第 10 章

USB 过滤驱动开发

有时我们不想修改设备或者设备程序无法修改，但是又想改变设备的功能或者数据，这时就可以考虑使用过滤驱动了。另外，过滤驱动还可以用来截获设备传输的数据，著名的 Bus Hound 之所以能够捕捉设备的数据，使用的就是过滤驱动的方法。本章将介绍一个简单的 USB 过滤驱动的开发，它实现的功能是将第 4 章中的 USB 键盘“变”成第 5 章中的用户自定义的 USB HID 设备。

10.1 过滤驱动简介

过滤驱动(Filter Driver)有上层过滤和下层过滤两种，处于功能驱动之上的叫做上层过滤驱动，处于功能驱动程序之下(但仍处于总线驱动之上)的叫做下层过滤驱动。上层过滤驱动夹在应用程序(或者更高层的驱动程序)与驱动程序之间，而下层过滤驱动则夹在驱动程序与其下层驱动程序之间。数据在传递时必然会经过过滤驱动，因而过滤驱动可以将数据捕捉、修改或者拦截。

在本章所介绍的 USB 下层过滤驱动的开发过程中，过滤驱动的作用是将 USB 键盘改成用户自定义的 USB HID 设备。我们知道，在 Windows 下 USB 键盘是系统独占设备，所以使用第 5 章设计的访问 HID 设备的应用程序将无法以读方式打开这个 HID 设备。如果在这个 HID 设备上安装一个过滤驱动，将它返回的报告描述符中的应用集合用途改成用户自定义(即 0xFF)，那么系统将认不出这个集合，从而就不会加载键盘的驱动，而是加载一个 HID 兼容设备的驱动，这样使用我们的软件就可以访问它了。当然，还需要修改报告描述符中输出报告的定义，因为我们软件的输出报告是 8 字节的。另外，要能够正确显示开关状态，返回数据也得修改。

Driver Studio 提供了创建过滤驱动的向导，只要按照向导一步步生成，就可以构造出一个过滤驱动的基本框架。

10.2 使用 DS 创建一个下层过滤驱动

创建过滤驱动与创建功能驱动的前几个步骤是一样的，以下为具体步骤：

(1)设置工程路径以及工程名。如图 10.2.1 所示，将工程名设置为 MyUs-

bLowerFilter，然后单击 Next 按钮。

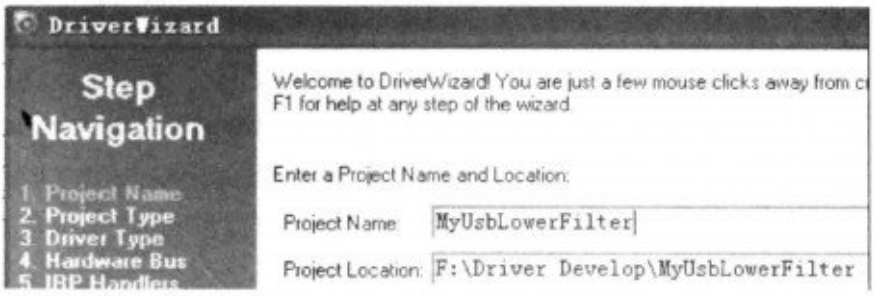

图 10.2.1　设置工程路径及工程名

(2)选择工程的类型。在弹出的设置工程类型对话框中选择 WDM 驱动(因为 USB 过滤驱动是属于 WDM 驱动)，如图 10.2.2 所示。驱动框架选择为 C++框架，之后单击 Next 按钮。

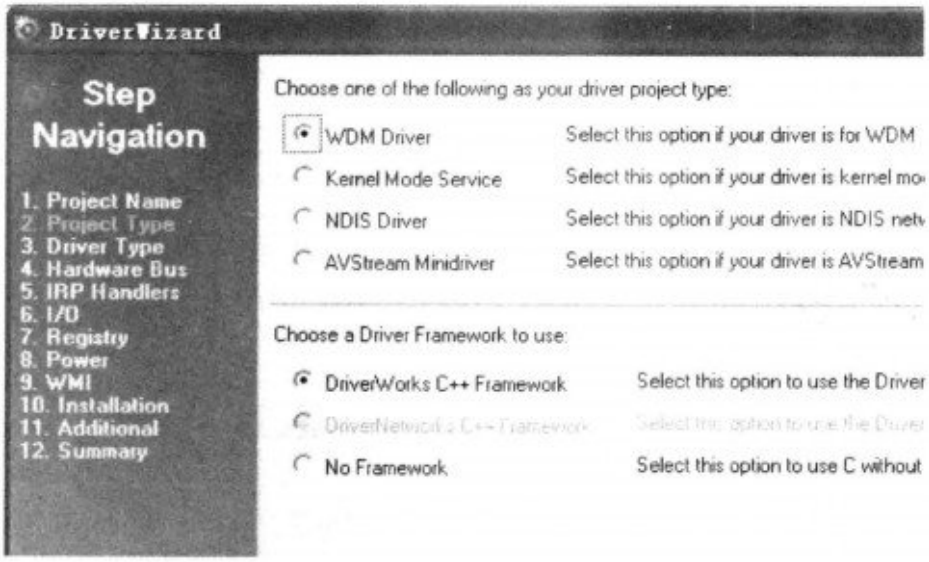

图 10.2.2　选择工程的类型

(3)选择驱动的类型。在选择驱动类型对话框中选择驱动类型为 WDM 过滤驱动(WDM Filter Driver)，然后单击 Next 按钮，如图 10.2.3 所示。

(4)选择过滤器的类型。如图 10.2.4 所示，在选择过滤器的对话框中，选择过滤的对象为设备(device)，设备描述填入“USB 人体学输入设备”。过滤器类型选择为下层过滤驱动(lower-level Filter)，并将下面的 Create Installation DLL 勾选上，这将帮我们产生一个安装驱动用的 DLL 工程，可以在安装过滤驱动时由应用程序调用，然后单击 Next 按钮。

(5)选择要过滤的 IRP。这里将 IRP_MJ_INTERNAL_DEVICE_CONTROL、IRP_MJ_READ、IRP_MJ_WRITE 三个勾选上，如图 10.2.5 所示。获取报告描述符以及读、写数据就是通过 IRP_MJ_INTERNAL_DEVICE_CONTROL 请求实现的，

