

# 浅谈Windows内核编程

主 讲 人： 黄凡玲  
学 院： 计算机与电子信息学院  
专 业： 信息安全  
班 级： 2011级（1） 班

# 主要内容：

- A. 简单的打印字符串程序
- B. Windows层次结构
- C. 底层驱动设备栈框架、核心概念解释
- D. 两个框架——传统型、Minifilter
  - 1. 串口过滤驱动，传统型
  - 2. 禁用Notepad.exe, Minifilter
- E. 透明加密过滤驱动，简单异或加密
- F. 入门心得
- G. 资料分享

# 简单的打印字符串程序

# 简单的打印字符串程序

## 准备工作

- 1、WDK：可以在：<http://www.microsoft.com/en-us/download/details.aspx?id=11800>，下载到，注意最好选择完整安装
- 2、Windbg：完整晚装WDK后可以在其安装目录的\7600.16385.1\Debuggers下找到。
- 3、DebugView：可以在网上搜索到资源，用于查看内核程序的输出。
- 4、srvinstw：软件也可以在网上搜索到资源，用于windows系统的服务安装与移除。

## 编译你需要两个文件：

Makefile 可以从WDK示例代码中拷贝一个，基本不变

### SOURCES

```
TARGETNAME=test1      ;最后编译完后生成的目标文件的名称，不包括扩展名。  
TARGETTYPE=DRIVER     ;生成的文件的类型  
TARGETPATH=obj        ;  
SOURCES=test1.c       ;指定要被编译的文件
```

欲了解更多，可咨询百度，^\_^

# 简单的打印字符串程序

## 内核中的main函数——DriverEntry

```
#include "ntddk.h"
//提供一个Unload函数只是为了让这个程序能动态卸载，方便调试
VOID DriverUnload(PDRIVER_OBJECT driver)
{
    DbgPrint("firstFilter: our driver is unloading...\r\n");
}
//DriverEntry，入口函数，相当于main函数
NTSTATUS DriverEntry(
    PDRIVER_OBJECT driver,
    PUNICODE_STRING reg_path
)
{
    //这是内核模块的入口，可以在这里写入我们想写的东西
    DbgPrint("firstFilter: Hello everyone, welcome to Technical Sharon!\r\n");

    //设置一个卸载函数，便于函数退出
    driver->DriverUnload = DriverUnload;
    return STATUS_SUCCESS;
}
```

