第八章 设备管理

- 8.1设备管理概念
- 8.2 Spooling系统
- 8.3.1 Linux模块机制
- 8.3.2 Linux驱动程序
- 8.3.3 Windows驱动程序

《操作系统原理》

8.3.3 Windows驱动程序

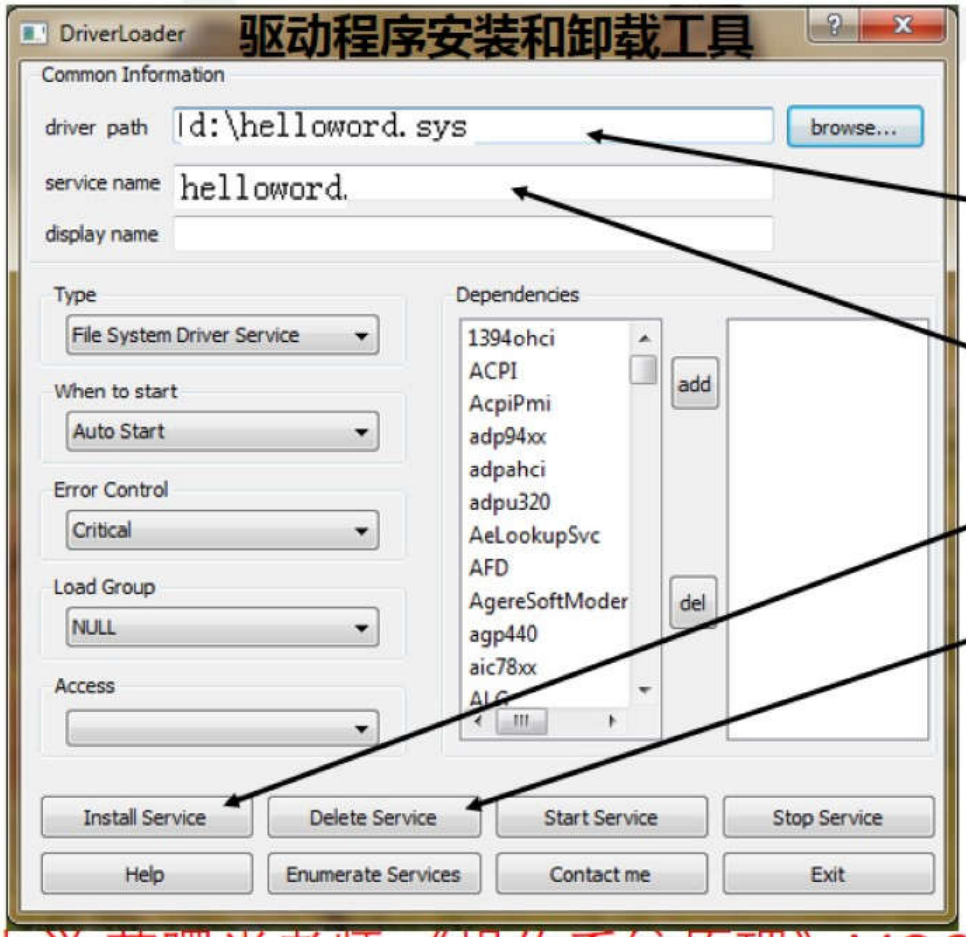
教师：苏曙光

华中科技大学软件学院

最简单的windows驱动——Hello world

```
#include <ntddk.h>

NTSTATUS DriverEntry (
                        IN PDRIVER_OBJECT pDriverObject,
                        IN PUNICODE_STRING pRegistryPath )
{
        NTSTATUS status = STATUS_SUCCESS;
        KdPrint((  “hello world!\n"));
        return status;
}
```

