USB 驱动开发实例

本节具体介绍如何进行 USB 驱动的开发，本节采用的源码来源自 DDK 的源程序，其位置在 DDK 子目录的 src\wdm\usb\bulkusb 目录下。该示例很全面地支持了即插即用 IRP 的处理，也很全面地支持了电源管理，同时很好地支持了 USB 设备的 bulk 读写。如果从头开发 USB 驱动，往往很难达到 USB 驱动的稳定性，所以强烈建议读者在此驱动修改的基础上进行 USB 驱动开发。

1 功能驱动与物理总线驱动

DDK 已经为 USB 驱动开发人员提供了功能强大的 USB 物理总线驱动（PDO），程序员需要做的事情是完成功能驱动（FDO）的开发。驱动开发人员不需要了解 USB 如何将请求转化成数据包等细节，程序员只需要指定何种管道，发送何种数据即可。

当功能驱动想向某个管道发出读写请求时，首先构造请求发给 USB 总线驱动。这种请求是标准的 USB 请求，被称为 URB（USB Request Block），即 USB 请求块。这种 URB 被发送到 USB 物理总线驱动以后，被 USB 总线驱动所解释，进而转化成请求发往 USB HOST 驱动或者 USB HUB 驱动，如图 17-21 所示。

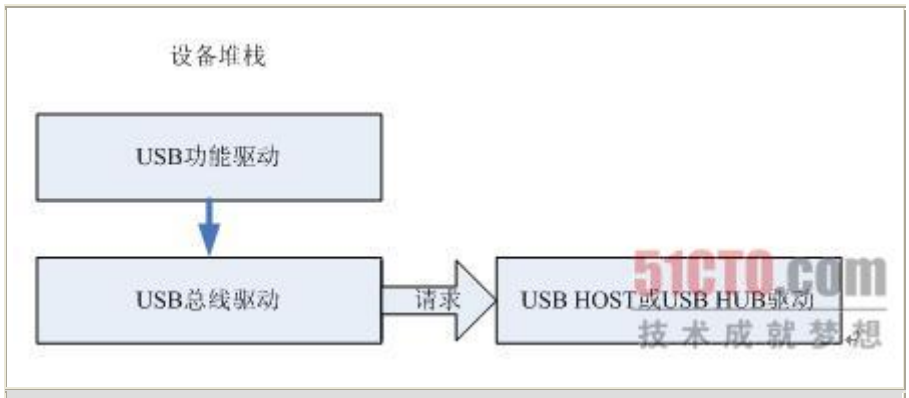

图 17-21 总线驱动与功能驱动的关系

可以看出，USB 总线驱动完成了大部分工作，并留给 USB 功能驱动标准的接口，即 URB 请求。USB 驱动开发人员只需要根据不同的 USB 设备的设计要求，在相应的管道中发起 URB 请求即可。

2. 构造 USB 请求包

USB 驱动在与 USB 设备通信的时候，如在控制管道中获取设备描述符、配置描述符、端点描述符，或者在 Bulk 管道中获取大量数据，都是通过创建 USB 请求包（URB）来完成的。URB 中填充需要对 USB 的请求，然后将 URB 作为

IRP 的一个参数传递给底层的 USB 总线驱动。在 USB 总线驱动中，能够解释不同 URB，并将其转化为 USB 总线上的相应数据包。

DDK 提供了构造 URB 的内核函数 UsbBuildGetDescriptorRequest，其声明如下：

```
VOID UsbBuildGetDescriptorRequest(

IN OUT PURB  Urb,

IN USHORT  Length,

IN UCHAR  DescriptorType,

IN UCHAR  Index,

IN USHORT  LanguageId,

IN PVOID  TransferBuffer  OPTIONAL,

IN PMDL  TransferBufferMDL  OPTIONAL,

IN ULONG  TransferBufferLength,

IN PURB  Link  OPTIONAL

);
```

- Urb：用来输出的 URB 结构的指针。
- Length：用来描述该 URB 结构的大小。
- DescriptorType：描述该 URB 的类型。它可以是 USB_DEVICE_DESCRIPTOR_TYPE、USB_CONFIGURATION_DESCRIPTOR_TYPE 和 USB_STRING_DESCRIPTOR_ TYPE。
- Index：用来描述设备描述符的索引。
- LanguageId：用来描述语言 ID。
- TransferBuffer：如果用缓冲区读取设备，TransferBuffer 是缓冲区内存的指针。
- TransferBufferMDL：如果用直接读取内存时，TransferBufferMDL 是直接读取内存时 MDL 的指针。
- TransferBufferLength：对于该 URB 所操作内存的大小。

在功能驱动中，所有与 USB 的通信，都需要用这个函数创建 URB，并通过 IRP 发送到底层 USB 总线驱动，以下是一个最基本的示例。

```
#001                UsbBuildGetDescriptorRequest(

#002                        urb,

#003                (USHORT) sizeof(struct _URB_CONTROL_DESCRIPTOR_REQUEST),
```

```
#004                        USB_DEVICE_DESCRIPTOR_TYPE,

#005                        0,

#006                        0,

#007                        deviceDescriptor,

#008                        NULL,

#009                        siz,

#010                        NULL);
```

3. 送 USB 请求包

功能驱动将 URB 包构造完毕后，就可以发送到底层总线驱动上了。URB 包要和一个 IRP 相关联起来，这就需要用 IoBuildDeviceIoControlRequest 创建一个 IO 控制码的 IRP，然后将 URB 作为 IRP 的参数，用 IoCallDriver 将 URB 发送到底层总线驱动上。由于上层驱动无法知道底层驱动是同步还是异步完成的，因此需要做一个判断。if 语句判断当异步完成 IRP 时，用事件等待总线驱动完成这个 IRP。

```
#001   //该函数实现对发送 URB 到 USB 物理总线驱动
#002   NTSTATUS
#003   CallUSBD(
#004       IN PDEVICE_OBJECT DeviceObject,
#005       IN PURB           Urb
#006       )
#007   {
#008       PIRP               irp;
#009       KEVENT             event;
#010       NTSTATUS           ntStatus;
#011       IO_STATUS_BLOCK    ioStatus;
#012       PIO_STACK_LOCATION nextStack;
#013       PDEVICE_EXTENSION  deviceExtension;
#014
#015    //首先是变量初始化
#016       irp = NULL;
#017       deviceExtension = DeviceObject->DeviceExtension;
#018       //初始化事件
#019       KeInitializeEvent(&event, NotificationEvent, FALSE);
#020    //创建 IO 控制码相关的 IRP
#021       irp = IoBuildDeviceIoControlRequest
```

```
(IOCTL_INTERNAL_USB_SUBMIT_URB,
#022                                          deviceExtension->
TopOfStackDeviceObject,
#023                                          NULL,
#024                                          0,
#025                                          NULL,
#026                                          0,
#027                                          TRUE,
#028                                          &event,
#029                                          &ioStatus);
#030
#031       if(!irp) {
#032     //如果 IRP 创建失败则返回
#033           BulkUsb_DbgPrint(1, ("IoBuildDeviceIo
ControlRequest failed\n"));
#034           return STATUS_INSUFFICIENT_RESOURCES;
#035       }
#036    //得到下一层设备栈
#037       nextStack = IoGetNextIrpStackLocation(irp);
#038       ASSERT(nextStack != NULL);
#039       nextStack->Parameters.Others.Argument1 = Urb;
#040       BulkUsb_DbgPrint(3, ("CallUSBD::"));
#041       BulkUsb_IoIncrement(deviceExtension);
#042    //通过 IoCallDriver 将 IRP 发送到底层驱动
#043       ntStatus = IoCallDriver(deviceExtension->
TopOfStackDeviceObject, irp);
#044    //如果 IRP 是异步完成时，等待其结束
#045       if(ntStatus == STATUS_PENDING) {
#046     //等待 IRP 结束
#047           KeWaitForSingleObject(&event,
#048                                 Executive,
#049                                 KernelMode,
#050                                 FALSE,
#051                                 NULL);
#052           ntStatus = ioStatus.Status;
#053       }
#054       //调用结束
#055       BulkUsb_DbgPrint(3, ("CallUSBD::"));
#056       BulkUsb_IoDecrement(deviceExtension);
#057       return ntStatus;
#058   }
```

此段代码可以在配套光盘中本章的 sys 目录下找到。

4. USB 设备初始化

USB 驱动的初始化和一般驱动类似，首先是进入入口函数 DriverEntry，在 DriverEntry 函数中，分别指定各个 IRP 的派遣函数地址、指定 AddDevice 例程函数地址、指定 Unload 例程函数地址等。

在 AddDevice 例程中，创建功能设备对象，然后将该对象挂载在总线设备对象之上，从而形成设备栈。另外为设备创建一个设备链接，便于应用程序可以找到这个设备。也可以根据具体需要，从注册表中读取一些必要的设置。

