linux System

WU Jia-dong[1,2], CHEN Xin-hua[2], ZHANG Zhi-min[1]

(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China;
2. College of Information Science and Engineering, SUST, Qingdao, Shandong 266510, China)

Abstract: Linux is popularly adopted in embedded system, because Linux has good open style and it is transplantable. The USB also is used as a most popular bus standard in computer's peripheral interface. USB host controller has been developed in Zhongke-SOC. This paper introduces structures of the driving program of the Linux, we will emphasize on explanation of the implementation of driving of USB host controller in Linux system.

Key words: Linux; USB; USB Driving Program; embedded System

Linux操作系统是一个源码公开、结构清晰、功能强大,已成为一个稳定可靠功能完善的系统。其具有很强的可移植性,在当前迅速发展的SOC领域,是一种被广泛采用的嵌入式操作系统。

USB是一种新型的通用串行总线,它具有可热插拔和传输速度快的特点,使得支持USB技术的产品和设备越来越多,在工业界获得广泛支持应用。目前多数嵌入式芯片中实现了USB主机控制器,使得系统能连接USB设备。

中科SOC是中科院计算所开发的一款功能强大的嵌入式系统芯片,它采用了mips指令系统的CPU,并开发了USB1.1主机控制器,软件采用了Linux操作系统,本文首先介绍linux驱动程序的架构,再介绍USB系统组成,重点说明USB总线驱动的实现。

1　Linux驱动程序分析

设备驱动程序是操作系统内核和机器硬件之间的接口,为应用程序屏蔽了硬件细节。应用程序可把硬件设备当普通文件看待,并进行操作。设备驱动是内核的一部分,它主要完成以下功能:对设备进行初始化;使设备投入运行和退出服务;把数据从内核传送给设备和从设备接受数据;检测和处理设备出现的错误。

Linux系统下的设备分为字符设备、块设备和网络设备三种。字符设备指那些没有缓冲区,以字符流形式发送和接受的设备,如键盘、鼠标。块设备是以数据块为单位读写数据有缓冲区的支持。一个文件系统必须安装在操作系统的块设备上。网络设备在Linux系统中做专门处理。Linux

收稿日期:2004-11-02　　**基金项目**:国家863资助项目(2002AA1Z1040)

作者简介:武甲东(1978-),男,山东肥城人,硕士研究生,从事计算机体系结构与SOC设计方面的研究.

网络系统主要是基于 BSDunix 的 socket 机制。在系统和驱动程序之间定义数据结构(sk_buff)进行数据传递。系统支持对发送数据和接受数据的缓存,提供流量控制机制,及对多协议的支持。

2　USB 系统的组成

USB 系统组成如图 1 所示,由软、硬件组成。

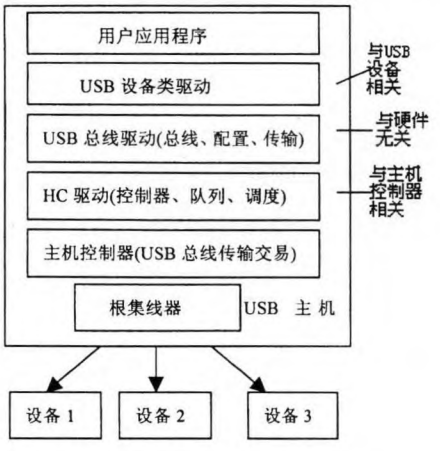

图 1　USB 系统组成图

Fig. 1 Sketch of USB system components

USB 软件系统由 USB 主机中的软件和 USB 设备中的固件构成。主机中的软件则由 USB 设备驱动,USB 总线驱动和 USB 主机控制器驱动 HCD(Host Control Driver)三部分构成。

主机控制器处于最底层,他负责对主机控制器进行抽象和对 USB 的低级支持。目前它有两种主机控制器接口:通用主机控制器接口(UHCI)和开放式主机控制器接口(OHCI),两者具有相似的处理能力,中科 SOC 设计实现中采用了 OHCI。主机控制器驱动为上层 USB 软件系统提供了统一的软件接口,使得 USB 上层驱动可不用理会硬件细节。它直接控制主机控制器的可操作寄存器来管理主机,为硬件中断提够中断服务处理例程,把上层软件的 USB 总线请求转化成符合 OHCI 规范要求的数据链表,并负责分配端点及传输带宽的调度等。在 linux 中由 ohci.o 模块来完成。

