内核中执行代码后需要将结果动态显示给应用层的用户，`DeviceIoControl`是直接发送控制代码到指定的设备驱动程序，使相应的移动设备以执行相应的操作的函数，如下代码是一个经典的驱动开发模板框架，在开发经典驱动时会用到的一个通用案例。

如下这段案例中，我们只介绍`DispatchIoctl`，该函数主要用于接收或发送`IOCTL`控制信号，以此来实现接收和发送数据。

首先，通过调用`IoGetCurrentIrpStackLocation`函数获取`IRP（I/O Request Packet）`的堆栈位置，并获取控制码、输入和输出缓冲区的相关信息。

然后，根据不同的控制信号码，进行相应的处理流程。这里的示例代码只列出了一个控制信号码`（IOCTL_IO_LyShark）`的处理流程，即接收传入数据，进行处理后返回传出数据。

最后，根据处理结果，设置`IRP`的状态和返回信息，并通过调用`IoCompleteRequest`函数完成`IRP`的处理。

需要注意的是，这段代码中没有对错误情况进行处理，如果出现错误，返回的状态码为`STATUS_INVALID_DEVICE_REQUEST`。因此，实际使用中需要根据具体情况进行修改和完善。

驱动程序开发通用模板代码如下:

```
#include <ntifs.h>
#include <windef.h>

// 控制器
#define IOCTL_IO_LyShark CTL_CODE(FILE_DEVICE_UNKNOWN,0x800,METHOD_BUFFERED,FILE_ANY_ACCESS)

// 卸载驱动执行
VOID UnDriver(PDRIVER_OBJECT pDriver)
{
    PDEVICE_OBJECT pDev;                                        // 用来取得要删除设备对象
    UNICODE_STRING SymLinkName;                                 // 局部变量symLinkName
    pDev = pDriver->DeviceObject;
    IoDeleteDevice(pDev);                                       // 调用IoDeleteDevice用于删除
设备
    RtlInitUnicodeString(&SymLinkName, L"\\??\\LySharkDriver");     // 初始化字符串将
symLinkName定义成需要删除的符号链接名称
    IoDeleteSymbolicLink(&SymLinkName);                         // 调用IoDeleteSymbolicLink删
除符号链接
    DbgPrint("驱动卸载完毕...");
}

// 创建设备连接
NTSTATUS CreateDriverObject(IN PDRIVER_OBJECT pDriver)
{
    NTSTATUS Status;
    PDEVICE_OBJECT pDevObj;
    UNICODE_STRING DriverName;
    UNICODE_STRING SymLinkName;

    // 创建设备名称字符串
    RtlInitUnicodeString(&DriverName, L"\\Device\\LySharkDriver");
    Status = IoCreateDevice(pDriver, 0, &DriverName, FILE_DEVICE_UNKNOWN, 0, TRUE,
&pDevObj);
```

```
    // 指定通信方式为缓冲区
    pDevObj->Flags |= DO_BUFFERED_IO;

    // 创建符号链接
    RtlInitUnicodeString(&SymLinkName, L"\\??\\LySharkDriver");
    Status = IoCreateSymbolicLink(&SymLinkName, &DriverName);
    return STATUS_SUCCESS;
}

// 创建回调函数
NTSTATUS DispatchCreate(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    pIrp->IoStatus.Status = STATUS_SUCCESS;          // 返回成功
    DbgPrint("派遣函数 IRP_MJ_CREATE 执行 \n");
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);        // 指示完成此IRP
    return STATUS_SUCCESS;                           // 返回成功
}

// 关闭回调函数
NTSTATUS DispatchClose(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    pIrp->IoStatus.Status = STATUS_SUCCESS;          // 返回成功
    DbgPrint("派遣函数 IRP_MJ_CLOSE 执行 \n");
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);        // 指示完成此IRP
    return STATUS_SUCCESS;                           // 返回成功
}

// 主控制器,用于判断R3发送的控制信号
NTSTATUS DispatchIoctl(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    NTSTATUS status = STATUS_INVALID_DEVICE_REQUEST;
    PIO_STACK_LOCATION pIrpStack;
    ULONG uIoControlCode;
    PVOID pIoBuffer;
    ULONG uInSize;
    ULONG uOutSize;

    // 获得IRP里的关键数据
    pIrpStack = IoGetCurrentIrpStackLocation(pIrp);

    // 获取控制码
    uIoControlCode = pIrpStack->Parameters.DeviceIoControl.IoControlCode;

    // 输入和输出的缓冲区（DeviceIoControl的InBuffer和OutBuffer都是它）
    pIoBuffer = pIrp->AssociatedIrp.SystemBuffer;

    // EXE发送传入数据的BUFFER长度（DeviceIoControl的nInBufferSize）
    uInSize = pIrpStack->Parameters.DeviceIoControl.InputBufferLength;

    // EXE接收传出数据的BUFFER长度（DeviceIoControl的nOutBufferSize）
    uOutSize = pIrpStack->Parameters.DeviceIoControl.OutputBufferLength;

    // 对不同控制信号的处理流程
```

```
    switch (uIoControlCode)
    {
        // 接收或发送
    case IOCTL_IO_LyShark:
    {
        DWORD dw = 0;

        // 得到输入参数
        memcpy(&dw, pIoBuffer, sizeof(DWORD));

        DbgPrint("[+] hello lyshark \n");

        // 对输入参数进行处理
        dw++;

        // 设置输出参数
        memcpy(pIoBuffer, &dw, sizeof(DWORD));

        // 返回通信状态
        status = STATUS_SUCCESS;
        break;
    }

    pIrp->IoStatus.Status = status;
    pIrp->IoStatus.Information = uOutSize;
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);
    return status;
    }

    // 设定DeviceIoControl的*lpBytesReturned的值（如果通信失败则返回0长度）
    if (status == STATUS_SUCCESS)
        pIrp->IoStatus.Information = uOutSize;
    else
        pIrp->IoStatus.Information = 0;

    // 设定DeviceIoControl的返回值是成功还是失败
    pIrp->IoStatus.Status = status;
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);
    return status;
}

// 入口函数
NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING RegistryPath)
{
    // 调用创建设备
    CreateDriverObject(pDriver);

    pDriver->DriverUnload = UnDriver;                                   // 卸载函数
    pDriver->MajorFunction[IRP_MJ_CREATE] = DispatchCreate;             // 创建派遣函数
    pDriver->MajorFunction[IRP_MJ_CLOSE] = DispatchClose;               // 关闭派遣函数
    pDriver->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DispatchIoctl;  // 分发函数

    DbgPrint("By:LyShark ...");
```

```
    return STATUS_SUCCESS;
}
```

