内核层与应用层之间的数据交互是必不可少的部分，只有内核中的参数可以传递给用户数据才有意义，一般驱动多数情况下会使用`SystemBuf`缓冲区进行通信，也可以直接使用网络套接字实现通信，如下将简单介绍通过SystemBuf实现的内核层与应用层通信机制。

内核与应用层传递结构体，实现应用层用户传入一个结构体到内核，内核处理后返回一段字符串。

内核代码如下，代码已经备注。

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

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

// 通信结构体
typedef struct Hread
{
    ULONG Flage;
    ULONG Addr;
    ULONG WriteBufferAddr;
    ULONG Size;
    ULONG Pid;
}_Hread, *PtrHread;

typedef struct _DEVICE_EXTENSION
{
    UNICODE_STRING SymLinkName;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;

// 驱动关闭提示
VOID DriverUnload(PDRIVER_OBJECT pDriverObject)
{
    PDEVICE_OBJECT pDevObj;
    pDevObj = pDriverObject->DeviceObject;
    PDEVICE_EXTENSION pDevExt = (PDEVICE_EXTENSION)pDevObj->DeviceExtension;
    UNICODE_STRING pLinkName = pDevExt->SymLinkName;

    IoDeleteSymbolicLink(&pLinkName);
    IoDeleteDevice(pDevObj);
}

// 默认派遣
NTSTATUS DefDispatchRoutine(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    NTSTATUS status = STATUS_SUCCESS;
    pIrp->IoStatus.Status = status;
    pIrp->IoStatus.Information = 0;
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);
    return status;
}

// 主派遣函数
NTSTATUS IoctlDispatchRoutine(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
```

```
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    ULONG_PTR Informaiton = 0;
    PVOID InputData = NULL;
    ULONG InputDataLength = 0;
    PVOID OutputData = NULL;
    ULONG OutputDataLength = 0;
    PIO_STACK_LOCATION  IoStackLocation = IoGetCurrentIrpStackLocation(pIrp);             //
Irp堆栈
    InputData = pIrp->AssociatedIrp.SystemBuffer;                                         //
输入堆栈
    OutputData = pIrp->AssociatedIrp.SystemBuffer;                                        //
输出堆栈
    InputDataLength = IoStackLocation->Parameters.DeviceIoControl.InputBufferLength;      //
输入数据大小
    OutputDataLength = IoStackLocation->Parameters.DeviceIoControl.OutputBufferLength;    //
输出数据大小
    ULONG Code = IoStackLocation->Parameters.DeviceIoControl.IoControlCode;               //
控制码

    switch (Code)
    {
    case My_Code:
        {
            PtrHread PtrBuff = (PtrHread)InputData;
            ULONG RetFlage = PtrBuff->Flage;
            ULONG RetAddr = PtrBuff->Addr;
            ULONG RetBufferAddr = PtrBuff->WriteBufferAddr;
            ULONG Size = PtrBuff->Size;
            ULONG Pid = PtrBuff->Pid;

            DbgPrint("读取文件标志：%d", RetFlage);
            DbgPrint("读取写入地址：%x", RetAddr);
            DbgPrint("读取缓冲区大小：%d", RetBufferAddr);
            DbgPrint("读取当前大小：%d", Size);
            DbgPrint("要操作进程PID: %d", Pid);

            // 通过内存返回数据.
            char *retBuffer = "hello lyshark";
            memcpy(OutputData, retBuffer, strlen(retBuffer));
            Informaiton = strlen(retBuffer) + 1;
            Status = STATUS_SUCCESS;

            // 通过内存返回数据,另一种通信方式.
            /*
            PVOID addr = (PVOID)"ok";
            RtlCopyMemory(OutputData, addr, 4);
            Informaiton = 4;
            Status = STATUS_SUCCESS;
            */
            break;
        }
    }
```

```
    pIrp->IoStatus.Status = Status;                    // 设置IRP完成状态，会设置用户模式下的
GetLastError
    pIrp->IoStatus.Information = Informaiton;          // 设置操作的字节
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);          // 完成IRP，不增加优先级
    return Status;
}

// 驱动入口
NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING pRegistryPath)
{
    pDriverObject->DriverUnload = DriverUnload;                          // 注册驱动卸载函数
    pDriverObject->MajorFunction[IRP_MJ_CREATE] = DefDispatchRoutine;    // 注册派遣函数
    pDriverObject->MajorFunction[IRP_MJ_CLOSE] = DefDispatchRoutine;
    pDriverObject->MajorFunction[IRP_MJ_WRITE] = DefDispatchRoutine;
    pDriverObject->MajorFunction[IRP_MJ_READ] = DefDispatchRoutine;
    pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IoctlDispatchRoutine;

    NTSTATUS status;
    PDEVICE_OBJECT pDevObj;
    PDEVICE_EXTENSION pDevExt;

    // 创建设备名称的字符串
    UNICODE_STRING devName;
    RtlInitUnicodeString(&devName, L"\\Device\\MyDevice");

    // 创建设备
    status = IoCreateDevice(pDriverObject, sizeof(DEVICE_EXTENSION), &devName,
FILE_DEVICE_UNKNOWN, 0, TRUE, &pDevObj);
    pDevObj->Flags |= DO_BUFFERED_IO;                        // 将设备设置为缓冲I/O设备
    pDevExt = (PDEVICE_EXTENSION)pDevObj->DeviceExtension; // 得到设备扩展

    // 创建符号链接
    UNICODE_STRING symLinkName;
    RtlInitUnicodeString(&symLinkName, L"\\??\\MyDevice");
    pDevExt->SymLinkName = symLinkName;
    status = IoCreateSymbolicLink(&symLinkName, &devName);
    return STATUS_SUCCESS;
}
```

客户端代码中只需要通过`DeviceIoControl()`发送控制信号即可，需要注意驱动需要安装并运行起来，否则无法获取到数据。

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

// 自定义的控制信号
#define My_Code CTL_CODE(FILE_DEVICE_UNKNOWN,0x800,METHOD_BUFFERED,FILE_ANY_ACCESS)

// 通信结构体
typedef struct Hread
{
    ULONG Flage;
```

```
    ULONG Addr;
    ULONG WriteBufferAddr;
    ULONG Size;
    ULONG Pid;
}_Hread, *PtrHread;

int main(int argc, char* argv[])
{
    // 创建
    HANDLE handle = CreateFileA("\\\\\.\\MyDevice", GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    unsigned char RetBufferData[20] = { 0 };
    DWORD ReturnLength = 4;
    _Hread buf;

    buf.Flage = 2;
    buf.Addr = 0x401234;
    buf.WriteBufferAddr = 1024;
    buf.Size = 100;
    buf.Pid = 2566;

    DeviceIoControl(handle, My_Code, &buf, 20, (LPVOID)RetBufferData, 4, &ReturnLength, 0);

    for (size_t i = 0; i < 20; i++)
    {
        printf("返回数据: %d \n", RetBufferData[i]);
    }

    CloseHandle(handle);

    getchar();
    return 0;
}
```

运行这段代码我们看下返回效果：