基于 WDM 模型 USB 驱动程序的设计与研究①

张智邦，鲍苏苏，金 敏

(华南师范大学 计算机学院，广州 510631)

摘 要：随着 USB 应用的日益广泛，作为其重要组成部分的驱动程序也受到了更多的关注。首先在此对 WDM 驱动模型和 USB 分层结构进行了详细的阐述，并以在新太科技实际工作中所使用 LPC2378 开发板为例，详细介绍了如何利用 DriverStudio 工具快速开发基于 WDM 模型的 USB 驱动程序。

关键词：WDM 驱动模型；USB 驱动；DriverStudio 工具

USB Driver's Design and Research Based on WDM Model

ZHANG Zhi-Bang, BAO Su-Su, JIN Min

(School of Computer, South China Normal University, Guangzhou 510631, China)

Abstract: With the increasingly wide range of USB applications, the driver, which is one of its important parts, also draws more attention. First, the paper expounded WDM driver model and USB hierarchical structure in details. And with the LPC2378 development board as an example which had been used in the practical work in The SUNTEK, a way of using DriverStudio to quickly design the USB driver based on WDM driver model is introduced.

Key words: WDM driver model; USB driver; Tools of DriverStudio

1 引言

USB，是 Universal Serial Bus(通用串行总线)的简称,为 PC 与其外围设备之间的连接提供了一种标准化、单一化的接口。而在开发 USB 设备时，设备驱动程序的设计是其中一项重要的技术环节，它直接影响整个设备系统的性能。Windows98 及其更高版本的操作系统对 USB 总线提供了全面的支持，并且用 WDM 驱动程序模型代替了 VxD 设备驱动程序。WDM 支持 USB 协议，并为其提供了高效的开发平台。

2 WDM驱动模型

2.1 WDM 驱动分层结构

WDM (Windows Driver Model)是在 Windows NT 驱动模型的基础上发展起来的[9],增加了对即插即用(PnP)、高级电源管理(Power Management)、Windows 管理接口(WMI)的支持。WDM 是一个分层化的驱动程序模型，在这个模型中，驱动程序的层或堆栈一起工作处理 I/O 请求[1]。在 windows 的 I/O 子系统中，每个 I/O 操作可以通过一个 IRP 描述，驱动程序的工作过程就是对 IRP 的处理过程。

在如图 1 的层次结构。用户模式的应用程序通过 Win32 API 调用 Win32 子系统,该系统又通过系统服务接口进入到内核模式下的 I/O 管理器,由管理器根据用户的相应需求,通过使用 IRP(I/O 请求包)与内核模式的 WDM 驱动程序通信,完成对驱动程序的调用,最终实现应用程序对硬件设备的调用及控制。图中左边是一个设备对象堆栈。一个物理硬件可以有多个这样的数据结构。处于堆栈最底层的设备对象称为物理设备对象(Physical Device Object)，或简称为 PDO。位于设备对象堆栈中部的对象称为功能设备对象(Functional Device Object)，或简称 FDO。在 FDO 的上面和下面一般有一些过滤器设备对象(filter device object)。在 WDM 驱动程序模型中，大多数硬件设备一般只包含两个驱动程序。其中一个驱动程序称为功能驱动程序，

① 收稿时间:2011-03-26;收到修改稿时间:2011-04-23

即硬件驱动程序。它知道如何控制设备的主要功能，负责初始化 I/O 操作，处理 I/O 操作完成时所产生的中断事件，并为用户提供一种适当的设备控制方式。另一个驱动程序称为总线驱动程序(bus driver)，它控制对总线上的所有设备的访问[2]。总线驱动程序负责枚举它的总线，这意味着发现总线上的全部设备和检测设备何时被添加或删除。分层结构可以使 I/O 请求过程更加明了，见图 1 内核模式的右侧。每个影响到设备的操作都使用 I/O 请求包，通常 IRP 先被送到设备堆栈的最上层驱动程序，然后逐渐过滤到下面的驱动程序[3]。

