驱动开发

1. 初始驱动编程

用户空间的代码不能直接访问内核空间，在内核空间的代码可以执行特权指令。由用户空间访问内核空间，必须通过一定的接口

我们编写驱动程序，并不能直接运行，是通过加载到内核空间，成为操作系统的一部分，为应用程序提供支持的一个模块

根据功能的不同，将驱动开发分为两类

(1) 仅仅作为windows操作系统的扩展，而并非时使得一个硬件工作起来，我们将其称为内核程序。

(2) 针对某一种硬件，使得其能够很好的工作起来，通常称之为驱动程序

根据开发框架的不同我们将驱动分为三类

1 NT 老式的驱动程序模型，适用于所有Windows系统，程序开发者可以编写 一个完全不支持硬件工作的驱动程序，却可以将代码运行在内核模式中

加载NT驱动一般分为4个步骤

(1) 调用OpenSCManager打开SCM管理器

(2) 调用OpenService打开此项服务

(3) 调用ControlService传递SERVICE_CONTROL_STOP来停止服务

(4) 调用DeleteService卸载此项服务

(5) 关闭句柄

2 WDM 不是通过服务，修改注册表项启动的，增加了即插即用的功能，需要安装 的时候提供一个inf文件进行配合

3 WDF WDM面向过程，WDF面向对象，对函数进行了封装

2.编写一个简单的驱动程序

```
#include <Ntifs.h>  // Ntifs.h包含Ntddk.h,Ntddk.h又包含Wdm.h

Void DriverUnload(PDRIVER_OBJECT driver)

{

// 避免编译器报未引用参数的警告

UNREFERENCED_PARAMETER(driver);

//什么也并不做
```

```
DbgPrint(“卸载”);//kdprint也可以
}
NTSTATUS DriverEntry(PDRIVER_OBJECT driver, PUNICODE_STRING path)
{
UNREFERENCED_PARAMETER(path);
_asm int 3;       //断点以便调试
DbgPrint(“First Driver”);
driver->DriverUnload = DriverUnload;
return STATUS_SUCCESS;
}
```

3.驱动加载（通过加载服务的方式加载驱动）

(1)使用OpenSCManager函数打开SCM

(2)利用SCM句柄和OpenService服务句柄，使用CreateService函数创建一个服务

(3)使用StartService函数启动创建的服务

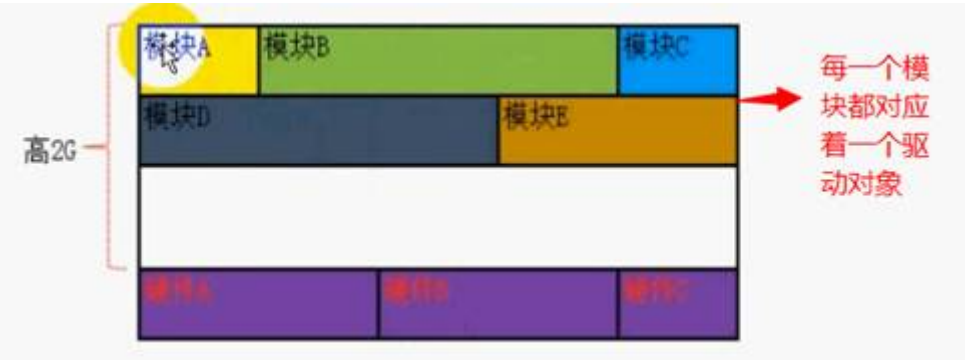
(4)QueryServiceState获取服务状态，使服务状态 == SERVICE_STOP