而在应用层中，最需要注意的是函数`DeviceIoControl`它是`Windows API`中的一个系统函数其工作于应用层，用于与设备驱动程序通信。该函数可以向设备驱动程序发送控制码，同时传输输入和输出缓冲区的数据。以下是该函数的语法：

```
BOOL DeviceIoControl(
  HANDLE       hDevice,
  DWORD        dwIoControlCode,
  LPVOID       lpInBuffer,
  DWORD        nInBufferSize,
  LPVOID       lpOutBuffer,
  DWORD        nOutBufferSize,
  LPDWORD      lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

参数解释：

- hDevice：要访问的设备的句柄，可以使用CreateFile函数获得。
- dwIoControlCode：设备驱动程序定义的控制码。
- lpInBuffer：输入缓冲区的指针，用于向设备驱动程序传递输入数据。
- nInBufferSize：输入缓冲区的大小（以字节为单位）。
- lpOutBuffer：输出缓冲区的指针，用于从设备驱动程序获取输出数据。
- nOutBufferSize：输出缓冲区的大小（以字节为单位）。
- lpBytesReturned：返回的字节数指针，用于记录成功传输的字节数。
- lpOverlapped：用于异步I/O操作的指针。

在驱动程序中，要实现`DeviceIoControl`函数的处理，需要在IRP的`IoControlCode`中获取控制码，并在`AssociatedIrp.SystemBuffer`中获取输入数据，将处理结果放回到同一个`SystemBuffer`中，然后将`SystemBuffer`中的数据作为输出数据传回给用户层。具体处理过程可以参考示例代码中的`DispatchIoctl`函数。

总的来说，`DeviceIoControl`函数提供了一种方便的方法，可以让用户层应用程序与设备驱动程序进行通信，实现设备的控制、配置和数据传输等功能。

应用层通用测试模板代码如下：

```
#include <iostream>
#include <Windows.h>
#include <winioctl.h>

#define IOCTL_IO_LyShark CTL_CODE(FILE_DEVICE_UNKNOWN,0x800,METHOD_BUFFERED,FILE_ANY_ACCESS)

int main(int argc, char *argv[])
{
    HANDLE hDevice = CreateFileA("\\\\.\\LySharkDriver", GENERIC_READ | GENERIC_WRITE, 0,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
    if (hDevice == INVALID_HANDLE_VALUE)
    {
        CloseHandle(hDevice);
        return 0;
    }

    // 发送控制信号

    // input = 发送数据 output = 接受数据 ref_len = 数据长度
    DWORD input = 100, output = 0, ref_len = 0;
    DeviceIoControl(hDevice, IOCTL_IO_LyShark, &input, sizeof(input), &output,
sizeof(output), &ref_len, 0);

    printf("输出: %d \n", output);

    system("pause");
    CloseHandle(hDevice);
    return 0;
}
```

输出效果如下: