# 基于DDK的USB接口WDM驱动开发

邓玖根[1,2],张正荣[1],胡 松[1],唐小萍[1]
(1.中国科学院 光电技术研究所,四川 成都 610209;
2.中国科学院 研究生院,北京 100039)

**摘 要**:USB以其诸多优点已被广泛用于PC接口设计,由此开发了基于USB的光刻机对准系统。与传统对准系统相比,新系统的稳定性大幅提高,同时成本显著降低。但USB设备驱动程序的开发是系统的关键和难点。以WDM的结构和USB通信模型为基础,结合图示详细介绍了与USB的WDM驱动开发相关的几个关键数据结构及其相互关系,并对开发过程中的疑难问题进行了深入分析,结合实际系统详细地介绍了驱动开发的整个流程及其相关注意事项。系统运行结果表明所开发的驱动程序满足系统需求。

**关键词**:通用串行总线;WDM;驱动程序;DDK
**中图分类号**:TP311.1 **文献标识码**:A **文章编号**:1673-629X(2008)03-0013-04

# USB's WDM Driver Development Based on DDK

DENG Jiu-gen[1,2], ZHANG Zheng-rong[1], HU Song[1], TANG Xiao-ping[1]
(1. Institute of Optics and Electronics, the Chinese Academy of Sciences, Chengdu 610209, China;
2. Graduate School of the Chinese Academy of Sciences, Beijing 100039, China)

**Abstract**: Because of USB's many merits, it's widely used in PC's interface design; therefore developed a new alignment system based on USB. Compared with traditional alignment systems, the new one's stability improves a lot, and its cost reduces drastically. But the driver's development of USB device was the key and nodus of the whole system. Introduces some key data structures related to USB's WDM driver in detail based on the framework of WDM and USB's corresponding model and combined with figures, and analyses some problems in the development process in depth. Finally, introduces the flow of driver development and other related notices. New system's practical performance shows that the driver developed meets requirement.

**Key words**: USB; WDM; driver; DDK

## 0 引 言

USB(Universal Serial Bus,通用串行总线)是由INTEL、微软、IBM等公司为解决传统总线不足而推出的一种新型总线标准。目前,USB2.0最高传输速率已经达到480Mb/s,可以满足包括视频设备在内的多种外部设备数据传输的需求。此外,USB总线还具有安装方便、支持热拔插、易于扩展等优点,是一种具有高性价比的接口,被广泛用于PC接口开发。

鉴于USB接口的诸多优点,笔者开发了基于USB接口的光刻机对准系统,克服了传统对准系统接口复杂、成本高、不稳定等缺点。要实现成功的开发,驱动程序的设计是一个重要环节。文中介绍了如何利用DDK开发基于WDM的USB驱动程序。

**收稿日期**:2007-06-16
**作者简介**:邓玖根(1983-),男,四川乐山人,硕士研究生,从事微电子设备系统软件研究;张正荣,高工,研究方向为光刻机软件系统。

## 1 WDM介绍

在Windows2000系统中,软件要么执行在用户模式,要么执行在内核模式。在x86计算机上,当用户模式程序需要读取设备数据时,先是通过系统服务接口调用内核模式的服务例程,内核模式的服务例程首先检查传递给它们的参数,然后创建一个称为"I/O请求包(IRP)"的数据结构,并把这个数据结构送到指定驱动程序的入口,执行IRP的设备驱动程序通过硬件抽象层(HAL)访问硬件读取数据。

设备驱动程序是一个软件组件,装入后即成为操作系统内核的一部分,为硬件和用户应用程序提供通信桥梁。Windows 2000系统可以使用多种驱动程序,包括虚拟设备驱动程序和内核模式驱动程序,PnP驱动程序是一种遵循Windows 2000即插即用协议的内核模式驱动程序,而WDM驱动程序又是一种PnP驱动程序。

