# 驱动程序与应用程序的通信离不开派遣函数，派遣函数是Windows驱动编程中的重要概念，一般情况下 驱动程序负责处理I/O特权请求，而大部分IO的处理请求是在派遣函数中处理的，当用户请求数据时，操 作系统会提前处理好请求，并将其派遣到指定的内核函数中执行，接下来将详细说明派遣函数的使用并 通过派遣函数读取Shadow SSDT中的内容。

先来简单介绍一下 输入输出请求包，该请求包在Windows内核中是一个

IRP(I/O Request Package)

非常重要的数据结构，当我们的上层应用与底层的驱动程序通信时，应用程序就会发出I/O请求，操作系 统将该请求转化为相应的IRP数据，然后会根据不同的请求数据将请求派遣到相应的驱动函数中执行，这 一点有点类似于Windows的消息机制。

**简单的驱动通信：** 注册两个派遣函数，当设备创建的时候触发，以及关闭时触发。

#include <ntddk.h>

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"\\??\\My\_Driver"); // 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("删除设备与符号链接成功...");

}

NTSTATUS DispatchCreate(PDEVICE\_OBJECT pDevObj, PIRP pIrp)

{

pIrp->IoStatus.Status = STATUS\_SUCCESS; // 返回成功DbgPrint(" 派 遣 函 数 IRP\_MJ\_CREATE 成 功 执 行 !\n"); IoCompleteRequest(pIrp, IO\_NO\_INCREMENT); // 指示完成此IRP return STATUS\_SUCCESS; // 返回成功

}

NTSTATUS DispatchClose(PDEVICE\_OBJECT pDevObj, PIRP pIrp)

{

pIrp->IoStatus.Status = STATUS\_SUCCESS; // 返回成功

DbgPrint(" 派 遣 函 数 IRP\_MJ\_CLOSE 成 功 执 行 !\n"); IoCompleteRequest(pIrp, IO\_NO\_INCREMENT); // 指示完成此IRP return STATUS\_SUCCESS; // 返回成功

}

NTSTATUS CreateDriverObject(IN PDRIVER\_OBJECT pDriver)

{

NTSTATUS Status; PDEVICE\_OBJECT pDevObj;

UNICODE\_STRING DriverName;

UNICODE\_STRING SymLinkName;

RtlInitUnicodeString(&DriverName, L"\\Device\\My\_Device");

Status = IoCreateDevice(pDriver, 0, &DriverName, FILE\_DEVICE\_UNKNOWN, 0, TRUE, &pDevObj);

DbgPrint("命令 IoCreateDevice 状态: %d", Status);

// DO\_BUFFERED\_IO 设置读写方式 Flags的三个不同的值分别为：DO\_BUFFERED\_IO、

DO\_DIRECT\_IO和0

pDevObj->Flags |= DO\_BUFFERED\_IO; RtlInitUnicodeString(&SymLinkName, L"\\??\\My\_Device"); Status = IoCreateSymbolicLink(&SymLinkName, &DriverName);

DbgPrint("当前命令IoCreateSymbolicLink状态: %d", Status);

return STATUS\_SUCCESS;

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING RegistryPath)

{

CreateDriverObject(pDriver); // 调用创建设备子过程

// 注册两个派遣函数,分别对应创建与关闭,派遣函数名可自定义

pDriver->MajorFunction[IRP\_MJ\_CREATE] = DispatchCreate; // 创建成功派遣函数

pDriver->MajorFunction[IRP\_MJ\_CLOSE] = DispatchClose; // 关闭派遣函数

DbgPrint(" 驱 动 加 载 完 成 ..."); pDriver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 客户端代码

#include <windows.h> #include <stdio.h> #include <winioctl.h>

int main()

{

HANDLE hDevice = CreateFile(L"\\\\.\\My\_Device", GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hDevice == INVALID\_HANDLE\_VALUE) //判断hDevice返回值是否为空

{

printf("获取驱动句柄失败!错误: %d\n", GetLastError()); getchar();

}

getchar(); CloseHandle(hDevice); return 0;