运行和测试



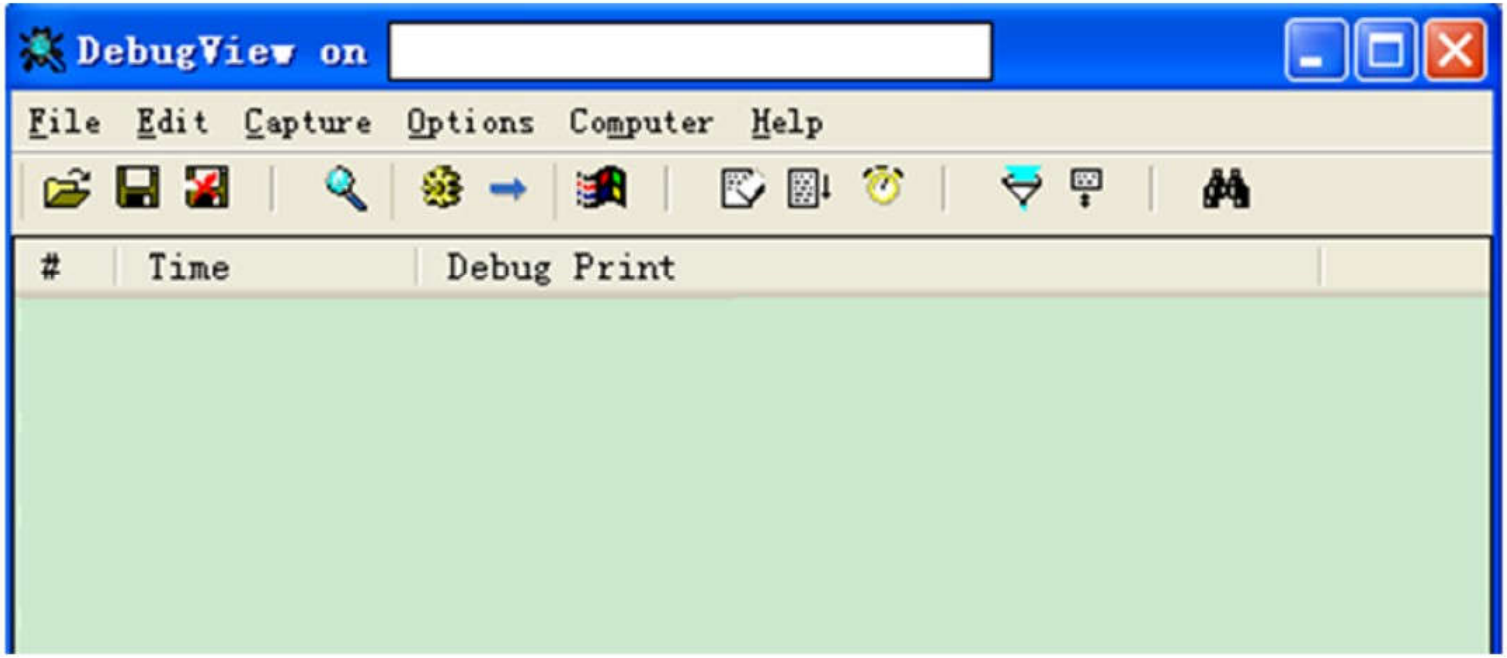
选择驱动程序(*.sys)