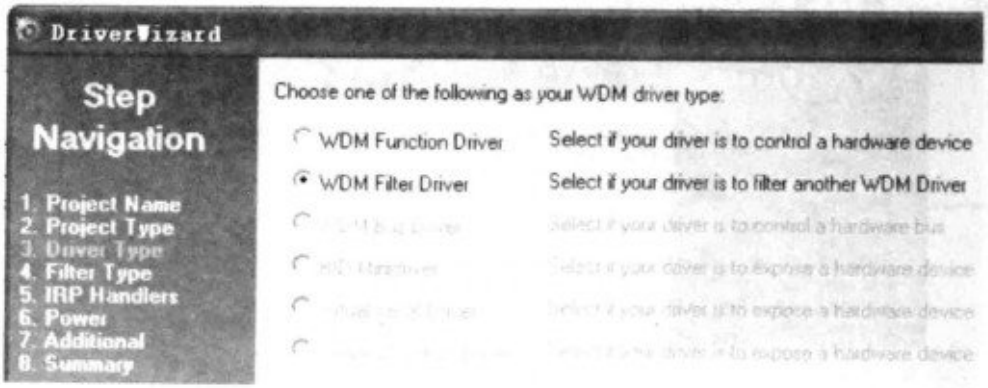

图 10.2.3　选择驱动的类型

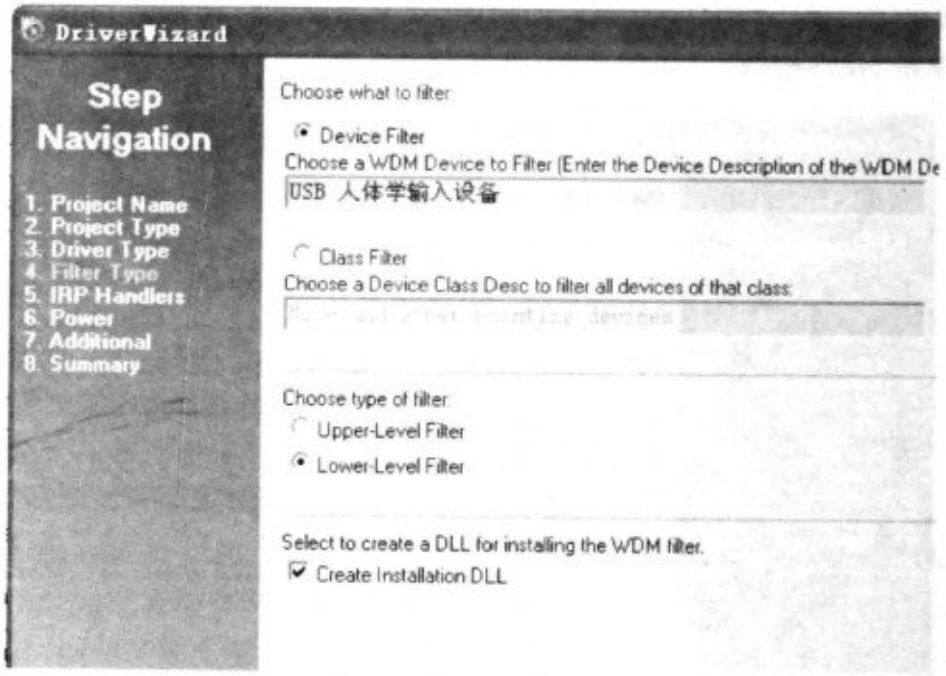

图 10.2.4　选择过滤器的类型

在这个请求完成时，修改返回的数据即可修改报告描述符和读取到的报告数据。

(6)配置 I/O 管理。如果你有应用程序需要访问该过滤驱动(例如捕捉经过该过滤驱动的数据，或者直接控制设备等)，在这里可以添加新的 IoControl 代码。我们这里不需要添加，仅是改变一下发送或返回数据。Only one handle can be open to the device at any time 和 Generate Test Application 都不用勾选，如图 10.2.6 所示。

(7)配置电源管理。电源管理使用向导的默认值即可，如图 10.2.7 所示。

(8)配置附加信息。附加信息使用向导的默认值即可，如图 10.2.8 所示。

(9)查看总结信息，在这里可以大致检查一下设置是否正确，如果发现有错误，还可以后退修改。确认无误后，就可以单击 Finish 按钮完成向导了，如图 10.2.9 所示。

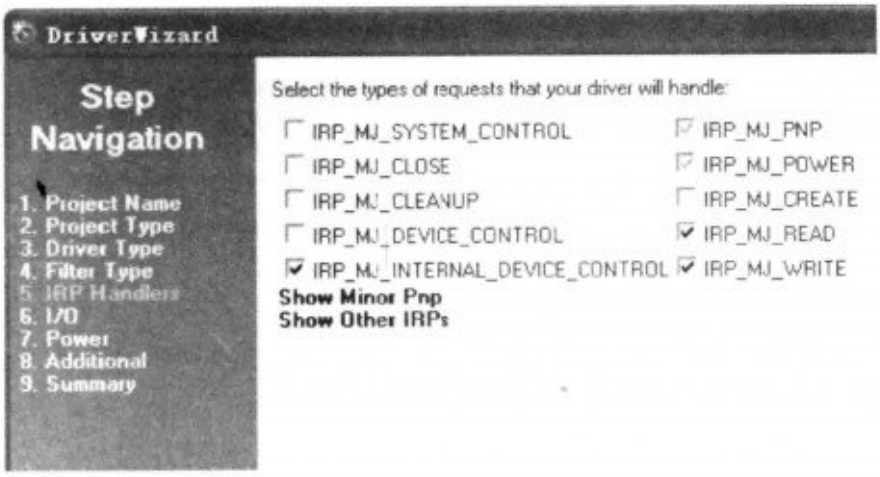

图 10.2.5　选择要过滤的 IRP

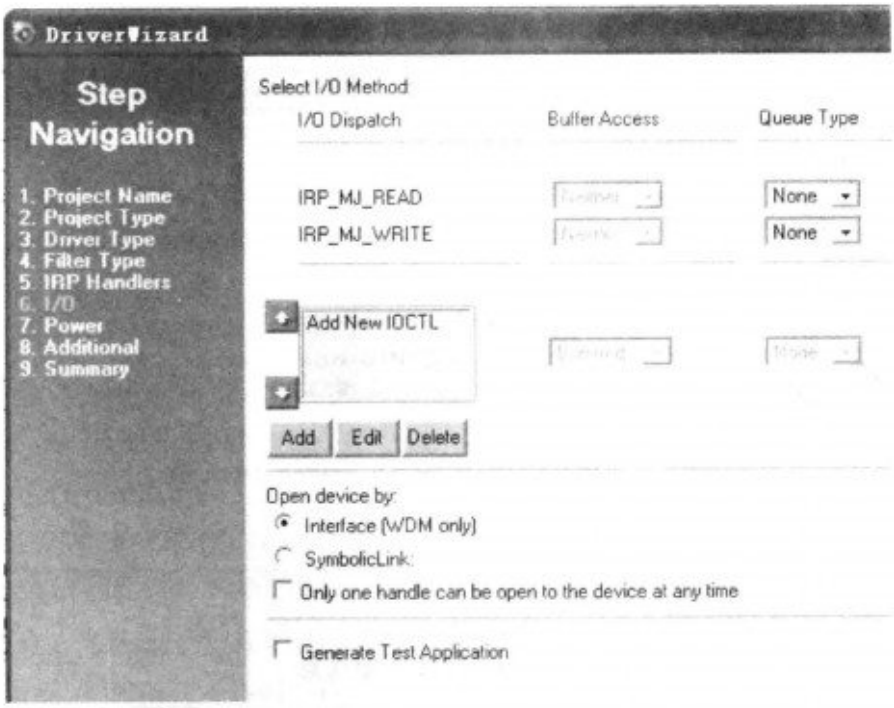

图 10.2.6　配置 I/O 管理

(10)编译工程。选择驱动工程，编译，同样会出现找不到 ntstrsafe.lib 的错误，在工程的 Settings 里将 ntstrsafe.lib 删除即可(参看第 9 章)。编译成功之后，就会在相应的目录(调试和发布版)下生成一个 MyUsbLowerFilter.sys 文件，它就是我们最终需要的驱动文件(当然接下来还需要修改相关的代码)。

选择安装驱动用的 DLL 工程，编译，将会得到 MyUsbLowerFilterDll.dll 和 MyUsbLowerFilterDll.lib 两个文件(位于 install 目录下的 objchk 或 objfre 目录下)。这两个文件就是开发安装程序要用到的 DLL 和库文件。你可以在这个 DLL 工程下修改代码，然后另外写个应用程序来调用这个 DLL；也可以参考这个 DLL 工

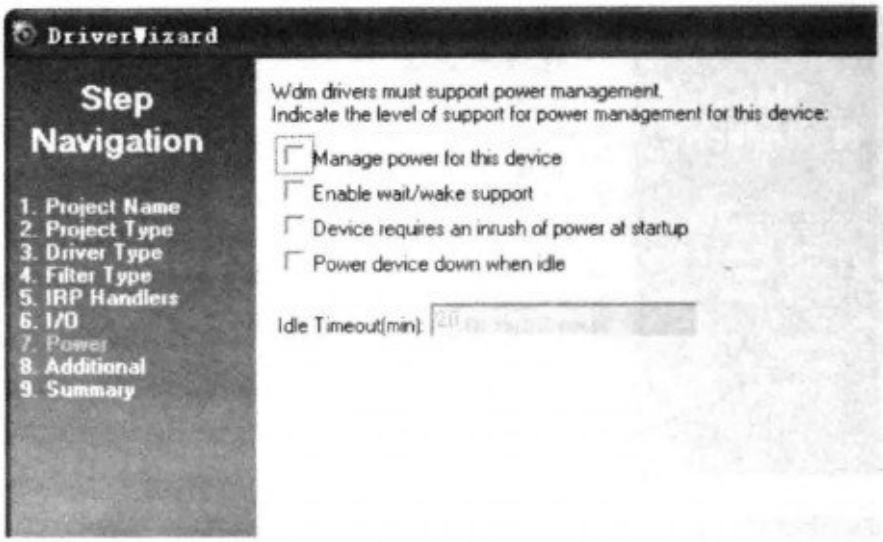

图 10.2.7　配置电源管理

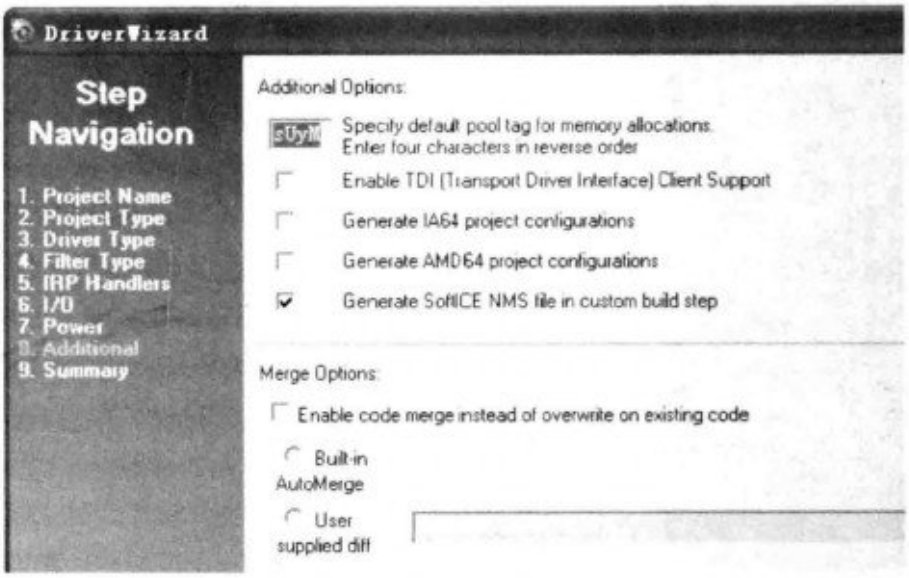

图 10.2.8　配置附加信息

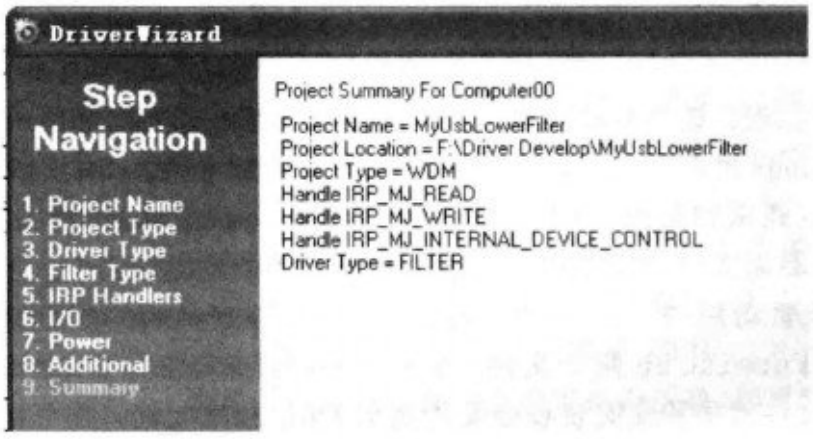

图 10.2.9　查看总结信息

程下的代码，另外创建一个独立的安装程序，这样调试起来就容易些，不用跟 DLL 连调。

10.3　过滤驱动代码的修改

主要修改 MyUsbLowerFilterDevice.cpp 文件中对相应 IRP 的处理。在修改代码之前，先说一些基本知识。

在向导生成的驱动代码中，是没有对数据过滤处理的，对于不需要处理的 IRP 请求，只要调用函数 PassThrough(I)简单地将这个 IRP 传递到下一层驱动即可。如果需要对某个 IRP 的数据进行过滤，怎么办呢？对于往设备输出数据比较简单，直接修改该 IRP 中缓冲区的数据即可。但是从设备读取数据时怎么处理呢？这就需要设置一个 IRP 完成时的处理函数了，当该 IRP 读取数据完成时，就进入该处理函数，对返回的数据进行修改。设置 IRP 完成处理函数的函数格式为

```
PassThrough(I, LinkTo(IrpCompletionRoutine), this)
```

其中，LinkTo 宏中的 IrpCompletionRoutine 就是要设置的完成函数，在这个函数中处理返回的数据即可。当然并不是对所有的 IRP 操作都需要设置完成函数，只有那些需要被过滤的请求才需要设置，可以从 IRP 中获取参数并判断是否为需要过滤的请求。

驱动程序调试比较麻烦，它不像普通程序那样可以设置断点、单步运行等。只能借助其他的一些工具来调试驱动，例如 DS 提供的 DriverMonitor 软件(所在位置：Compuware\ DriverStudio\ Tools\ Monitor\ monitor. exe)。在驱动程序中使用 KTrace 类来发送调试信息，这些发送的调试信息将显示在 DriverMonitor 中。在向导生成的代码中，有一个全局变量 T，它就是 KTrace 类的实例，我们可以使用 T 来显示调试信息。KTrace 类有很多重载函数，可以直接显示不同的数据类型。注意只有调试版(选择为 Win32 Checked)的驱动才会显示调试信息，发行版(选择为 Win32 Fre)的驱动不会显示调试信息。

下面就来修改驱动的代码，打开 MyUsbLowerFilterDevice.cpp 文件，可以找到一个 InternalDeviceControl(KIrp I)的函数。这个函数就是上层驱动与下层驱动之间打交道的函数。在这里先判断 I/O 控制代码是否为 IOCTL_INTERNAL_USB_SUBMIT_URB(需要引用头文件 kusb.h)，如果是，则说明是提交一个 URB。然后再从入口参数的 KIrp 类的 I 中获取到当前的 URB，就可以知道具体是什么请求了。获取 URB 的方法是调用 KIrp 类的 URB 函数，入口参数为 CURRENT 表示当前的 IRP。然后从 URB 结构体中就可以获取到该 URB 的功能(Function)代码。我们知道请求报告描述符是发送到设备的接口的，因而该 URB 的功能代码就是 URB_FUNCTION_GET_DESCRIPTOR_FROM_INTERFACE(0x28)，而批量或中断传

输的 URB 功能代码为 URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER (0x09)。如果功能代码为从接口获取描述符，那么就判断描述符类型是否为报告描述符(0x22)，如果是，就设置完成函数。如果功能代码为批量或中断传输，那么先判断是输入还是输出，如果是输入请求，就设置完成函数；如果是输出数据，就将数据长度改成 1 字节，并判断是否需要清除计数器(还记得自定义 HID 设备中那个计数器吗？这里将在过滤驱动中增加一个 MyCount 的变量实现)。

下面是实际修改好的 InternalDeviceControl 的代码：

```
NTSTATUS MyUsbLowerFilterDevice::InternalDeviceControl(KIrp I)
{
 T<<"进入 InternalDeviceControl 函数\n";                    //显示调试信息
 //如果 Ioctl 代码为 IOCTL_INTERNAL_USB_SUBMIT_URB,那么就是提交 URB 的请求,
 //在这里判断我们需要过滤的 IRP
 if(I.IoctlCode() == IOCTL_INTERNAL_USB_SUBMIT_URB)
 {
  PURB pUrb = I.Urb(CURRENT);                                //获取当前 IRP 的 URB
  if(pUrb! = NULL)                                           //如果 pUrb 有效
  {
   //T << * pUrb;          可以显示 URB 的详细信息(需要增加 usbd.lib 才能使用)
   T<<"URB 功能代码为:";                                     //显示 URB 的功能代码
   T<<pUrb->UrbHeader.Function;                              //获取功能代码
   T<<"\n";                                                  //换行
   switch(pUrb->UrbHeader.Function)                          //对功能代码散转
   {
    //如果功能代码为从接口请求描述符(0x28)
    case URB_FUNCTION_GET_DESCRIPTOR_FROM_INTERFACE:
     T<<"从接口获取描述符的 URB\n";                          //显示调试信息
     //判断请求的描述符是否为报告描述符(0x22)
     if(pUrb->UrbControlDescriptorRequest.DescriptorType == 0x22)
     {
      T<<"描述符的类型为报告描述符(0x22)\n";                 //显示调试信息
      T<<"设置完成函数\n";
      //设置完成函数为 IrpCompletionRoutine,并返回
      return PassThrough(I, LinkTo(IrpCompletionRoutine), this);
     }
    break;

    //如果功能代码为批量或中断传输(0x09)
    case URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER:
     T<<"批量或中断传输的 URB ";                            //显示调试信息
     //如果为输入请求(TransferFlags 的最低位为传输方向)
```

```
    if((pUrb->UrbBulkOrInterruptTransfer.TransferFlags)&0x01)
    {
     T<<"(输入传输)\n";                                    //显示调试信息
     T<<"设置完成函数\n";
     //设置完成函数为 IrpCompletionRoutine,并返回
     return PassThrough(I, LinkTo(IrpCompletionRoutine), this);
    }
    else                                                   //为输出请求
    {
     UCHAR * pBuf;                                         //保存缓冲区地址
     ULONG Len;                                            //保存长度
     T<<"(输出传输)\n";                                    //显示调试信息
     //获取缓冲区地址
     pBuf = (UCHAR * )pUrb->UrbBulkOrInterruptTransfer.TransferBuffer;
     //获取传输的长度
     Len = pUrb->UrbBulkOrInterruptTransfer.TransferBufferLength;
     if(Len == 8)                                          //8 字节的报告
     {
      if(pBuf[1]! = 0)              //当第二字节不为 0 时,要清除计数器
      {
       T<<"清除计数器\n";
       MyCount = 0;                                        //计数器清 0
      }
     }
     //仅发送 1 字节数据
     pUrb->UrbBulkOrInterruptTransfer.TransferBufferLength = 1;
     T<<"修改输出报告完毕\n";
    }
   break;
   default:
   break;
  }
 }
 //对于不需要过滤的 IRP,将它直接传递给下层即可
 NTSTATUS status = PassThrough(I);
 return status;
}
```