}

**读取驱动中的数据：** 实现读取内核缓冲区中的数据，并打印出来。

#include <ntddk.h>

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"\\??\\My\_Driver"); // 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("删除设备与符号链接成功...");

}

NTSTATUS DispatchCreate(PDEVICE\_OBJECT pDevObj, PIRP pIrp)

{

pIrp->IoStatus.Status = STATUS\_SUCCESS; // 返回成功

DbgPrint(" 派 遣 函 数 IRP\_MJ\_CREATE 成 功 执 行 !\n"); IoCompleteRequest(pIrp, IO\_NO\_INCREMENT); // 指示完成此IRP return STATUS\_SUCCESS; // 返回成功

}

NTSTATUS DispatchClose(PDEVICE\_OBJECT pDevObj, PIRP pIrp)

{

pIrp->IoStatus.Status = STATUS\_SUCCESS; // 返回成功

DbgPrint(" 派 遣 函 数 IRP\_MJ\_CLOSE 成 功 执 行 !\n"); IoCompleteRequest(pIrp, IO\_NO\_INCREMENT); // 指示完成此IRP return STATUS\_SUCCESS; // 返回成功

}

NTSTATUS DispatchRead(PDEVICE\_OBJECT pDevObj, PIRP pIrp)

{

NTSTATUS Status = STATUS\_SUCCESS;

PIO\_STACK\_LOCATION Stack = IoGetCurrentIrpStackLocation(pIrp); ULONG ulReadLength = Stack->Parameters.Read.Length;

pIrp->IoStatus.Status = Status;

pIrp->IoStatus.Information = ulReadLength;

DbgPrint("应用要读取的长度：%d\n", ulReadLength);

// 将内核中的缓冲区全部填充为0x68 方便演示读取的效果

memset(pIrp->AssociatedIrp.SystemBuffer, 0x68, ulReadLength); IoCompleteRequest(pIrp, IO\_NO\_INCREMENT);

return Status;

}

NTSTATUS CreateDriverObject(IN PDRIVER\_OBJECT pDriver)

{

NTSTATUS Status; PDEVICE\_OBJECT pDevObj;

UNICODE\_STRING DriverName;

UNICODE\_STRING SymLinkName;

RtlInitUnicodeString(&DriverName, L"\\Device\\My\_Device");

Status = IoCreateDevice(pDriver, 0, &DriverName, FILE\_DEVICE\_UNKNOWN, 0, TRUE, &pDevObj);

DbgPrint("命令 IoCreateDevice 状态: %d", Status);

pDevObj->Flags |= DO\_BUFFERED\_IO; RtlInitUnicodeString(&SymLinkName, L"\\??\\My\_Device"); Status = IoCreateSymbolicLink(&SymLinkName, &DriverName);

DbgPrint("当前命令IoCreateSymbolicLink状态: %d", Status);

return STATUS\_SUCCESS;

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING RegistryPath)

{

CreateDriverObject(pDriver); // 调用创建设备pDriver->MajorFunction[IRP\_MJ\_CREATE] = DispatchCreate; // 创建成功派遣函数pDriver->MajorFunction[IRP\_MJ\_CLOSE] = DispatchClose; // 关闭派遣函数pDriver->MajorFunction[IRP\_MJ\_READ] = DispatchRead;

DbgPrint(" 驱 动 加 载 完 成 ..."); pDriver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 客户端代码

#include <windows.h> #include <stdio.h> #include <winioctl.h>

int main()

{

HANDLE hDevice = CreateFile(L"\\\\.\\My\_Device", GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hDevice == INVALID\_HANDLE\_VALUE)

{

printf("获取驱动句柄失败: %d\n", GetLastError()); getchar();

}

UCHAR buffer[10]; ULONG ulRead;

ReadFile(hDevice, buffer, 10, &ulRead, 0); for (int i = 0; i < (int)ulRead; i++)

{

printf("%02X", buffer[i]);

}

getchar(); CloseHandle(hDevice); return 0;

}

本书作者： 王瑞 (LyShark)

作者邮箱： m [e@lyshark.com](mailto:e@lyshark.com)

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com