# 简单的打印字符串程序

## Dos命令

启动服务: `net start 服务名`

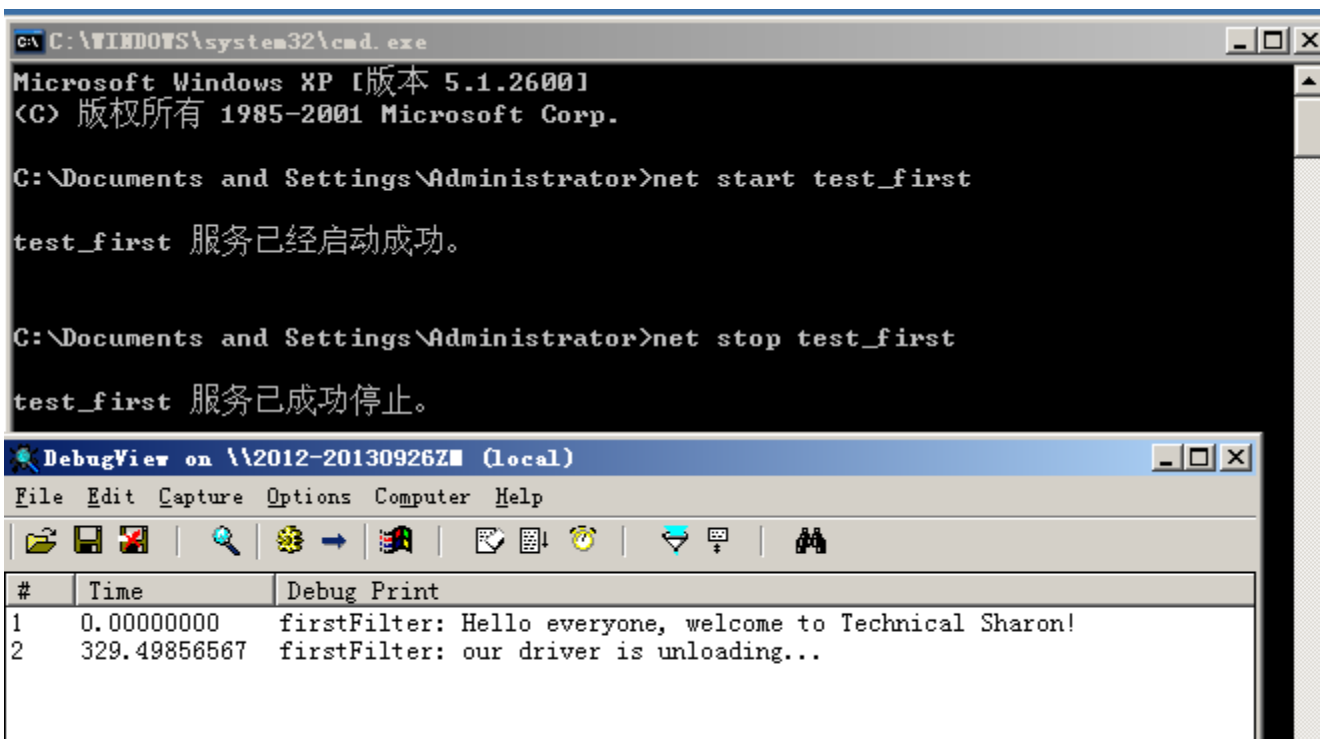
停止服务: `net stop 服务名`

注: DebugView设置, "Capture"—》"Capture Kernel"

编译得到: `first.sys`

```
E:\Windows\first>build
path contains nonexistant e:\winddk\7600.16385.1\bat, removing
BUILD: Compile and Link for x86
BUILD: Loading e:\winddk\7600.16385.1\build.dat...
BUILD: Computing Include file dependencies:
BUILD: Start time: Wed Jun 10 14:13:59 2015
BUILD: Examining e:\winddk\first directory for files to compile.
      e:\winddk\first Invalidating OACR warning log for 'root:x86chk'
BUILD: Saving e:\winddk\7600.16385.1\build.dat...
BUILD: Compiling and Linking e:\winddk\first directory
Configuring OACR for 'root:x86chk' - <OACR on
_NT_TARGET_VERSION SET TO WINXP
Compiling - first.c
Linking Executable - objchk_x86_x86\i386\first.sys
BUILD: Finish time: Wed Jun 10 14:14:03 2015
BUILD: Done

3 files compiled - 1 Warning - 29 LPS
1 executable built
```



The screenshot shows a Windows XP desktop with two windows. The top window is a command prompt titled "C:\WINDOWS\system32\cmd.exe" showing the execution of a service named "test\_first". The bottom window is "DebugView on \\2012-20130926Z (local)" showing the debug output of the service.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>net start test_first