填写服务名（任意）

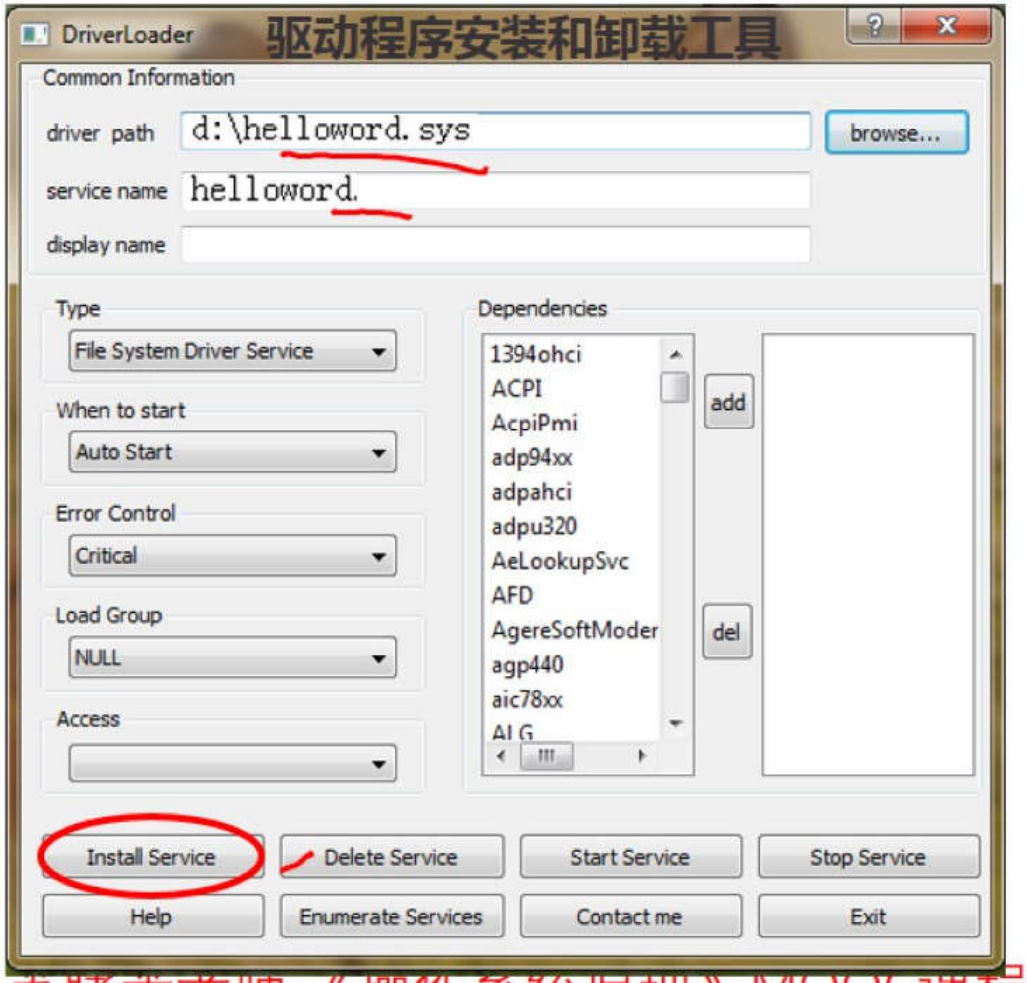
安装驱动程序

卸载驱动程序

DebugView内核开发的辅助调试工具
作用：显示内核缓冲区的信息

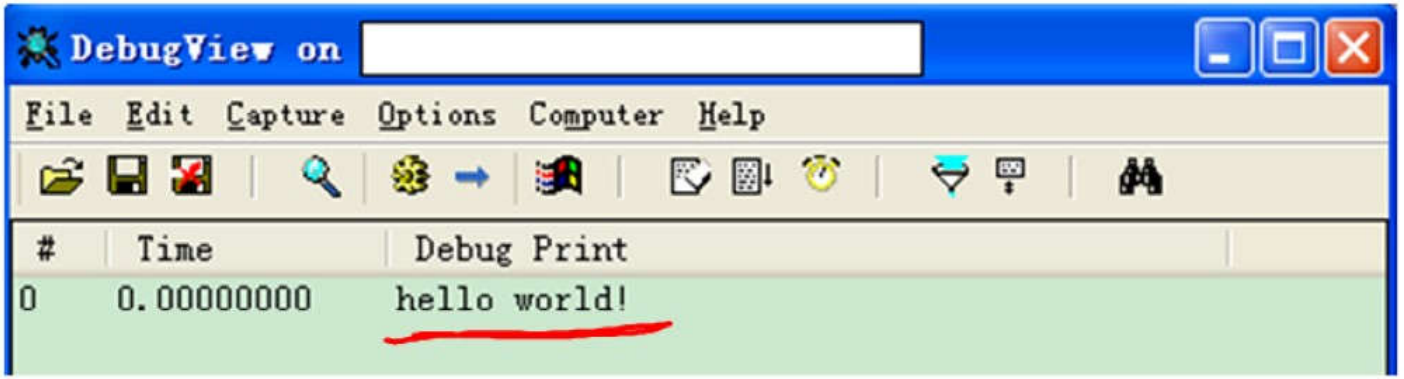


运行和测试

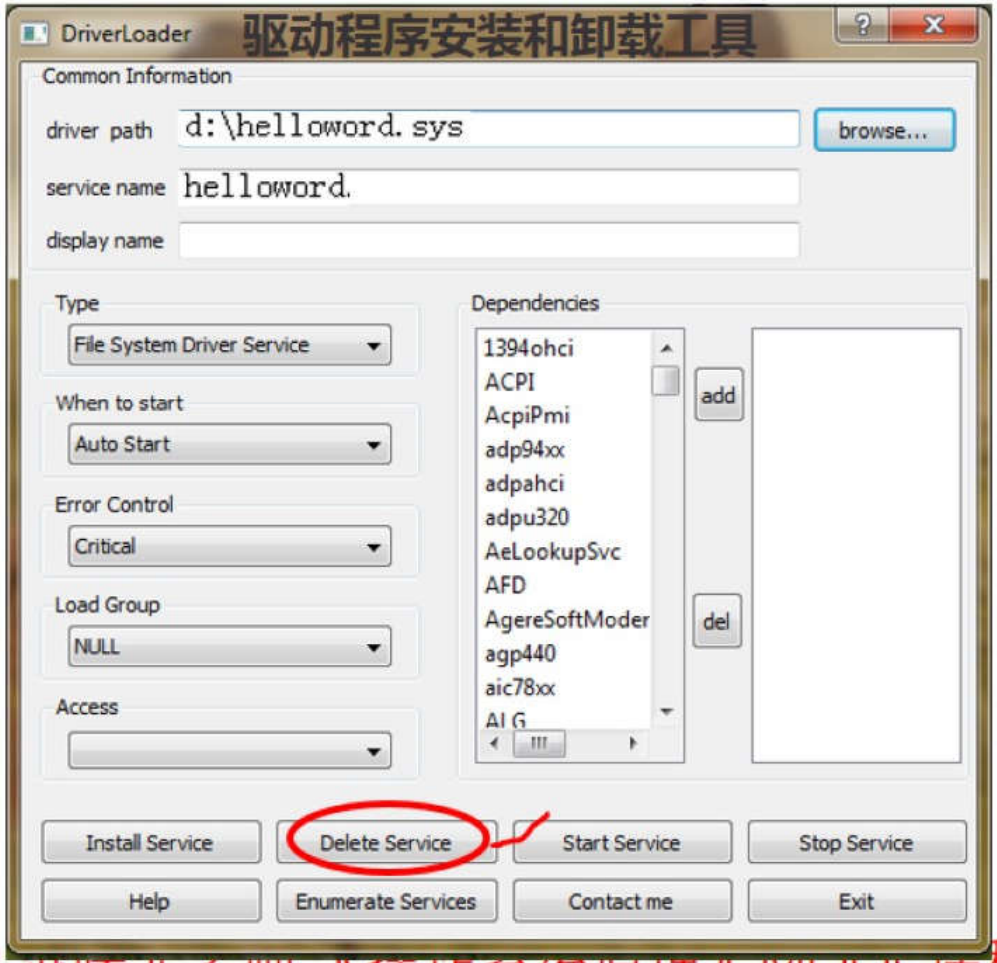


运行和测试

- 点击“Install Service”安装驱动时

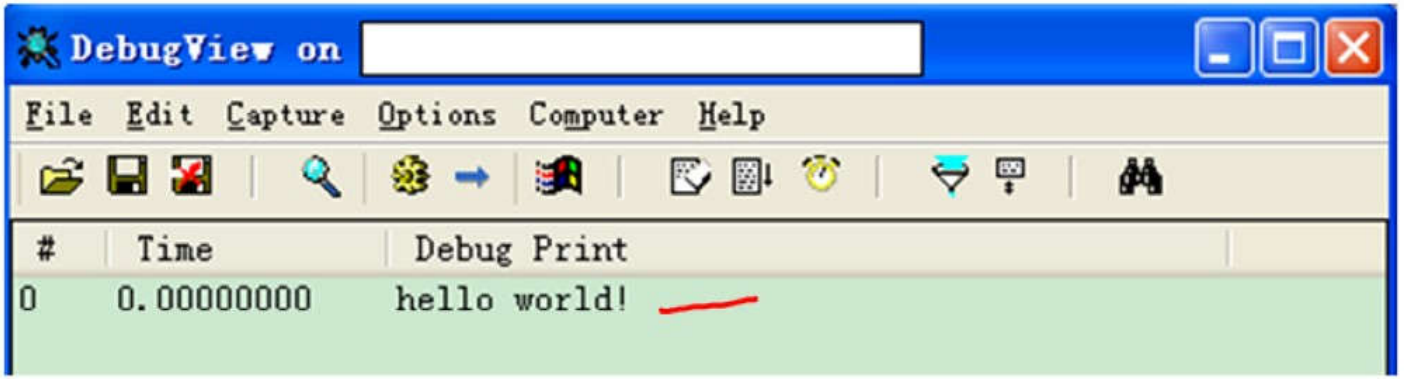


运行和测试



运行和测试

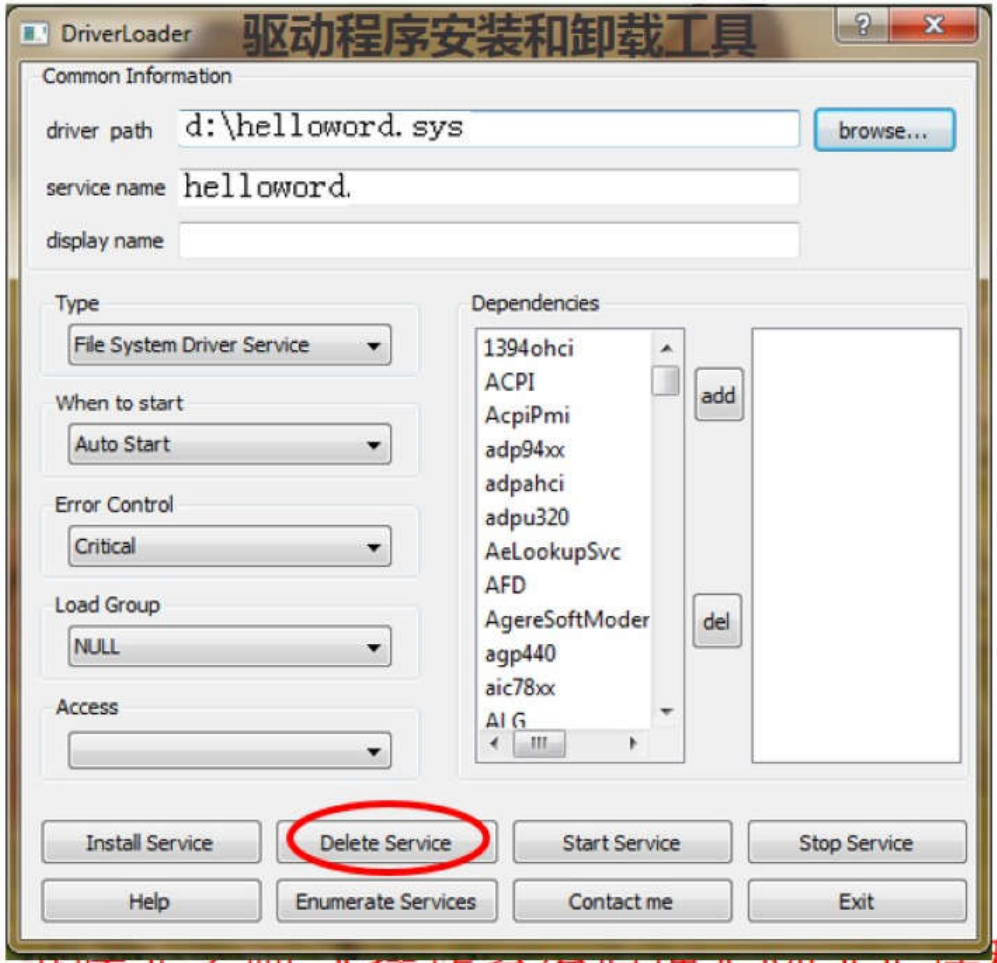
- 点击“Delete Service”卸载驱动时



改写：最简单的windows驱动——Hello world

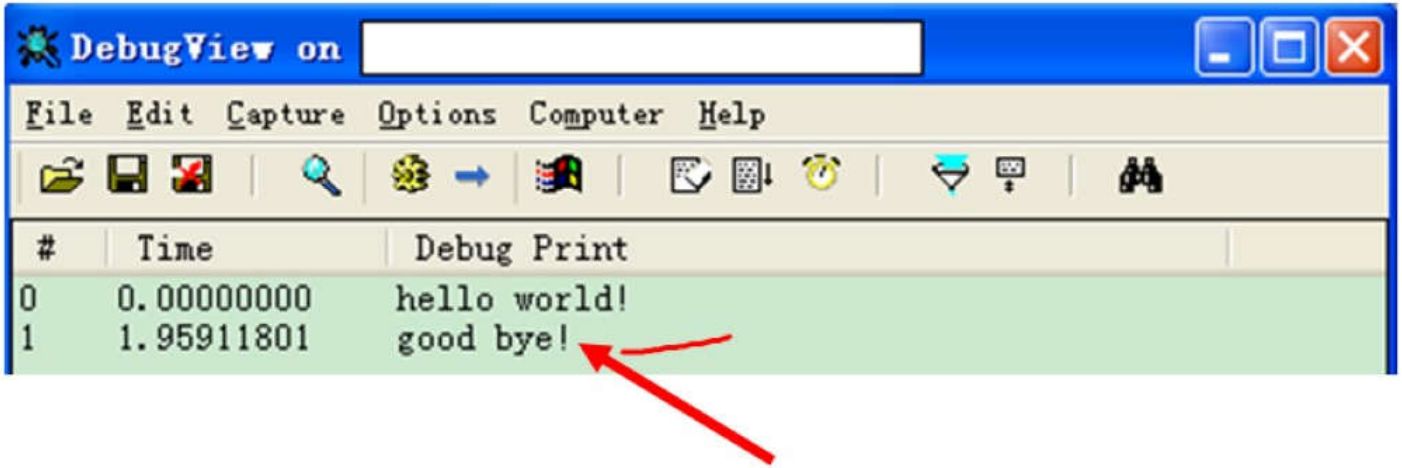
```
#include <ntddk.h>
VOID ExitDriver(IN PDRIVER_OBJECT pDriverObject);

NTSTATUS DriverEntry (
                    IN PDRIVER_OBJECT pDriverObject,
                    IN PUNICODE_STRING pRegistryPath )
{
    NTSTATUS status = STATUS_SUCCESS;
    pDriverObject->DriverUnload = ExitDriver;