处于最上层的是 USB 设备类驱动软件,它按 USB 设备的协议要求与其对应的 USB 设备进行通信和读写控制,实现各个 USB 设备特殊的功能应用。如对键盘、鼠标和 U 盘的数据传输支持。

在主机控制器和 USB 设备驱动之间的是 USB 总线驱动,Linux 系统中由 Usbcore.o 模块实现。USB 总线驱动是独立于 USB 设备和设备驱动软件,并为它们提供统一的编程接口。它实现对协议规定的 USB 总线以及 USB 设备共有的操作和性质提供支持,如设备插入和拔出的动态处理、地址分配、配置、数据传输、供电管理、设备标准请求处理等。

3　USB 主机控制器驱动的设计

OHCI 驱动的设计主要包括四个部分,分别是主机控制器的初始化和管理、交易执行和资源的调度、中断处理以及根集线器操作和控制。主机控制器驱动为 USB 总线驱动屏蔽了硬件操作,完成 USB 底层数据传输,它为上层驱动提供了简单的软件接口,定义了 usb_operations 函数结构体供 USB 总线驱动调用。

```
struct usb_operations {
int (*allocate)(struct usb_device *);/*分配 USB 设备资源*/
int (*deallocate)(struct usb_device *);/*收回 USB 设备资源*/
int (*get_frame_number) (struct usb_device *usb_dev); /*取当前帧号*/
int (*submit_urb) (struct urb* purb); /*提交 USB 请求块*/
int (*unlink_urb) (struct urb* purb); /*撤消 USB 请求块*/
};
```

函数调用涉及两个不同数据结构的变量,其中 usb_device 包含了对设备地址、描述表、配置、端点等基本设备信息;urb 说明 USB 总线请求的基本信息,包括 USB 通道信息、数据缓冲区、带宽和完成处理函数等。USB 总线上的所有交易都是由函数 submit_urb()发起的。

3.1　主机控制器的初始化和管理

操作系统加载 ohci.o 后驱动调用 ohci_int(void)函数,进行主机控制器的初始化。首先创建一个 ohci 实体(结构如下所示),分配主机控制器的寄存器的内存影射,由函数 memset(ohci->hcca, 0, sizeof(struct ohci_hcca))来完成;然后置控制器于复位 10 个毫秒,调用 hc_start(ohci)函数继续对主机控制器进行初始化。其中包括初始链表头指针寄存器,设置中断使能位,启动控制器使其总状态机进入操作状态。最后通过 usb_new_device(usb_dev)函数把虚拟根集线器连接到 USB 总线,并注册了 usbhub 的驱动,把 ohci 的控制权移交给了上层驱动即 USB 总线驱

动。至此完成了对 ohci 的初始化。

```
typedef struct ohci {
……
struct ohci_hcca * hcca; /* hcca */
int irq; /* OHCI 对应的中断号 */
struct ohci_regs * regs;/* OHCI 寄存器 */
int ohci_int_load[32];/* 32 个 interrupt 链表 */
ed_t * ed_rm_list[2]; /* 除掉的端点链表 */
ed_t * ed_bulktail; /* bulk 链表 */
ed_t * ed_controltail;/* control 链表 */
ed_t * ed_isotail;/* iso 链表 */
struct usb_bus * bus;/* USB 总线 */
struct usb_device * dev[128];/*设备链表 */
struct virt_root_hub rh;/*虚拟根集线器 */
……
} ohci_t;
```