那么完成函数 IrpCompletionRoutine()中该怎么处理呢？这里要先判断传输的类型(即 URB 的功能代码)。

如果是控制传输(当获取描述符完成时,URB 的功能代码会变为控制传输,其代码值为 0x08,宏定义为 URB_FUNCTION_CONTROL_TRANSFER),则判断描述符的类型;如果是报告描述符,则要对报告描述符修改。报告描述符要修改应用集合的用途以及输出报告的长度。原来的报告描述符中设置的应用集合为 0x06(即用途为键盘),现在要改为用户自定义设备,所以改成 0xFF(改成 0x00 也行,只要系统不认识就好)。而输出报告的长度则通过修改填充的 3 位常量字段,因为原来的键盘程序使用了 5 位做 LED,这里将 3 位的填充改成 59 位,那么总共就有 64 位了(即 8 字节)。集合用途位于报告描述符偏移量 3 处(即第 4 字节),填充字段的长度位于偏移量 61 处(即第 62 字节)。

如果是批量或者中断传输,则判断是否为输入传输,如果是,就说明接收到了数据。然后读取数据的长度,看是否为 8 字节,如果是,就要修改这 8 字节的报告值。在原来的键盘程序中,按键情况是分布在这 8 字节输入报告中的,这里要根据这 8 字节数据将按键状态放入到第一字节中。具体请参看键盘程序的代码以及下面函数的代码。而第 2 至第 4 字节保存的为计数器值,这里将 MyCount 的值装入到这 4 字节中。剩余的 3 字节没有用到,将它们清零。

修改好的 IrpCompletionRoutine()函数如下:

```
NTSTATUS MyUsbLowerFilterDevice::IrpCompletionRoutine(KIrp I)
{
 NTSTATUS status = I.Status();
 //T.Trace(TraceInfo, __FUNCTION__" IRP %p, STATUS %x\n", I, status);
 // TODO: Add driver specific code to process the IRP that the lower
 // device has completed.
 UCHAR * pBuf;
 ULONG Len;
 T<<"进入完成函数\n";
 PURB pUrb = I.Urb(CURRENT);                    //获取当前 IRP 中的 URB
 if(pUrb! = NULL)                               //如果 pUrb 有效
 {
  switch(pUrb->UrbHeader.Function)              //对功能代码散转
  {
   //如果是控制传输(注意:当获取描述符的 IRP 完成时,功能代码会变成控制传输)
   case URB_FUNCTION_CONTROL_TRANSFER:          //值为 0x08
    //判断描述符类型是否为报告描述符(0x22)
    if(pUrb->UrbControlDescriptorRequest.DescriptorType == 0x22)
    {
     T<<"获取报告描述符完成\n";
     //获取缓冲区地址
```

```
    pBuf = (UCHAR * )pUrb - >UrbControlDescriptorRequest.TransferBuffer;
    //获取传输的长度
    Len = pUrb - >UrbControlDescriptorRequest.TransferBufferLength;
    if(Len == 65)  //说明报告描述符长度正确,因为我们的键盘报告描述符为 65 字节
    {
     //原来的报告描述符第四字节为 0x06,表示集合用于键盘,这里改成 0xFF,表示自定义
     T<<"报告描述符第 4 字节被修改为 0xFF\n";
     pBuf[3] = 0xFF;
     //原来的报告描述符第 62 字节为 0x03,表示附加字段的大小为 3 位,
     //上面已经定义了 5 位,共 8 位,即 1 字节。现在需要再增加 7 字节,
     //即增加 7×8 = 56 位,加上原来的 3 位后为 59 位,因此修改 pBuf[61]为 59
     pBuf[61] = 59;                          //这样整个输出报告就是 8 字节的了
     T<<"报告描述符第 62 字节被修改为 59\n";
    }
   }
   }
  break;

  //如果是批量或中断传输
  case URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER:  //值为 0x09
   //如果为输入请求(TransferFlags 的最低位为传输方向)
   if((pUrb - >UrbBulkOrInterruptTransfer.TransferFlags)&0x01)
   {
    T<<"批量或中断读数据完成\n";
    //获取缓冲区地址
    pBuf = (UCHAR * )pUrb - >UrbBulkOrInterruptTransfer.TransferBuffer;
    //获取传输的长度
    Len = pUrb - >UrbBulkOrInterruptTransfer.TransferBufferLength;
    if(Len == 8)  //如果返回的是 8 字节数据
    {
     unsigned int i;
     //用来保存按键的状态
     UCHAR KeyStatus;
     //在 USB 键盘程序中,KEY1、KEY2、KEY3 刚好对应着报告第一字节的低 3 位,
     //所以这里可以直接将 pBuf[0]的低 3 位赋值给 KeyStatus
     KeyStatus = (pBuf[0])&(0x07);
     //第二字节(即 pBuf[1])为保留字节,剩下的 6 个字节(pBuf[2]~pBuf[7])为 6 个键
码,当某个
     //键按下时会出现对应的键码,KEY4 对应的键码为 0x59,KEY5 对应的键码为 0x5A,
     //KEY6 对应的键码为 0x5B,KEY7 对应的键码为 0x39,KEY8 对应的键码为 0x53
     //在这 6 个字节中查找是否有对应的键码出现就知道该键是否被按下了
     for(i = 2;i<8;i ++ )
```

```
            {
             if(pBuf[i] == 0x59)KeyStatus| = 0x08;                    //KEY4
             if(pBuf[i] == 0x5A)KeyStatus| = 0x10;                    //KEY5
             if(pBuf[i] == 0x5B)KeyStatus| = 0x20;                    //KEY6
             if(pBuf[i] == 0x39)KeyStatus| = 0x40;                    //KEY7
             if(pBuf[i] == 0x53)KeyStatus| = 0x80;                    //KEY8
            }
            pBuf[0] = KeyStatus;                                      //设置按键情况
            MyCount ++ ;                                              //计数器加 1
            //pBuf[1]~pBuf[4]为计数器值
            pBuf[1] = (UCHAR)(MyCount&0xFF);                          //最低字节
            pBuf[2] = (UCHAR)((MyCount>>8)&0xFF);                     //次低字节
            pBuf[3] = (UCHAR)((MyCount>>16)&0xFF);                    //次高字节
            pBuf[4] = (UCHAR)((MyCount>>24)&0xFF);                    //最高字节
            //pBuf[5]~[7]设置为 0
            pBuf[5] = 0;
            pBuf[6] = 0;
            pBuf[7] = 0;
            T<<"修改输入报告完毕\n";
           }
          }
          break;
          default:
          break;
         }
        }
        return status;
    }
```