WDM(Windows Driver Model)是Microsoft公司推

出的新型驱动程序模型，它增加了对即插即用(PnP)、电源管理(Power Management)、WMI 的支持。在一个 WDM 驱动程序模型中，每个硬件至少有两个驱动程序：一个是功能驱动程序，负责初始化 I/O 操作、处理 I/O 操作完成时所带来的中断事件并为用户提供一种设备适合的控制方式；另一个是总线驱动程序，负责管理硬件与计算机的连接[1]。WDM 驱动程序模型使用了如图 1 所示的层次结构[2]，一旦总线驱动程序检查到新硬件存在，则响应 PnP 管理器的 IRP_MN_QUERY_DEVICE_RELATIONS 请求并创建一个 PDO，之后 PnP 管理器参照注册表中的信息载入与这个 PDO 相关的过滤器和功能驱动程序，驱动程序调用 AddDevice 例程构建相应设备对象。AddDevice 例程通常是调用 IoCreateDevice 函数创建设备对象，并用 IoAttachDeviceToDeviceStack 函数把设备对象连接到当前设备堆栈的顶部，完成如图 1 所示层次结构的构造。之后，PnP 管理器向该设备的驱动程序发出副功能码为 IRP_MN_START_DEVICE 的 IRP，为其分配资源并启动设备。之后，该设备将进入启动状态，可以开始通信。

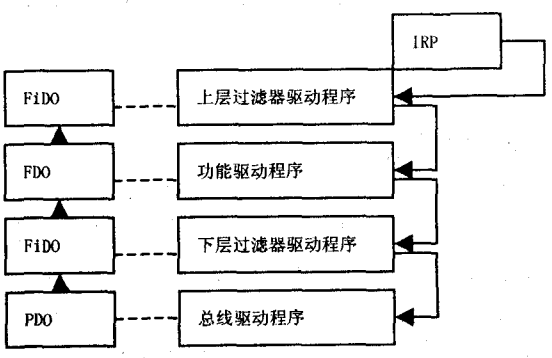


图 1 WDM 中设备对象和驱动程序的层次结构

Windows 2000 中，I/O 管理器使用驱动程序对象来代表设备驱动程序，当操作系统加载一个驱动程序时，I/O 管理器负责为其建立一个驱动程序对象，而驱动程序栈中的每个驱动程序都要为其控制的设备建立一个设备对象(如图 1 所示)。驱动程序对象和设备对象是 WDM 驱动程序中两种非常重要的数据结构，它们结构形式和两者之间的关系如图 2 所示。