    KdPrint(( "hello world!\n"));
    return status;
}

VOID ExitDriver(IN PDRIVER_OBJECT pDriverObject)
{
    KdPrint(( "good bye!\n"));
}
```

运行和测试



运行和测试

点击“Delete Service”卸载驱动时



Windows声明驱动的入口/退出函数

- Windows固定采用DriverEntry作为入口函数；
- 给pDriverObject->DriverUnload赋值退出函数指针。

Linux声明驱动的入口/退出函数

```
//向Linux系统声明入口函数
module_init(my_init);
//向Linux系统声明退出函数
module_exit(my_exit);
```

增加创建/读/写/关闭等其他功能函数

```
#include <ntddk.h>
VOID ExitDriver(IN PDRIVER_OBJECT pDriverObject);

NTSTATUS DriverEntry (
                    IN PDRIVER_OBJECT pDriverObject,
                    IN PUNICODE_STRING pRegistryPath )
{
    NTSTATUS status = STATUS_SUCCESS;
    pDriverObject->DriverUnload = ExitDriver;
    KdPrint(( "hello world!\n"));
    return status;
}

VOID ExitDriver(IN PDRIVER_OBJECT pDriverObject)
{
    KdPrint(( "good bye!\n"));
}
```

第二个简单的驱动程序

```
#include <ntddk.h>
VOID ExitDriver (IN PDEVICE_OBJECT pDevObj);
NTSTATUS HelloCreate(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
NTSTATUS HelloWrite(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
NTSTATUS HelloRead(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
NTSTATUS HelloClose(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
NTSTATUS DriverEntry (   IN PDRIVER_OBJECT pDriverObject,
                         IN PUNICODE_STRING pRegistryPath  )
{
        NTSTATUS status = STATUS_SUCCESS;

        pDriverObject->DriverUnload = ExitDriver;
        pDriverObject->MajorFunction[IRP_MJ_CREATE] = HelloCreate;
        pDriverObject->MajorFunction[IRP_MJ_WRITE] = HelloWrite;
        pDriverObject->MajorFunction[IRP_MJ_READ] = HelloRead;
        pDriverObject->MajorFunction[IRP_MJ_CLOSE] = HelloClose;
        KdPrint( "hello world!\n");
        return status;
}
```

第二个简单的驱动程序

```
VOID ExitDriver(IN PDRIVER_OBJECT pDriverObject)
{
        KdPrint(( "good bye!\n"));
}

NTSTATUS HelloCreate(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
{
        KdPrint(( "Create Device\n" ));
        return status;
}

NTSTATUS HelloWrite(IN PDEVICE_OBJECT pDevObj, IN PIRP pIrp)
{
        KdPrint(( "Write Device\n" ));
        return status;
}
```

Linux声明设备的文件操作函数

```
static struct file_operations my_fops = {
        open:   my_open,
        write:  my_write,
        release: my_release
};
```

Windows声明设备的文件操作函数

- pDriverObject->MajorFunction[IRP_MJ_CREATE] = ?;
- pDriverObject->MajorFunction[IRP_MJ_CLOSE] = ?;
- pDriverObject->MajorFunction[IRP_MJ_WRITE] = ?;
- pDriverObject->MajorFunction[.....] = ?;

Windows声明设备的文件操作函数

应用程序的函数	*MajorFunction[IRP主功能代码]*
CreateFile	IRP_MJ_CREATE
ReadFile	IRP_MJ_READ
WriteFile	IRP_MJ_WRITE
DeviceIoControl	IRP_MJ_DEVICE_CONTROL
CloseHandle	IRP_MJ_CLOSE/CLEANUP

驱动程序开发工具

- Windows DDK（ Driver Development Kit ）
- DriverWorks + Windows DDK
- WinDriver

工具	开发效率	执行效率
DDK（Driver Development Kit）	低	高
Driver Works + DDK	较高	较高
WinDriver	高	低

DDK中的内容

- 运行库：runtime Library
- 文档：帮助文档
- 编译器：C/C++编译器\链接器
- 调试\分析工具：内核调试工具、分析工具
- 与内核API函数相关的头文件(如ddk.h, wdm.h等)
- 与内核API函数相关的库文件(wdm.lib等)

编写驱动时不能调用的函数

- Windows用户模式API函数；
- ISO C/C++标准函数库

驱动程序与与应用程序的连接方式

- 驱动程序中创建设备对象并命名
- 应用程序通过符号链接访问设备对象

创建设备对象并命名

- 利用内核API函数 IoCreateDevice创建设备对象

```
NTSTATUS   IoCreateDevice(
    IN PDRIVER_OBJECT  DriverObject,   //对应的驱动对象
    IN ULONG  DeviceExtensionSize,    //设备扩展区的大小
    IN PUNICODE_STRING  DeviceName,  //设备名
    IN DEVICE_TYPE  DeviceType,   //设备类型
    IN ULONG  DeviceCharacteristics,  //设备对象特征
    IN BOOLEAN  Exclusive,  // 一般设为FALSE
    OUT PDEVICE_OBJECT  *DeviceObject //OS创建的设备对象
 );
```

创建设备对象并命名——

- IoCreateDevice( ) 第3个参数是设备名

```
UNICODE_STRING devname;
RtlInitUnicodeString(&devname, "\\Device\\ Hello ");
IoCreateDevice (DriverObject, sizeof(DEVICE_EXTENSION),
                &devname,  ...);
```

- 内核可以通过设备名访问\Device\目录下面的设备。
- 应用程序对\Device\目录没有访问权
  - 创建符号链接给应用程序访问\Device\中的设备。

应用程序通过符号链接访问设备对象

- 创建符号链接让应用程序访问\Device\中的设备
  - 在内核\??目录中创建符号链接指向\Device\中的设备。
  - 创建符号链接的函数：*IoCreateSymbolicLink( )*

```
UNICODE_STRING devname;
UNICODE_STRING linkname;
RtlInitUnicodeString(&devname, "\\Device\\Hello");
RtlInitUnicodeString(&linkname,  "\\??\\MyHello");
IoCreateDevice(DriverObject, sizeof(DEVICE_EXTENSION),
                    &devname,  ...);
IoCreateSymbolicLink (&linkname, &devname);
```

应用程序通过符号链接访问设备

- 用户模式下内核的\??子目录叫做\\.\子目录
- 用户模式下通过访问\\.\下的符号链接访问设备

```
hDevice = CreateFile("\\\\.\\HelloDDK",
                     GENERIC_WRITE|GENERIC_READ,
                     FILE_SHARE_WRITE | FILE_SHARE_READ,
                     NULL,
                     OPEN_EXISTING,
                     0,
                     NULL);
ReadFile( hDevice,  lpBuffer, ……..);
WriteFile(hDevice,  lpBuffer, ……..);
……..
```