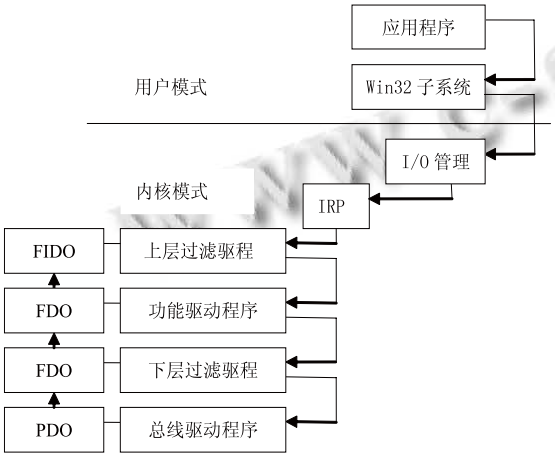

图 1

2.2 WDM 驱动模型加载过程

对于 WDM 驱动模型，当有设备插入电脑时总线驱动会枚举到设备的插入，并通知 Pnp 管理器寻找需要加载的驱动程序。首先根据此种设备的总线，加载相应的总线驱动设备以得到物理设备对象（pdo）。Pnp 管理器根据设备的 PID 和 VID 来加载此设备的功能驱动程序，并将刚才创建的 pdo 做为参数传递给 AddDecice 例程，而 AddDecice 将自己功能设备对象挂载到 pdo 的上面[4]。在某些时候，在物理设备对象和功能设备对象之间会有一层或多层过滤驱动设备对象。这样一个完整的设备驱动程序栈就建立了。

3 USB驱动概述

3.1 USB 驱动程序分层结构

USB 设备驱动程序是基于 WDM 结构之上的。WDM 的分层驱动程序结构在 USB 设备上体现在驱动程序分为 USB 总线驱动程序和 USB 功能驱动程序两个层次。USB 总线驱动程序一般由 win98 或更高的操作系统提供，它位于 USB 功能驱动程序的下面，负责与实际的硬件打交道，实现烦琐的底层通信[5]。USB 功能驱动程序(也称为用户自定义驱动)由设备开发者编写，位于 USB 总线驱动程序的上层，不与实际的硬件打交道，而是通过向 USB 总线驱动程序发送包 URB(USB Request Block 请求块)的 IRP 请求包，来实现对 USB 设备信息的发送或接收[6]。

3.2 USB 驱动程序工作原理

USB 驱动程序的工作原理可以如下描述：

① 若应用程序想对 USB 设备进行 I/O 操作，它便使用 Windows API 函数对 Win32 子系统进行 Win32 调用。此调用由 I/O 系统服务接收并通知 I/O 管理器，I/O 管理器对此请求构造成一个合适的 I/O 请求包，并把它传递给 USB 功能驱动程序。

② USB 功能驱动程序接收到这个 IRP 以后，根据 IRP 中包含的具体操作代码，构造相应的 USB 请求块并把 URB 放到一个新的 IRP 中，然后把此 IRP 传递到 USB 总线驱动程序，USB 总线驱动程根据 IRP 中所含的 URB 执行相应的操作(如从 USB 设备读取数据)，并把操作结果通过 IRP 返还给 USB 功能驱动程序[7]。

③ USB 功能驱动程序接收到此 IRP 后，将操作结果通过 IRP 返还给 I/O 管理器，最后 I/O 管理器将此 IRP 中操作结果返还给应用程序，至此应用程序对 USB 设备的一次 I/O 操作完成。

④ 功能驱动程序除负责处理应用程序的 I/O 请求外，还要处理 PnP 管理器发送给它的 PnP 请求(如设备启动请求 IRP_MN_START_DEVICE，设备删除请求 IRP_MN_REMOVE_DEVICE 等)。通过对这些 PnP 请求的处理，USB 功能驱动程序可支持设备的热插拔和即插即用功能。

4 USB驱动程序设计

4.1 驱动程序框架设计

在此项目中主要是完成功能性驱动程序的开发，总线驱动由系统提供。在开发工具上本文选择的是 Visual C++，WINDDK，Driverstudio。DDK 软件包中包括有关设备驱动程序开发的文档、编译驱动程序时所需要的头文件和库文件、调试工具和一些设备驱动程序范例。但是直接使用 DDK 开发比较困难，而且设备的驱动程序本身比较复杂，一旦运行错误可能会对