至此，该过滤驱动就算改好了，不算太难吧？然后编译程序，就可以得到一个驱动程序文件 MyUsbLowerFilter.sys。下面介绍如何安装这个过滤驱动。

10.4　过滤驱动的安装

在使用向导创建驱动时，选择了创建安装用的 DLL 工程，但是 DLL 的调试不像普通应用程序调试那么方便，所以可以参考该安装用的 DLL 工程里的代码，自己写一个应用程序来安装过滤驱动。不过，向导产生的安装程序比较简单，只是简单地根据设备描述来判断需要安装过滤驱动的设备，并增加服务。实际上，为了准确地安装到某个设备上，应该通过 GUID 查找指定设备类下的硬件 ID 来决定是否需要安装过滤驱动。另外，由于该过滤驱动修改了报告描述符应用集合的用途，导致最终产生

的设备不一样(原来为键盘设备,安装过滤驱动后变成了 HID 兼容设备),需要先将原来的驱动卸载掉,然后才能安装过滤驱动;否则系统会不知道这个情况,还是使用以前的驱动,也就是说,并不会出现新的硬件。如果这时强行将设备拔下,还会导致系统死机。安装过滤驱动后还需要重新启动设备,主机才会识别到新硬件。

就在当前 Workspace 下新增一个 FilterInstallerOfComputer00 的工程(选择为基于对话框的模板),放置两个按钮,一个为安装过滤驱动,另外一个为卸载过滤驱动。

安装过滤驱动的步骤如下:

(1) 复制过滤驱动文件 MyUsbLowerFilter.sys 到当前系统的 Windows\System32\Drivers 目录下。可以调用 GetWindowsDirectory()函数获取操作系统下的 Windows 目录,然后生成目标路径,接着使用 CopyFile()函数复制文件。注意在 VC 环境下调试时,复制文件会提示不成功,这时可以手动复制一次文件或者直接运行一次程序以复制文件,文件只需要复制一次就可以了,除非驱动文件有改动。注意过滤驱动文件 MyUsbLowerFilter.sys 的版本,有发行版和调试版,我们分别将它们复制到 FilterInstallerOfComputer00 工程下的调试目录和发行目录。当然,如果是在调试驱动,总这样复制也不方便,可以直接在驱动程序工程设置中指定输出目标文件的路径到 FilterInstallerOfComputer00 工程下的对应目录中。

(2) 卸载旧的键盘驱动。通过查找所有键盘设备,查看是否有指定硬件 ID 的设备,如果有,则卸载它。这里使用键盘的安装类 GUID 来查找设备,那么首先应该要找到键盘的安装类 GUID。去哪儿找呢? 在注册表中可以找到。将带有 USB 键盘程序的 USB 学习板连上,然后展开注册表中的 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\HID(可按 F5 键刷新注册表),找到下面的 Vid_8888&Pid_0002 并展开它,可以看到下面有很多子键,分别点击每个子键,然后查看右边的表项。检查名称为 Class 的数据是否为 Keyboard,如果是,就可以看到它的 ClassGUID 为{4D36E96B-E325-11CE-BFC1-08002BE10318};然后再看 HardwareID,肯定有一个是与我们设备相匹配的,即 VID 为 8888,PID 为 0002,版本为 0100 的 HardwareID"HID\Vid_8888&Pid_0002&Rev_0100"。通过列举该 GUID 的所有已连接的设备,判断 HardwareID 是否为"HID\Vid_8888&Pid_0002&Rev_0100",即可确认是否为指定的设备。使用 SetupDiGetClassDevs()函数可以获取指定类的设备集合信息,其中参数 GUID 可以通过最后的 Flags 参数指定为安装类 GUID 还是接口类 GUID,这里使用的是安装类 GUID,别搞错了。然后调用 SetupDiEnumDeviceInfo()函数列举所有该集合中的设备,并使用 SetupDiGetDeviceRegistryProperty()函数获取 HardwareID。获取到 HardwareID 后,判断是否为我们设备的 HardwareID"HID\Vid_8888&Pid_0002&Rev_0100"。如果是,就调用 SetupDiCallClassInstaller()函数删除这个设备。

(3) 添加服务。调用 OpenSCManager()函数打开 SCManager 数据库,然后调

用 CreateService()函数增加 MyUsbLowerFilter 服务。

(4) 更新设备过滤驱动服务名称列表。我们的过滤驱动是安装在键盘设备下面那个 HID 设备上的,过滤驱动应该增加在这个设备上。在注册表中可以找到这个设备的信息,展开注册表中的"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB"子键,在下面找到 Vid_8888&Pid_0002 的子键并展开它,可以找到一个"2008-07-12"的子键(它就是我们在固件程序中指定的设备序列号),选中它。然后在右边就可以看到它的设备类为 HIDClass,安装 ClassGUID 为{745A17A0-74D3-11D0-B6FE-00A0C90F57DA},HardwareID 为"USB\Vid_8888&Pid_0002&Rev_0100"。与步骤(2)中的查找方式类似,使用 GUID 和硬件 ID 来找到这个设备。找到后就可以将 MyUsbLowerFilter 增加到下层过滤驱动服务名称列表中(当然如果已经存在就不用再增加了,这个需要自己程序判断)。成功更新后可在右边看到增加了一个名称为 LowerFilters 的表项,数据为 MyUsbLowerFilter。这就是告诉系统该设备将使用服务名为 MyUsbLowerFilter 的下层过滤驱动器。如果没有看到,可以使用 F5 键刷新一下。设置服务名称列表的函数为 SetupDiSetDeviceRegistryProperty()。

(5) 重新启动设备。使用 SetupDiSetClassInstallParams 及 SetupDiCallClassInstaller()函数来停止和启动设备。安装过滤驱动后要能够自动发现新设备,必须要重新启动设备。

具体实现的代码较长,这里就不再给出了,读者可以参看光盘中 MyUsbLowerFilter 目录下的 FilterInstallerOfComputer00 工程,主要在 FilterInstallerOfComputer00Dlg.cpp 文件中。

10.5　过滤驱动的卸载

同过滤驱动的安装一样,卸载过滤驱动后导致了新设备的产生,所以需要先卸载掉原来的旧驱动(这次需要卸载的旧驱动就是 HID 兼容设备了);接着再更新过滤驱动服务名称列表,最后删除服务,并重新启动设备。其详细步骤如下:

(1) 卸载旧的 HID 兼容设备的驱动。由于该过滤驱动安装之后,会变成一个 HID 兼容设备,卸载过滤驱动之前要先卸载这个 HID 兼容设备。跟安装过滤驱动的步骤(2)类似,要先找到该设备的 GUID 和硬件 ID。同样在 Enum\HID\Vid_8888&Pid_0002 下可以找到该 HID 兼容设备,设备类为 HIDClass,设备描述 DeviceDesc 为 HID-compliant device,ClassGUID 为{745A17A0-74D3-11D0-B6FE-00A0C90F57DA},硬件 ID 为"HID\Vid_8888&Pid_0002&Rev_0100"。用该 GUID 和硬件 ID 去搜索设备,如果找到,则卸载该设备。

(2) 更新设备过滤驱动服务名称列表。与安装过滤驱动时的步骤(4)刚好相反,这里是将 MyUsbLowerFilter 从过滤驱动服务名称列表中移除,告诉系统该设备不再使用

该服务作为下层过滤驱动了。成功更新后，可以看到名称为 LowerFilters 的表项中不再有前面指定的服务了。

（3）删除服务。调用 DeleteService()函数将原来新增的 MyUsbLowerFilter 服务删除。

（4）重新启动设备。同安装过滤驱动时的步骤(5)。

图 10.5.1～10.5.3 分别为 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\Vid_8888&Pid_0002\2008-07-12 子键右边表项安装过滤驱动之前、安装过滤驱动之后和卸载过滤驱动之后的变化。

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
Capabilities	REG_DWORD	0x00000094 (148)
Class	REG_SZ	HIDClass
ClassGUID	REG_SZ	{745A17A0-74D3-11D0-B6FE-
CompatibleIDs	REG_MULTI_SZ	USB\Class_03&SubClass_01
ConfigFlags	REG_DWORD	0x00000000 (0)
DeviceDesc	REG_SZ	USB 人体学输入设备
Driver	REG_SZ	{745A17A0-74D3-11D0-B6FE-
HardwareID	REG_MULTI_SZ	USB\Vid_8888&Pid_0002&Re
LocationInformation	REG_SZ	USB Device
Mfg	REG_SZ	(标准系统设备)
ParentIdPrefix	REG_SZ	7&2824ea3b&3
Service	REG_SZ	HidUsb
UINumber	REG_DWORD	0x00000000 (0)

图 10.5.1　安装过滤驱动之前

从图 10.5.2 中可以看出，安装过滤驱动之后，新增了一个名称为 LowerFilters（下层过滤器），数据为 MyUsbLowerFilter 的表项。而卸载过滤驱动之后，LowerFilters 表项中的数据被设置成了空，即将 MyUsbLowerFilter 删除了。如果原来就安装过其他下层过滤器，在这里也可以看到，卸载 MyUsbLowerFilter 并不会影响它们，这些都是在安装程序中实现的。

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
Capabilities	REG_DWORD	0x00000094 (148)
Class	REG_SZ	HIDClass
ClassGUID	REG_SZ	{745A17A0-74D3-11D0-B6FE-
CompatibleIDs	REG_MULTI_SZ	USB\Class_03&SubClass_01
ConfigFlags	REG_DWORD	0x00000000 (0)
DeviceDesc	REG_SZ	USB 人体学输入设备
Driver	REG_SZ	{745A17A0-74D3-11D0-B6FE-
HardwareID	REG_MULTI_SZ	USB\Vid_8888&Pid_0002&Re
LocationInformation	REG_SZ	USB Device
LowerFilters	REG_MULTI_SZ	MyUsbLowerFilter
Mfg	REG_SZ	(标准系统设备)
ParentIdPrefix	REG_SZ	7&2824ea3b&3
Service	REG_SZ	HidUsb
UINumber	REG_DWORD	0x00000000 (0)

注意这里的改变

图 10.5.2　安装过滤驱动之后

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
Capabilities	REG_DWORD	0x00000094 (148)
Class	REG_SZ	HIDClass
ClassGUID	REG_SZ	{745A17A0-74D3-11D0-B6FE-
CompatibleIDs	REG_MULTI_SZ	USB\Class_03&SubClass_01&
ConfigFlags	REG_DWORD	0x00000000 (0)
DeviceDesc	REG_SZ	USB 人体学输入设备
Driver	REG_SZ	{745A17A0-74D3-11D0-B6FE-
HardwareID	REG_MULTI_SZ	USB\Vid_8888&Pid_0002&Rev
LocationInformation	REG_SZ	USB Device
LowerFilters	REG_MULTI_SZ	
Mfg	REG_SZ	(标准系统设备)
ParentIdPrefix	REG_SZ	7&2824ea3b&3
Service	REG_SZ	HidUsb
UINumber	REG_DWORD	0x00000000 (0)

注意这里的改变

图 10.5.3　卸载过滤驱动之后

10.6　驱动程序测试

图 10.6.1 为安装过滤驱动时显示的信息，图 10.6.2 为卸载过滤驱动时显示的信息。注意被安装过滤驱动的设备是一个“USB 人体学输入设备”，它的硬件 ID 开头为 USB。而原来的键盘是由该“USB 人体学输入设备”产生的，它的硬件 ID 以 HID 开头。安装过滤驱动之后的“HID 兼容设备”也是由这个“USB 人体学输入设备”产生的，硬件 ID 也是以 HID 开头。“USB 人体学输入设备”和“HID 兼容设备”的安装类 GUID 都为{745A17A0 - 74D3 - 11D0 - B6FE - 00A0C90F57DA}，通过硬件 ID 可以区分它们。而键盘设备的安装类 GUID 为{4D36E96B - E325 - 11CE - BFC1 - 08002BE10318}，可以与“HID 兼容设备”区分开来。

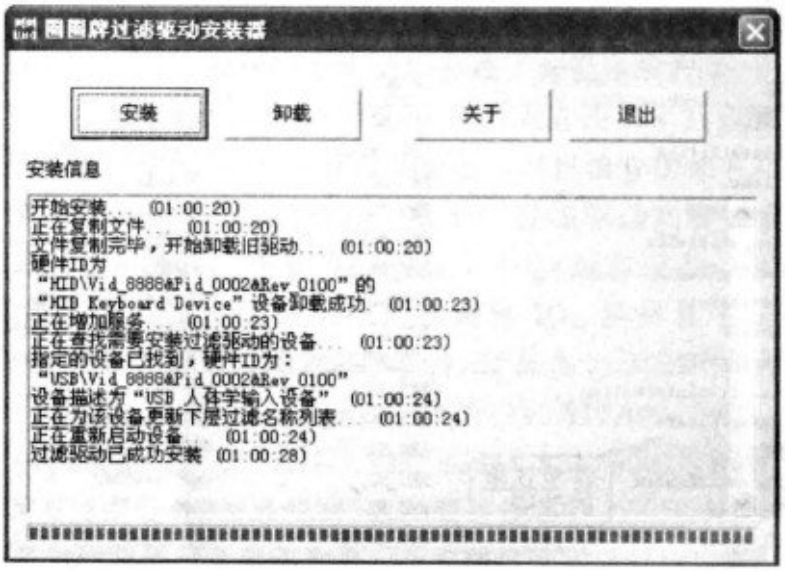

图 10.6.1　安装过滤驱动

图 10.6.2　卸载过滤驱动

连接 USB 学习板，启动第 5 章中的自定义 HID 设备测试软件（MyUsbHidTestApp），将 PID 修改为 0002（记得一定要修改，否则会找不到设备），单击"打开设备"按钮，提示设备已经找到，但是"读访问打开设备失败"。这就是因为该 HID 设备已经被系统独占了，所以无法访问。然后单击"关闭设备"按钮，再安装过滤驱动。等过滤驱动安装完毕之后，再单击"打开设备"按钮，这时设备就被成功打开了，并且应用程序能够控制板上的 LED，也能够正常显示板上按键的情况了。

打开设备管理器，可以看到在安装过滤驱动之后，原来在键盘类下的 HID Keyboard Device不见了，而在"USB 人体学输入设备"下增加了一个新的 HID compliant device 设备。

使用 BUS Hound 捕捉"USB 人体学输入设备"以及"HID-compliant device"的数据，结果如图 10.6.3 所示。其中 Device 为 27.1 的为"USB 人体学输入设备"，Device 为 29 的是我们的测试软件操作的"HID-compliant device"。前两行数据是单击应用程序上 LED1 时产生的数据，这两行数据的顺序颠倒一下看起来更清楚些。第二行的数据就是应用程序发出给 HID-compliant device 的数据，第 1 字节 00 为报告 ID，第二字节 01 表示 LED1 亮。这些数据由"HID-compliant device"提交给"USB 人体学输入设备"之后，输出就变成一字节的 01 了，这正是我们键盘程序所需要接收的 1 字节数据。如果没有过滤驱动修改，"USB 人体学输入设备"输出的数据应该是去

Device	Phase	Data
27.1	DO	01
29	DO	00 01 00 00 00 00 00 00 00
27.1	DI	00 00 53 00 00 00 00 00
29	DI	00 80 01 00 00 00 00 00 00
27.1	DI	00 00 00 00 00 00 00 00
29	DI	00 00 02 00 00 00 00 00 00

图 10.6.3　BUS Hound 捕捉到的数据

掉报告 ID 之后剩下的 8 字节数据。而三四行是按下 KEY8 时捕捉到的数据，五、六行是松开 KEY8 时捕捉到的数据。第三行数据中第三字节为 53，它正是键盘程序中 KEY8 返回的键码值，经过过滤驱动之后，将第四行数据中第二字节的最高位设置成了 1，表示 KEY8 被按下了。同时，后面增加了计数器值 01。第五行数据全为 00，表示所有按键都已经释放，经过过滤驱动之后，第六行第一字节就为 00，同时后面增加了计数器值 02。总之，过滤驱动将键盘输入的数据格式翻译成了应用程序所需要的格式，将应用程序输出的数据格式翻译成了键盘所需要的输出数据格式。

如果安装的是调试版的驱动程序，那么就可以使用 DriverMonitor 来显示在驱动中输出的调试信息了，如图 10.6.4 所示。启动 DriverMonitor，打开需要显示调试信息的驱动文件，这时就能在 DriverMonitor 的信息窗口中显示出调试信息。图 10.6.4 所显示的是安装过滤驱动、控制 LED 和按动开关时的部分信息。为了让读者方便阅读，圈圈将它们各个阶段划分开来，并在右边增加了注释。显示的第一阶段为捕捉到请求报告描述符的请求，这时需要设置完成函数。第二阶段就是报告描述符返回后，进入完成函数之后的处理，对返回的报告描述符进行了修改。第三、四阶段是检测到读取数据的请求，需要设置完成函数。第五阶段显示检测到一个输出数据的请求（由操作 LED 引起的），这时只需要修改输出的数据长度，不用设置完成函数。第六阶段显示有数据返回（按动了学习板上的按键），从而进入了设置的完成函数，在这里对返回数据进行了修改。当数据成功返回后，上层程序又一次请求数据输入（第七阶段）。

图 10.6.4　使用 DriverMonitor 显示驱动的调试信息

调试版的驱动通常只在调试阶段使用，实际使用的应该是发行版的驱动。调试版和发行版的驱动文件分别位于 driver\objchk\i386 目录和 driver\objfre\i386 目录下，要分别编译。安装时要将驱动文件和安装程序放在同一个目录下。

10.7　本章小结

本章通过一个 USB 下层过滤驱动的实例介绍了如何构造和安装一个过滤驱动程序，并在光盘中给出了完整的代码。该过滤驱动程序通过修改设备返回的报告描述符以及设备通信的数据，将原来的 USB 键盘设备“改装”成了 HID 兼容设备。读者可以发挥自己的想象，改出更好玩的东西来，例如，在键盘上增加鼠标功能；把键盘改成鼠标；把普通键盘改成多媒体键盘（可以控制音量、播放等）；屏蔽某些键等，爱怎么玩就怎么玩，想怎么玩就怎么玩，总之过滤驱动是个好东西。