(5)DeleteService卸载服务

4.内核编程基础

(1)驱动对象

①当驱动一加载，对象管理器便会生成一个驱动对象，并调用DriverEntry，将驱动对象传入。



DriverSection--->LDR_DATA_TABLE_ENTRY 链表结构，此结构体存储着驱动的很多信息，同时也圈着所有驱动对象，

遍历模块时，第一个为自己的模块，第二个为0，从第三个开始是下一个模块的驱动信息

(2)设备对象同样作为一个结构体

①DriverObject：指出设备对象归属于哪个驱动程序

②NextDevice：下一个设备对象（一个驱动程序可以创建多个设备对象），这个设备对象是同一层的

③AttachedDevice：指向下一层驱动程序的设备对象（可以理解被挂载的设备对象）

④CurrentIrp：使用Irp串行化时很重要，用来决策当前IRP完成还是挂起等

⑤StackSize：设备栈的个数

⑥DeviceExtension：指向LDR链的指针

(3)IRP（I/O Request Package）输入输出请求包

①消息被封装成IRP结构体

1.创建设备对象

```
//创建设备名称
UNICODE_STRING Devicename;
RtlInitUnicodeString(&Devicename,L"\\Device\\MyDevice");

//创建设备
IoCreateDevice(
pDriver,                    //当前设备所属的驱动对象
0,
&Devicename,                //设备对象的名称
FILE_DEVICE_UNKNOWN,
FILE_DEVICE_SECURE_OPEN,
FALSE,
&pDeviceObj                 //设备对象指针
);
```

2.设置数据交互方式

```
pDeviceObj->Flags |= DO_BUFFERED_IO;
```

缓冲区方式读写(DO_BUFFERED_IO)：操作系统将应用程序提供缓冲区的数据复制到内核模式下的地址中。
直接方式读写(DO_DIRECT_IO) ：操作系统会将用户模式下的缓冲区锁住。然后操作系统将这段缓冲区在内核模式地址再次映射一遍。这样，用户模式的缓冲区和内核模式的缓冲区指向的是同一区域的物理内存。缺点就是要单独占用物理页面。
其他方式读写(在调用IoCreateDevice创建设备后对pDevObj->Flags即不设置DO_BUFFERED_IO也不设置DO_DIRECT_IO此时就是其他方式) ：
在使用其他方式读写设备时，派遣函数直接读写应用程序提供的缓冲区地址。在驱动程序中，直接操作应用程序的缓冲区地址是很危险的。只有驱动程序与应用程序运行在相同线程上下文的情况下，才能使用这种方式。

3.创建符号链接名称，创建个名字给3环用

```
//创建符号链接
IoCreateSymbolicLink(&SymbolicLinkName,&Devicename);
```

特别说明：

1、设备名称的作用是给内核对象用的，如果要在Ring3访问，必须要有符号链接其实就是一个别名，没有这个别名，在Ring3不可见。

2、内核模式下，符号链接是以“\??\”开头的，如C 盘就是“\??\C:”

3、而在用户模式下，则是以“\\.\”开头的，如C 盘就是“\\.\C:

4.设置分发函数和卸载函数

<1> 当应用层通过CreateFile，ReadFile，WriteFile，CloseHandle等函数打开、从设备读取数据、向设备写入数据、关闭设备的时候，会使操作系统产生出IRP_MJ_CREATE，IRP_MJ_READ，IRP_MJ_WRITE，IRP_MJ_CLOSE等不同的IRP。

<2> 其他类型的IRP

IRP类型	发送
IRP_MJ_DEVICE_CONTROL	DeviceControl函数会产生此IRP
IRP_MJ_POWER	在操作系统处理电源消息时，产生一次IRP

驱动对象的最后一个成员，将派遣函数根据IRP类型进行赋值

```
//设置分发函数和卸载函数
pDriver->MajorFunction[IRP_MJ_CREATE] = IrpCreateProc;
pDriver->MajorFunction[IRP_MJ_CLOSE] = IrpCloseProc;
pDriver->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IrpDeviceControlProc;
pDriver->DriverUnload = DriverUnload;
```

5.填写派遣函数的格式

```
//派遣函数的格式：

NTSTATUS MyDispatchFunction(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
	//处理自己的业务...

	//设置返回状态
	pIrp->IoStatus.Status = STATUS_SUCCESS;  //  getlasterror()得到的就是这个值
	pIrp->IoStatus.Information = 0;                  //  返回给3环多少数据 没有填0
	IoCompleteRequest(pIrp, IO_NO_INCREMENT);
	return STATUS_SUCCESS;
}
```

6.3环 CreateFile （符号链接名）

7.通过IODeviceControl获取0环的数据进行读写