整个操作系统产生灾难性的后果[8]。所以我们选择配合第三方软件来实现。DriverStudio 是专门用于设备驱动程序开发的软件包，包含 VtoolsD、SoftICE 和 DriveWorks 等开发工具。可以实现驱动制作的“自动化”，安装驱动向导一步一步的生成驱动。下面将要详细介绍的是如何为 LPC2378 开发板的 USB 模块开发具有批量传输和中断传输功能的 USB 驱动程序,通过端点 1 来实现中断传输，端点 2 实现批量传输。

在做好相关的配置工作之后就可以按照 DriverStudio 的 DriverWorks 来逐步生成驱动框架了。其关键步骤如下：

① 从 VCIDE 的菜单"DriverStudio"中选择"DriverWizard",在所示的第一个对话框中,写上项目名称"LPCusb"。

② 第二步,需要选择驱动程序的类型。选择 WDM 类型。

③ 第三步,选择驱动程序操作的总线类型。这里选择 USB。在 USB Vendor ID 和 USB ProductID 中填入 USB 设备的 VID 和 PID, 这必须和 LPC2378 固件中的设备描述符中的 VID，PID 一致。

④ 第四步,添加 Endpoint1 和 Endpoint2 的定义。由于 Endpoint0 为默认控制端点,故无需设置。芯片的端点具体配置如下:1 IN/OUT Interrupt0x81/0x01 64/64，2 IN/OUT Bulk 0x82/0x02 64/64。

⑤ 第五步，为驱动程序选择接口函数。

⑥ 第六步，选择为 Endpoint1 和 Endpoint2 生成读写代码。

⑦ 第十四步,为驱动程序生成一个 console 的测试程序。

至此,一个 USB 驱动的框架就已产生。在工作区中有两个工程文件,一个就是自己的 USB 驱动程序框架,另一个就是相应的 console 测试程序。下面就可以根据我们的需求,对它们进行修改,添加相应的代码,得到我们所需要的驱动程序。

4.2 端点的读写函数实现

本驱动程序主要实现了端点 1 与端点 2 的读写功能，在此以端点 1 读函数为例来阐述驱动程序的具体实现。其主要代码如下：

```
PURBpUrb=m_Endpoint1IN.BuildInterruptTransfer
(outputBuffer, //读缓冲区
outputSize,//读数据的数据字节数
FALSE,//表示设备传输的字节数不可以少于指定的字节数
NULL,//连接下一个传输的 URB，这里没有，置为 NULL。
NULL); //置为 NULL，分配一个新的 URB。
if(pUrb==NULL)    //如果分配失败
{
//返回资源不足
status=STATUS_INSUFFICIENT_RESOURCES;
}
else
{
//提交 URB，并设置 3s 超时
status=m_Endpoint1IN.SubmitUrb(pUrb,NULL,this
,DlyTime);
//实际读到的数据字节数
I.Information()=pUrb->UrbBulkOrInterruptTransfer.
TransferBufferLength;
//删除刚刚分配的 URB
delete pUrb;
}
```

端点 1 读函数为 LpcUsbDevice: Endpoint1_Read_Handler(KIrpI)。其中 I 是应用程序提交给驱动程序的 I/O 请求包，驱动程序通过 I.IoctlBuffer() 和 I.IoctlOutputBufferSize()可以获得读数据成功后所存储的输出缓冲区 outputBuffer 以及输出缓冲区大小 outputSize。并以他们为参数调用函数 m_Endpoint1IN.BuildInterruptTransfer 相应的请求生成 URB，最后将生成的 URB 通过 m_Endpoint1IN.SubmitUrb(pUrb, NULL,this,1000)提交给 USB 总线驱动，然后由 USB 总线驱动和具体的 USB 设备通信以完成上层应用所需要的服务。其中参数 1000 表示延迟 1 秒，即如果端点 1 已被占用则驱动程序会等待 1 秒，如果在这 1 秒内端点 1 被释放则驱动程序就接着提交上次未完成的请求。如果在 1 秒内没有被释放则驱动程序错误返回。如果将此参数设为 0 则驱动程序一直等待直到所请求端点释放为止。

下位机 USB 设备收到上位机的请求后首先将所请求的结果数据返回给 USB 总线，USB 总线再将数据返回给 USB 驱动程序并将其存入到 outputBuffer 缓冲区中，最后由系统将 outputBuffer 中的数据拷贝给用户

应用程序。至此端点 1 的读过程到此完成了。

端点 1 的写函数以及端点 2 的读写函数都大同小异，最大的不同就是端点 2 是采用的批量传输方式，生成 URB 的函数为 m_Endpoint2IN.BuildBulk Transfer。在此 USB 驱动中端点 1 使用的是中断传输适用于传输少量和中量，且对服务周期有要求的数据。比如 USB 鼠标、键盘等要求主机在规定的时间内进行轮询。端点 2 使用的是批量传输适用与传输大量且对传输时间和传输速率没有严格要求的数据。当 USB 总线带宽紧张时它会为其他传输类型让出自己所占用的帧/小帧的时间，而本身被延迟。

4.3 应用程序访问驱动程序

应用程序访问驱动程序不是通过驱动程序名来访问的而是通过 128 位的全局标志符 GUID 来实现对驱动程序的访问的，这个 GUID 是通过 Driverstudio 自动生成的。应用程序通过函数 OpenByInterface(GUID* pClassGuid, DWORD instance, PDWORD pError)来打开 USB 驱动程序，驱动程序通过安装文件（.inf）来识别 USB 设备。

5 结语

USB 技术的不断发展和完善，已经使其逐渐成为先进总线接口技术的标志和方向，开发一些特定功能的 USB 接口并设计其设备驱动程序也将成为应用 USB 技术的关键。如果在 USB 驱动开发中直接利用 DDK 将是一个比较繁琐的过程。而利用 DriverStudio 工具简化了开发过程，加快了开发周期。

参考文献

1 武安河.Windows2000/XP WDM 驱动程序开发.第 2 版.北京:机械工业出版社,2005.98–99.

2 张帆,史彩成.Windows 驱动开发技术详解.北京:电子工业出版社,2008.400–456.

3 于勇.基于 Windows 的 USB 接口 WDM 驱动研究和应用.南京:南京信息工程大学,2008.

4 Cant C. Writing Windows WDM Device Drivers. Published by R&D/Freeman.2000.

5 Windows 2000 DDK Documents.Mircosoft Corporation.

6 王根根.基于 Windows WDM 的 USB 设备驱动程序的开发与应用[硕士学位论文].太原:中北大学,2007.

7 苏远平.USB 总线接口的 WDM 驱动程序设计.电脑知识与技术,2006,36,142–144.

8 何丽华.基于 Windows 的 USB 驱动程序开发.微型电脑应用,2010:26,2.

9 Oney W. Programming the Microsoft Windows Driver Model. 北京:国防工业出版社,2001.27–29.

10 黄陇,李虎译.Windows 核心编程.第 4 版.北京:机械工业出版社,2008.593–607.

(上接第 209 页)

载后已经在 PCB 板上下载调试通过，并能演示液晶显示触摸屏的基本功能，液晶屏能正确的显示数据和图片，屏幕显示画面基本无闪烁，画面切换速度快, 触摸点响应迅速。无闪屏或花屏等缺点。该设计为分析液晶触摸屏接口电路提供了参考，有助于进一步研究以 Cortex-M3 为内核的具有丰富硬件资源的 LM3S3749 嵌入式芯片和 Linux 操作系统上的移植。

参考文献

1 Burr B. Corporation touch screen controller ADS7846. [2004-11-11]. http://www.ti.com/

2 郭改枝,郝怀姝,张鹏举.基于嵌入式系统触摸屏接口电路的实现.内蒙古师范大学学报(自然科学汉文版),2009,38(1):55–57.

3 贾智平,张瑞华.嵌入式系统原理与接口技术.北京:清华大学出版社,2005.93–170.

4 翟霄翔.电阻式触摸屏在嵌入式系统中的应用.电子测量技术,2006,29(2):36–37.

5 datasheet–lm3s3749–zh_cn.pdf.[2009-10-26]. http: //www.zlgmcu.com/

6 王田苗,魏洪兴.嵌入式系统设计与实例开发.第 3 版.北京:清华大学出版社,2008.93–180.

7 王晖,马明锦.基于触摸屏控制器 ADS7846 的触点坐标与压力的测量计算.电子实际应用,2003,(9):78–80.