5. USB 设备的插拔

由于 USB 设备驱动是基于 WDM 框架的，因此需要对即插即用消息进行处理。BulkUSB 程序对即插即用 IRP 的支持非常完善，具体可以参照其代码，这里简单提一下其对插拔的处理。

插拔设备会设计 4 个即插即用 IRP，包括 IRP_MN_START_DEVICE、IRP_MN_STOP_DEVICE、IRP_MN_EJECT 和 IRP_MN_SURPRISE_REMOVAL。其中，IRP_MN_START_DEVICE 消息是当驱动争取加载并运行时，操作系统的即插即用管理器会将这个 IRP 发往设备驱动。因此，当获得这个 IRP 后，USB 驱动需要获得 USB 设备类别描述符，如设备描述符、配置描述符、接口描述符、端点描述符等。并通过这些描述符，从中获取有用信息，记录在设备扩展中。

IRP_MN_STOP_DEVICE 是设备关闭前，即插即用管理器发的 IRP。USB 驱动获得这个 IRP 时，应该尽快结束当前执行的 IRP，并将其逐个取消掉。另外，在设备扩展中还应该有表示当前状态的变量，当 IRP_MN_STOP_DEVICE 来临时，将当前状态记录成停止状态。

IRP_MN_EJECT 是设备被正常弹出，而 IRP_MN_SURPRISE_REMOVAL 则是设备非自然弹出，有可能意外掉电或者强行拔出等。在这种 IRP 到来的时候，应该强迫所有未完成的读写 IRP 结束并取消。并且将当前的设备状态设置成设备被拔掉。

6. USB 设备的读写

USB 设备接口主要是为了传送数据，80%的传输是通过 Bulk 管道。在 BulkUSB 驱动中，Bulk 管道的读取是在 IRP_MJ_READ 和 IRP_MJ_WRITE 的派遣函数中，这样在应用程序中就可以通过 ReadFile 和 WriteFile 等 API 对设备进行操作了。

在 IRP_MJ_READ 和 IRP_MJ_WRITE 的派遣例程中设置了完成例程，如图 17-22 所示。其原理是将读写的大小分成单位为 BULKUSB_MAX_TRANSFER_SIZE 的若干块，依次将请求发往底层 USB 总线驱动。第一个块是派遣例程先设置 BULKUSB_MAX_TRANSFER_SIZE 大小的读写，并设置完成例程，然后将请求发往 USB 总线驱动。当 USB 总线驱动完成 BULKUSB_MAX_TRANSFER_SIZE 大小的读写后，会调用读写的完成例程。

这时候在完成例程中再次发起 BULKUSB_MAX_TRANSFER_SIZE 大小的读写，并将请求发往底层 USB 总线驱动，当 USB 总线驱动完成后，又会进入完成例程。之后发送第三个数据块，并且依此类推直到传送完毕。