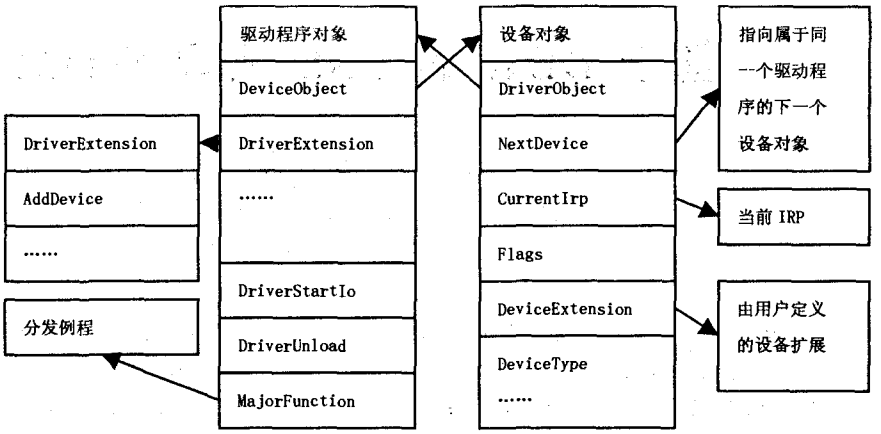


图 2 驱动程序对象与设备对象的结构以及它们之间的关系

如前所述，Windows2000 操作系统使用 I/O 请求包(IRP)的数据结构与内核模式驱动程序通信，所以 IRP 在 WDM 驱动程序中也起着至关重要的作用，因此要掌握 WDM 驱动开发，还必须正确理解和使用 IRP。IRP 数据结构定义于 wdm.h[3] 文件中，其重要的域有 MdlAddress、AssociatedIrp、IoStatus、CurrentLocation、PendingReturned、CancelRoutine 以及 Tail。其中，CurrentLocation 为非透明域，它和 Tail.Overlay.CurrentStackLocation(PIO_STACK_LOCATION)都没有公开给驱动程序使用。CurrentLocation 为当前 I/O 堆栈单元的索引而 CurrentStackLocation 就是指向它的指针。当某个“实体”创建 IRP 时，同时还创建了一个与之关联的 I/O 堆栈(IO_STACK_LOCATION)结构数组。I/O 堆栈中重要的域有 MajorFunction、MinorFunction、Parameters、DeviceObject、CompletionRoutine 等。MajorFunction 域为 IRP 的主功能码，与驱动程序对象中 MajorFunction 表的某个派遣函数指针相对应，而 MinorFunction 是该 IRP 的副功能码。DeviceObject 是与该堆栈单元对应的设备对象的地址，由 IoCallDriver 函数填写。因此，可以清晰地给出 WDM 驱动程序模型中非常重要而又难于理解的 IRP、I/O 堆栈、设备对象以及驱动程序对象之间的关系，如图 3 所示(图中只给出重要的域)。至此，已经对一个

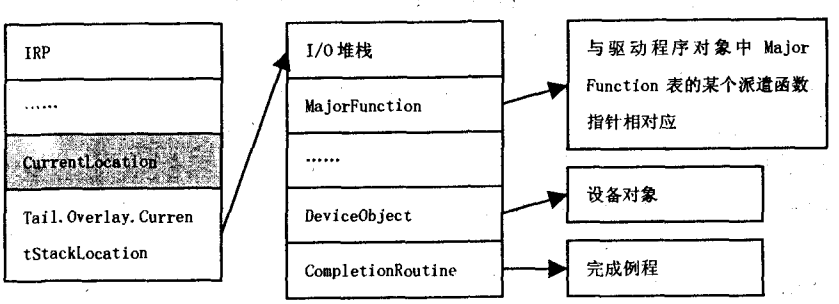


图 3 IRP、I/O 堆栈、设备对象以及驱动程序对象之间的关系

WDM 驱动程序的核心数据结构有了完整的了解。

一个完整的 WDM 驱动程序就像一个包含许多例程的容器,当操作系统遇到一个 IRP 时,它就调用驱动程序中的例程来执行该 IRP 相关的各种操作。一个 WDM 驱动程序的基本例程有驱动程序入口例程、即插即用例程、分发例程、电源管理例程和卸载例程,而一些复杂的 WDM 驱动程序还包含中断服务例程(ISR)、延时过程调用(DPC)等例程,一些高级的驱动程序还将处理 IRP 串行化、同步、WMI、DMA 等问题,因此,开发 WDM 驱动程序的主要工作就是根据具体设备的要求实现这些例程。

## 2　USB 驱动程序设计

一个完整的 USB 系统包括主机和 USB 设备,其多层次的通信模型如图 4 所示[4],其中,USB 主控制器负责处理主机与设备之间电气和协议层的互连,USB 系统还使用 USB 主控制器来管理主机和 USB 设备之间的数据传输,而应用软件不能直接访问 USB 设备硬件。微软提供的一组驱动程序占据了图 4 中 USB 系统软件方块的底部,包括主控制器驱动程序(OPENHCI.SYS 或者 UHCD.SYS),hub 驱动程序(USBHUB.SYS),和一个类驱动程序(USBD.SYS),而由笔者开发的 WDM 驱动程序占据图 4 中 USB 系统软件方块的顶部,因此更具体的 USB 驱动结构如图 5 所示。

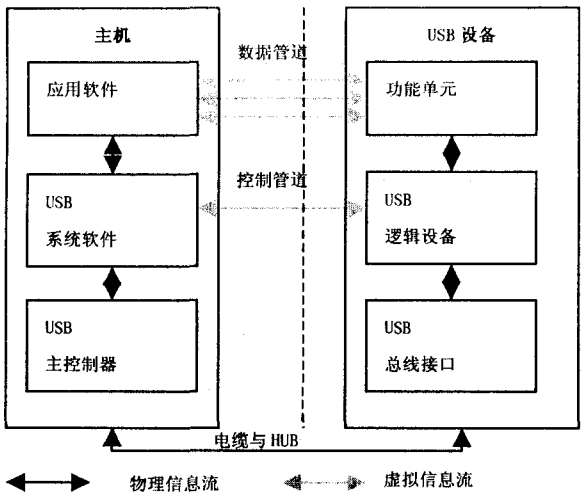


图 4　USB 通信模型

通常,一个 USB 设备可以有一个或者多个配置,每个配置可有一个或多个接口,一个接口又可有一个或多个端点,它们分别与各自的描述符对应,可通过发送一个类型为_URB_CONTROL_DESCRIPTOR_REQUEST 的 URB 来获取这些信息。

在 USB 系统中,USB 设备驱动程序一般不直接与硬件通信,而是通过创建 URB 并把 URB 提交到总线驱动程序来完成相应操作。URB 是 USB 请求块的缩写,它是一种共用体类型的数据结构,在 DDK 的 usbdi.h[3]文件中有定义,其成员中除了_URB_HEADER 外的其他成员均以_URB_HEADER 结构开始,即_URB_HEADER 是每个 URB 的开始,其定义如下:

```
struct_URB_HEADER {
    USHORT Length;
    USHORT Function;
    USBD_STATUS Status;
    //仅由 USBD 使用的域
    ……
};
```

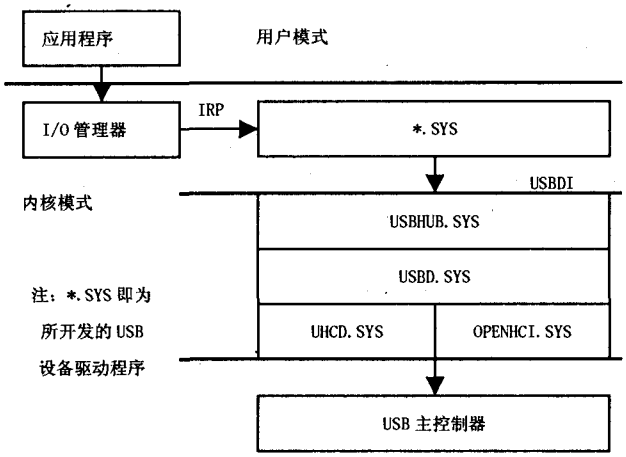


图 5　USB 驱动体系结构

Length 字段指明了 URB 所含的字节数,Function 字段指明了 URB 的功能代码,两者应在使用 URB 前设置,而各个 URB 结构的具体定义可参考 DDK 中的相关定义。可以说,在理解了 WDM 驱动程序模型的基础上,理解 URB 和_URB_HEADER 结构就成了掌握 WDM 型 USB 驱动程序的关键。

根据 USB 协议[4],USB 的数据传输方式有 4 种:块(bulk)、控制(control)、中断(interrupt)、同步(isochronous)。为了进行数据传输,通常有两种方法可以创建 URB:一种是用 DDK 定义的宏;另一种是手动创建。在系统中,图像数据采用块传输,而控制命令采用控制传输。

下面是用 DDK 定义的宏创建和发送块传输类型的 URB 的关键代码:

```
{……//创建 URB
URB urb;
UsbBuildInterruptOrBulkTransferRequest ( &urb, sizeof (_URB_
BULK_OR_INTERRUPT_TRANSFER), hpipe, NULL, Irp->
MdlAddress, ubufferlength, USBD_TRANSFER_DIRECTION_IN
| USBD_SHORT_TRANSFER_OK, NULL);
……//发送 URB
```

```
PIRP
Irp = IoBuildDeviceIoControlRequest(IOCTL_INTERNAL_USB_
SUBMIT_URB, pdx -> StackDeviceObject, NULL, 0, NULL, 0,
TRUE, &event, &iostatus);
PIO_STACK_LOCATION stack = IoGetNextIrpStackLocation
(Irp);/* 获取 I/O 堆栈 */
stack -> Parameters.Others.Argument1 = (PVOID) urb;/* 把
URB 的地址填入 I/O 堆栈的 Parameters.Others.Argument1 域
*/
NTSTATUS status = IoCallDriver(pdx -> StackDeviceObject,
Irp);/* 发送请求到下一层驱动程序 */
```

由于控制命令的数据较少，可用_URB_CONTROL_VENDOR_OR_CLASS_REQUEST 类型的 URB 来完成，此 URB 可用宏 UsbBuildVendorRequest 构造，若在某些特殊场合需要自己构造一个控制传输的 URB(手动创建)时，其代码应具有如下格式：

```
{PURB urb = NULL;
urbSize = sizeof(struct_URB_CONTROL_TRANSFER);
urb = ExAllocatePool(NonPagedPool, urbSize);
RtlZeroMemory(urb, urbSize);
urb -> UrbHeader.Length = sizeof(struct_URB_CONTROL_
TRANSFER);
urb -> UrbHeader.Function = URB_FUNCTION_CONTROL_
TRANSFER;//功能代码
//设置其他域
urb -> UrbControlTransfer.PipeHandle = pipehandle;
.....
}
```

## 3 DDK 开发环境构造

要用 DDK 开发基于 WDM 的 USB 驱动程序，需要安装 VC++ 和 DDK，DDK 可从微软的网站上获得，需要注意的是在安装 DDK 前应该先完成 VC++ 的安装。之后可运行 setenv.bat 来设置 build 环境，也可手动添加环境变量来进行设置[5]。一切就绪后，即可进行 WDM 驱动程序的开发。

## 4 结束语

介绍了基于 WDM 的驱动程序的基本结构，同时对开发过程中的疑难问题进行了深入剖析。在此基础上，介绍了 USB 的通信模型，分析了基于 WDM 的 USB 驱动开发的关键所在，给出了相应例程，并介绍了 DDK 开发环境的构建，最终结合实际系统完成了基于 DDK 的 USB 接口 WDM 驱动开发和调试。

**参考文献：**


[1] 武安河. Windows 2000/XP WDM 设备驱动程序开发[M]. 第 2 版. 北京：电子工业出版社，2005.

[2] Oney W. Programming the Windows Driver Model[M]. US: Microsoft Press, 1999.

[3] Microsoft Corporation. DDK Documentation[CP/DK]. 2003. http://www.microsoft.com.

[4] USB-IF. Universal Serial Bus Specification Revision 2.0 [EB/OL]. 2002. http://www.usb.org.

[5] 张 弘. USB 接口设计[M]. 西安：西安电子科技大学出版社，2002.


(上接第 12 页)

合 HTTP 协议所包含的内容协商机制，构建网页实时、在线翻译系统的具体方案。该方案对于传统网页在线翻译的不便和效率低下起到了改进的作用，内容协商机制的实现使得网页的翻译操作以及具体服务提供者对于客户端透明，使得使用浏览器浏览的客户端得以省去大量的不必要的操作时间。

通过一系列实验来比较传统网页在线翻译和基于内容协商和网络缓存网页在线实时翻译在网页呈现方面的性能。证实了改进方案的网络缓存技术对于网页内嵌对象，尤其是对于静态图像文件的缓存，使得改进方案相对于原始方案在重复请求相同网页或者同一站点类似网页的呈现效率上有了很大提高。实验数据证明改进方案重复请求的 PST 几乎是原始方案的 PST 的 50%。

**参考文献：**


[1] 王少春，陈家骏，王启祥，等. Internet 在线翻译浏览器技术探讨[J]. Application Research of Computers, 2001(1): 11-13.

[2] Hao Mao. On Applied Machine Translation Softwares & Web Sites[J]. Chinese Science & Technology Translators Journal, 2004, 17(4): 24-25.

[3] Wang Jia. A Survey of web Caching Schemes for the Internet [J]. Computer Communication Review, 1999, 29(5): 36-46.

[4] Krishnamurthy B, Mogul J C, Kristol D M. Key Differences between HTTP/1.0 and HTTP/1.1[M/OL]. 1999. http://www.research.att.com/bala/papers/h0vh1.html.

[5] Fielding R, Gettys J, Mogul J C, et al. Hypertext Transfer Protocol-HTTP/1.1 RFC2068[S/OL]. 1997-01. http://rfc.net/rfc2068.html.

[6] Fielding R, Mogul J C, Frystyk H, et al. Hypertext Transfer Protocol-HTTP/1.1. RFC 2616[S/OL]. 1999. ttp://rfc.net/rfc2616.html.

[7] Holtman K. Transparent Content Negotiation in HTTP-RFC 2295[S/OL]. 1998. http://rfc.net/rfc2295.html.

[8] Wessels D. Squid: The Definitive Guide[M]. [s.l.]: O'Reilly, 2004.