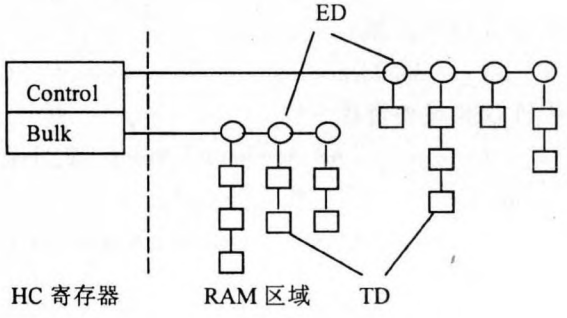

图2 控制和批传送的数据链表结构图

Fig. 2 Diagram of structure of data chain of control and bulk transmission lists

3.2 交易执行和资源的调度

主机控制器所进行的 USB 数据交易是按照 OHCI 驱动所建立的端点描述符链表 ED(endpoint descriptor)和传输描述符链表 TD(transfer descriptor)工作的。当 USB 上层驱动发起一个 USB 总线请求时通过调用 sohci_submit_urb(urb)来建立和填充相应的数据链表;当撤消某个 USB 总线请求时调用 sohci_unlink_urb(urb)删除和释放内存中的数据链表。USB 总线上发起的所有数据传送(控制传送、批传送、中断传送、同步传送)的数据都是来自这些数据链表中定义的数据。下面按不同数据类型来介绍数据链表的维护。

(1)对于控制和批传送的数据链表结构如图2所示,它们各有一个相互独立的 ED 链表, ED 为串行连接,每个 ED 下面挂有一个 TD 链表,主机控制器中用两个32位寄存器保存链表的头指针。

主机控制器在每一帧开头的10%带宽内处理控制和批传送数据链表,两个链表之间的调度按照主机 HcControl 寄存器中 CBSR(contro bulk service ratio)的值进行。即每处理一个批传输交易,要处理 CBSR + 1 个控制传送交易。ED 描述表中定义了 USB 总线上某个通道连接设备上的一个端点的基本信息,如设备地址、端点号、最大数据包长度、全/低速标志等。如下所示:

```
struct ed {
__u32 hwINFO; /* 数据端点基本信息 */
__u32 hwTailP; /* TD 链尾指针 */
__u32 hwHeadP; /* TD 链头指针 */
__u32 hwNextED; /* 指向下一个 ED */
struct ed * ed_prev; /* */
……
__u8 state; /* 当前 ED 状态 */
struct ed * ed_rm_list; /* 被删除的 ED 链 */
……
} __attribute((aligned(16)));
```

发起一个总线请求时,通过函数 ep_add_ed(urb->dev, pipe, urb->interval, 1, mem_flags)建立一个新的 ED,然后用 ep_link(ohci, ed)把它连接到主机控制器。TD 描述表中定义了,某次交易的具体信息,包括本次交易的包标志、数据 toggle 位和数据缓冲区的头尾指针等。其数据结构如下所示:struct td {

```
__u32 hwINFO; /* 包类型,数据 tlggle 等 */
__u32 hwCBP;/* 数据缓冲区头指针 */
__u32 hwNextTD;/* 指向下一个 TD */
__u32 hwBE;/* 缓冲区尾指针 r */
__u16 hwPSW[MAXPSW]; /* 同步描述符的 PSW */
struct ed * ed; /*指向所属 ED */
struct td * next_dl_td; /* 完成队列 TD 链 */
struct urb * urb; /*所属 URB */
……
} __attribute((aligned(32)))
```

函数 td_submit_urb (urb_t * urb)发起了传送请求,此函数中它按照请求的不同类型,数据缓冲的大小用 td_fill()函数建立响应的 TD 描述表。这样数据链表建立完成后便可通过 OHCI 状态寄存器对应位的置"1":writel (OHCI_BLF, &ohci->regs->cmdstatus)通知主机控制器去处理响应链表进行 USB 总线传输了。

(2)USB 中的中断和同步传送数据链表的结

构与控制和批传送有所不同,它的拓扑结构如图3所示。它有32个分支,每个分支为一个ED链表。处在树型结构末梢的ED节点为每个链表的头节点,主机控制器驱动以32位对齐的格式把他们存放在系统内存的某一区域,用控制器中Hchcca寄存器指向其基址。在每一帧中,当主机控制器要处理周期性链表时,它就根据当前主机帧序号低5位的值决定从基址开始第几条数据链被处理。由图3看到处于结构中处于不同位置的ED会有不同频率的访问周期。叶子节点每32ms处理一次,根节点每1ms处理一次,同步传送的ED连接在根节点上。因为USB的每一帧为固定长度既一个毫秒12兆位,所以为保证每条周期性链表的ED节点都能在它所在的帧中被处理,中断链表上的ED节点数目不能无限增加。提交一个中断或同步传送请求时需调用函数usb_check_bandwidth (urb - >dev, urb)检查带宽利用率;并分配带宽usb_claim_bandwidth (struct usb_device * dev, struct urb * urb, int bustime, int isoc)以保证在ED插入的链表不会出现调度溢出。