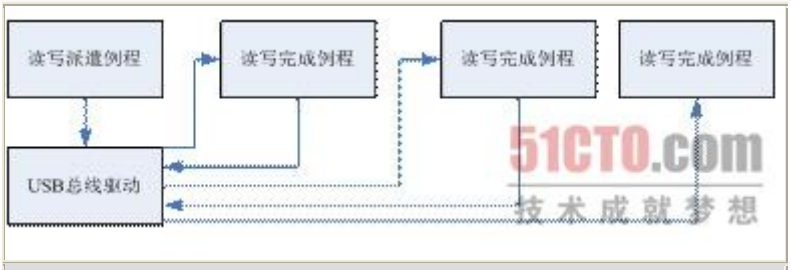

图 17-22 USB 读写派遣例程与完成例程

以下是 BulkUSB 的读写派遣函数的部分代码：

```
#001   NTSTATUS
#002   BulkUsb_DispatchReadWrite(
#003       IN PDEVICE_OBJECT DeviceObject,
#004       IN PIRP           Irp
#005       )
#006   {
#007       PMDL                  mdl;
#008       PURB                  urb;
#009       ULONG                 totalLength;
#010       ULONG                 stageLength;
#011       ULONG                 urbFlags;
#012       BOOLEAN               read;
#013       NTSTATUS              ntStatus;
#014       ULONG_PTR             virtualAddress;
#015       PFILE_OBJECT          fileObject;
#016       PDEVICE_EXTENSION     deviceExtension;
#017       PIO_STACK_LOCATION   irpStack;
#018       PIO_STACK_LOCATION   nextStack;
#019       PBULKUSB_RW_CONTEXT   rwContext;
#020       PUSBD_PIPE_INFORMATION pipeInformation;
#021
#022    //初始化变量
#023    urb = NULL;
#024       mdl = NULL;
```

```
#025       rwContext = NULL;
#026       totalLength = 0;
#027       irpStack = IoGetCurrentIrpStackLocation(Irp);
#028       fileObject = irpStack->FileObject;
#029       read = (irpStack->MajorFunction == IRP_MJ_READ) ? TRUE : FALSE;
#030       deviceExtension = (PDEVICE_EXTENSION)
DeviceObject->DeviceExtension;
#031    //....略
#032
#033    //设置完成例程的参数
#034       rwContext = (PBULKUSB_RW_CONTEXT) ExAllocatePool(NonPagedPool,
#035                                    sizeof(BULKUSB_RW_CONTEXT));
#036    //...略
#037       if(Irp->MdlAddress) {
#038           totalLength = MmGetMdlByteCount(Irp->MdlAddress);
#039       }
#040    //设置 URB 标志
#041       urbFlags = USBD_SHORT_TRANSFER_OK;
#042       virtualAddress = (ULONG_PTR)
MmGetMdlVirtualAddress(Irp->MdlAddress);
#043
#044    //判断是读还是写
#045       if(read) {
#046           urbFlags |= USBD_TRANSFER_DIRECTION_IN;
#047       }
#048       else {
#049           urbFlags |= USBD_TRANSFER_DIRECTION_OUT;
#050       }
#051
#052    //设置本次读写的大小
#053       if(totalLength > BULKUSB_MAX_TRANSFER_SIZE) {
#054           stageLength = BULKUSB_MAX_TRANSFER_SIZE;
#055       }
#056       else {
#057           stageLength = totalLength;
#058       }
#059
#060    //建立 MDL
#061       mdl = IoAllocateMdl((PVOID) virtualAddress,
#062                           totalLength,
#063                           FALSE,
#064                           FALSE,
#065                           NULL);
#066    //将新 MDL 进行映射
```

```
#067       IoBuildPartialMdl(Irp->MdlAddress,
#068                         mdl,
#069                         (PVOID) virtualAddress,
#070                         stageLength);
#071
#072    //申请 URB 数据结构
#073       urb = ExAllocatePool(NonPagedPool,sizeof
(struct _URB_BULK_OR_INTERRUPT_ TRANSFER));
#074
#075    //建立 Bulk 管道的 URB
#076       UsbBuildInterruptOrBulkTransferRequest(
#077                                urb,
#078                                sizeof(struct
_URB_BULK_OR_INTERRUPT_TRANSFER),
#079                                pipeInformation->PipeHandle,
#080                                NULL,
#081                                mdl,
#082                                stageLength,
#083                                urbFlags,
#084                                NULL);
#085
#086    //设置完成例程参数
#087       rwContext->Urb             = urb;
#088       rwContext->Mdl             = mdl;
#089       rwContext->Length          = totalLength - stageLength;
#090       rwContext->Numxfer         = 0;
#091       rwContext->VirtualAddress  = virtualAddress + stageLength;
#092       rwContext->DeviceExtension = deviceExtension;
#093    //设置设备堆栈
#094       nextStack = IoGetNextIrpStackLocation(Irp);
#095       nextStack->MajorFunction = IRP_MJ_INTERNAL_DEVICE_CONTROL;
#096       nextStack->Parameters.Others.Argument1 = (PVOID) urb;
#097       nextStack->Parameters.DeviceIoControl.IoControlCode =
#098                                                IOCTL_INTERNAL_USB_SUBMIT
_URB;
#099    //设置完成例程
#100       IoSetCompletionRoutine(Irp,
#101                              (PIO_COMPLETION_ROUTINE)BulkUsb_ReadWriteCo
mpletion,
#102                              rwContext,
#103                              TRUE,
#104                              TRUE,
#105                              TRUE);
#106    //将当前 IRP 阻塞
```

```
#107       IoMarkIrpPending(Irp);
#108    //将 IRP 转发到底层 USB 总线驱动
#109       ntStatus = IoCallDriver(deviceExtension->TopOfStackDeviceObject,
#110                               Irp);
#111    //...略去对不成功时的处理
#112       return STATUS_PENDING;
#113   }
```

此段代码可以在配套光盘中本章的 sys 目录下找到。