test_first 服务已经启动成功。

C:\Documents and Settings\Administrator>net stop test_first

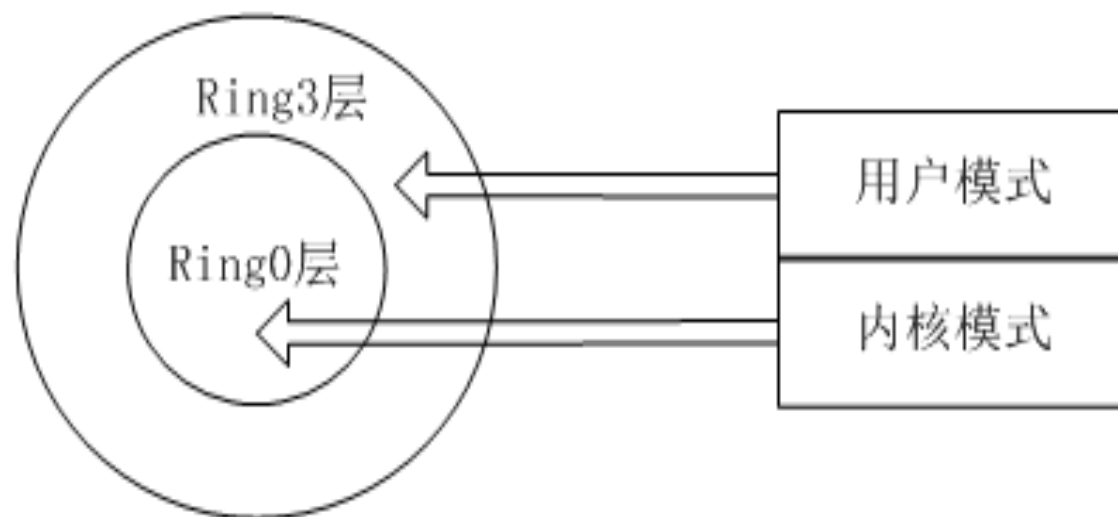
test_first 服务已成功停止。
```

The DebugView window shows the following debug output:

#	Time	Debug Print
1	0.00000000	firstFilter: Hello everyone, welcome to Technical Sharon!
2	329.49856567	firstFilter: our driver is unloading...