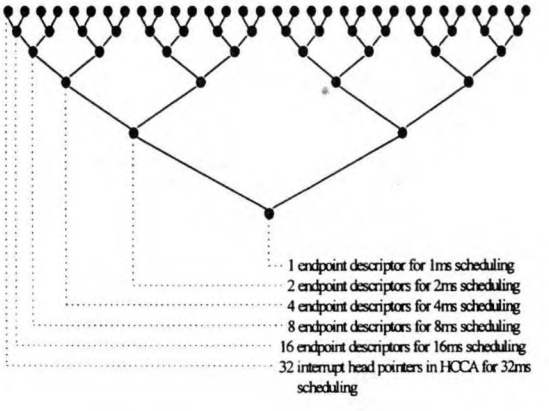

图3　中断链表拓扑结构图

Fig. 3 Diagram of topologic configuration of interrupting chain lists

3.3　主机控制器中断处理

OHCI有多个中断源,分别是:调度溢出中断(SO),即上述的如果中断链表在某一帧中不能完成处理而产生的中断;写回完成队列中断(WHD),在一帧结尾如果主机控制器将HcDoneHead值写回HCCA时产生;帧开始中断(SF),主机发出SOF包后产生此中断;远程唤醒中断(RD),当主机检测到USB设备发出复位信号时产生;系统故障中断(UE),当主机控制器检测到自己不能处理的错误发生时产生此中断,如果此中断发生需要对主机控制器进行复位;帧序号溢出中断(FNO),当HcFmNumber寄存器溢出时产生;根集线器状态变化中断(RHSC)。如果这些中断被使能,则当某中断源满足中断条件时主机控制器就会产生发往系统的硬件中断。

以上中断源其中最重要且经常发生的是写回完成队列中断。主机控制器处理数据链表时,把完成的TD数据节点挂到完成队列中。在每一帧的开始如果OHCI已完成提交的交易请求,就会产生此中断,驱动就会进入中断处理函数hc_interrupt (int irq, void * __ohci, struct pt_regs * r)在这个函数中通过处理完成队列函数dl_done_list (ohci_t * ohci, td_t * td_list)把USB交易完成的数据传给上层USB驱动。

3.4　虚拟根集线器

为了方便管理主机控制器上的各个USB端口,在驱动中实现了一个虚拟的USB根集线器,并为之分配固定设备号为"1"。可以把对根端口管理,交给USB集线器驱动统一管理。建立了符合USB规范的设备描述符和配置描述符。用rh_submit_urb (urb_t * urb)函数来模拟执行hub.c发来的总线请求。集线器控制器以查询的方式循环读每个根端口的状态,主机控制器把从根端口寄存器读来的数据送回,这样集线器驱动就会根据端口状态的变化判断设备的插入和拔出。

4　结束语

因为当前Linux调试工具中没有单步跟踪内核代码和设置断点的功能,因此仍然采用printk语句打印调试信息,同时用ioctl方法进行调试。Linux的发展非常迅速,它不同版本的内核提供的设备驱动接口之间有一定程度的不兼容,在调试中科SOC中采用的是2.2.14版本的内核,开发的OHCI的驱动实现了对USB鼠标、键盘和移动硬盘的支持。

参考文献:

[1] Alessandro Rubini & Jonathan Corbet. Linux Device Drivers O'Reilly & Assocoates[R]. Inc. 2002.

[2] Programming Gide for Linux USB Device Drivers[EB/OL]. http:/usb.cs.tum.edu, 2002.

[3] 陈莉君. Linux操作系统内核分析[M]. 北京:人民邮电出版社,2000.

[4] Universal Serial Bus Specification Reversion 1.1[S]. www.usb.org, 1998.