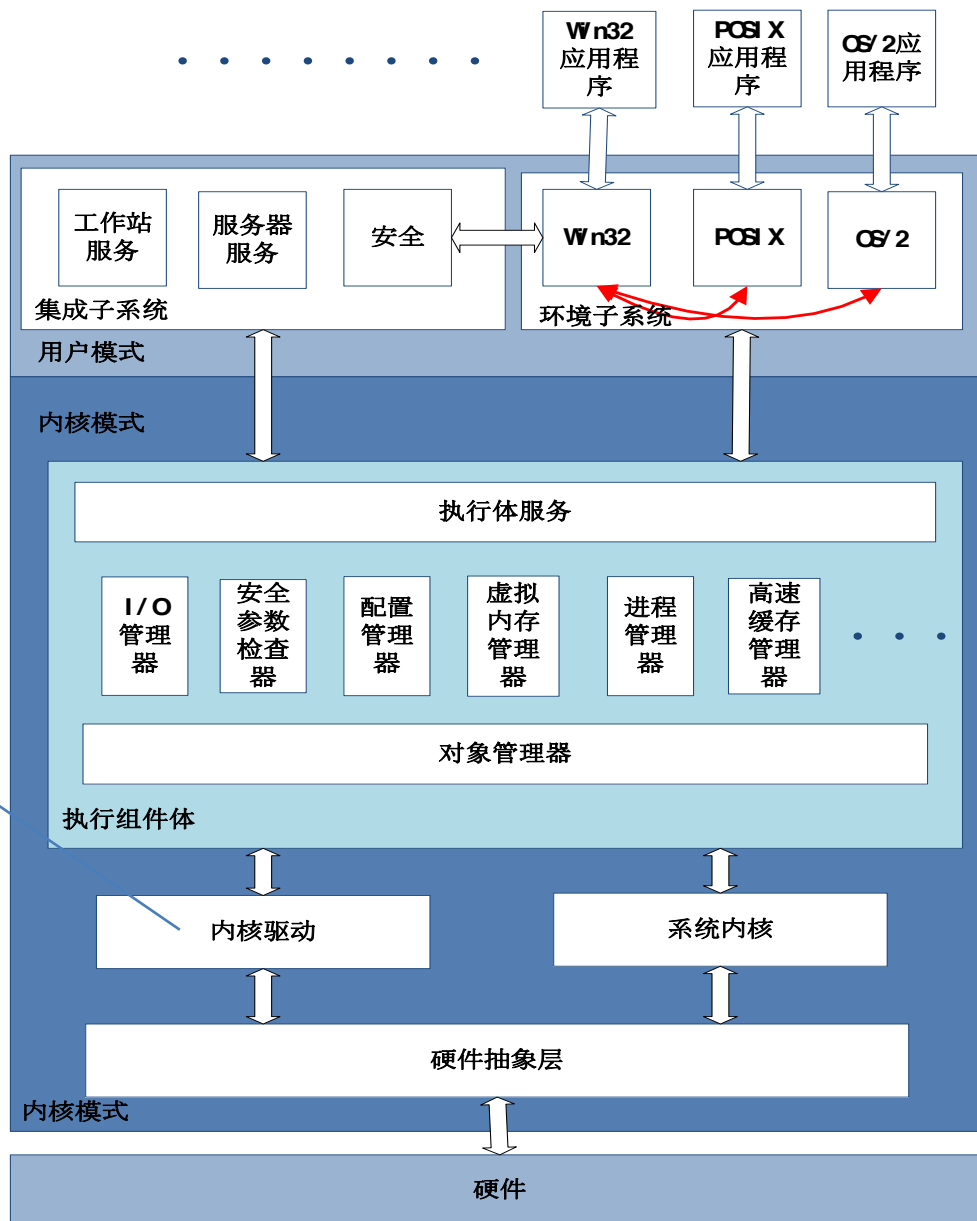
# Windows层次结构

# Windows层次结构





# Windows层次结构

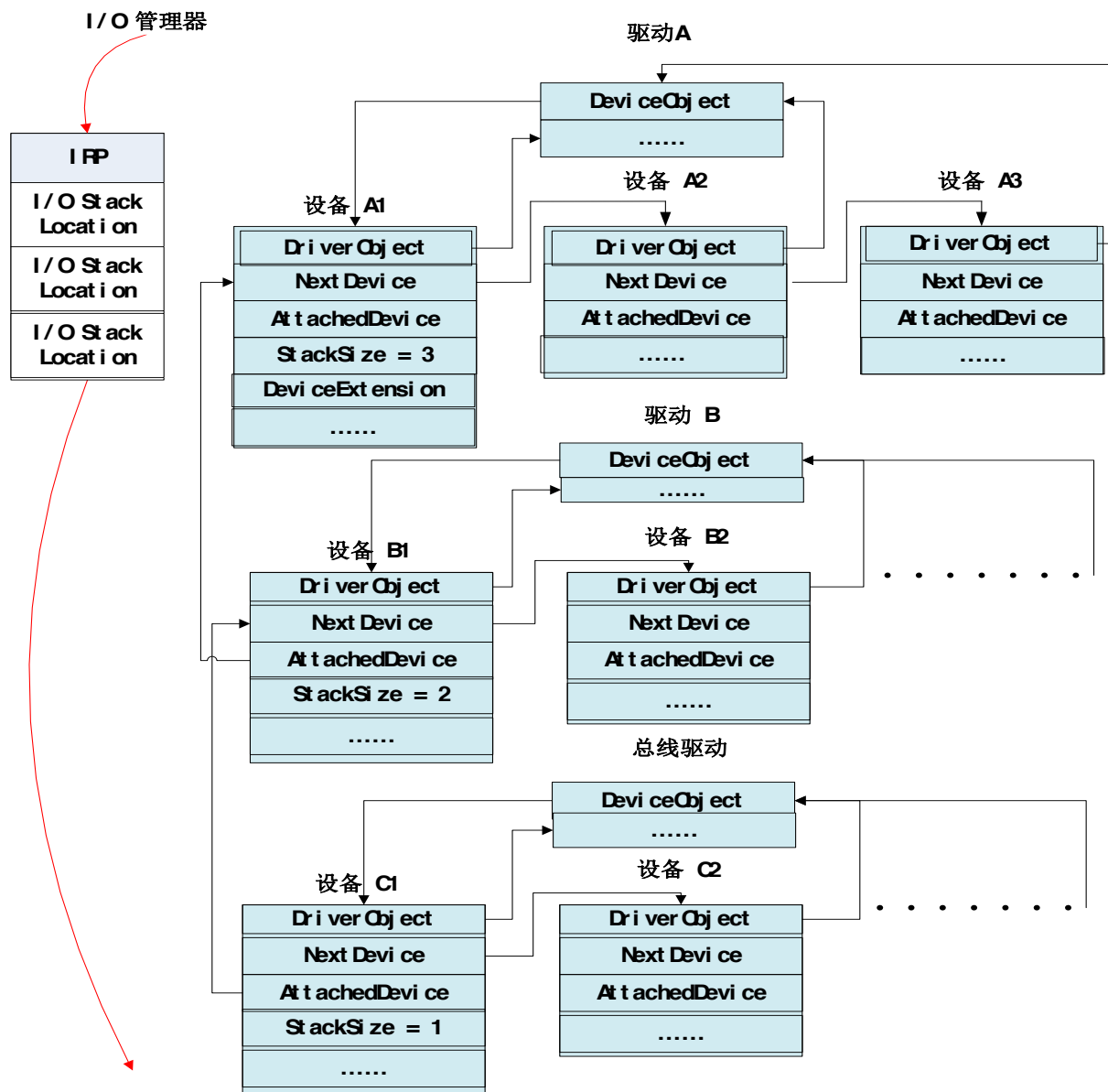


可由我们编程模块

底层驱动设备栈框架、核心概念

解释

# 驱动设备框架图



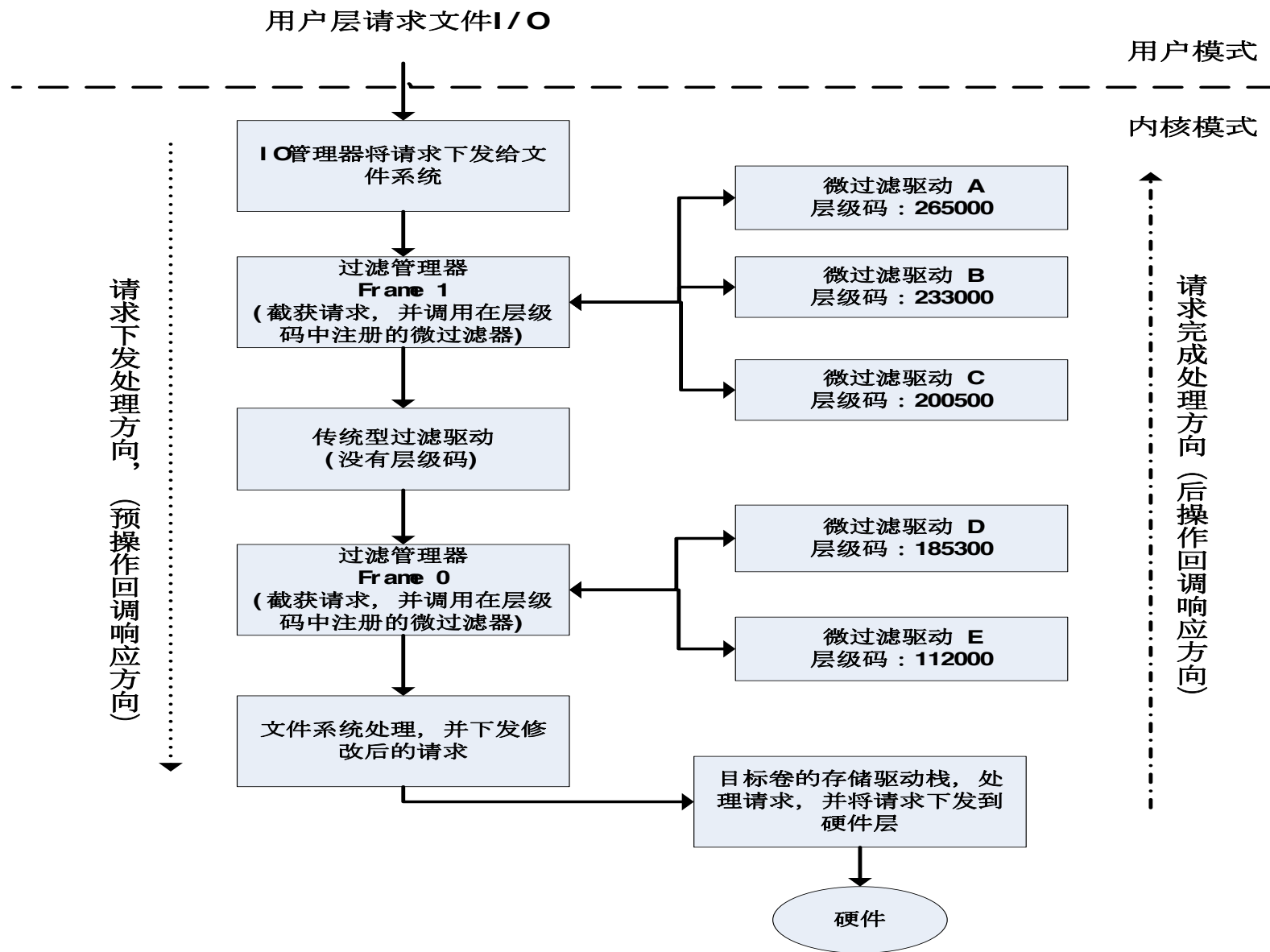
**驱动对象：**每个成功加载的内核驱动镜像都有且只有一个驱动对象表示，作为I/O管理器加载的一个实例。

**设备对象：**由驱动创建，设备对象间通过NextDevice域连接。设备链表由I/O管理器维护。此外，设备对象是唯一能接收IRP的实体。

**IRP：**该数据结构被用来描述I/O请求。使用这样一个结构对通信过程中涉及的大量参数进行封装，使得通信过程变得很简单。

**IRP Stack Location：**I/O管理器为每个IRP创建一个I/O Stack Location 数组 (Stack Locations)，存放将要处理该IRP请求的所有驱动对应的IO\_STACK\_LOCATION结构，该结构中封装了与该驱动相关的参数。

# Minifilter框架图



两个框架

——传统型、Minfiniter

串口过滤驱动，传统型

# 串口过滤驱动程序

- 过滤的概念。过滤是在不影响上层和下层接口的情况下，在Windows系统内核中加入新的层，从而不需要修改上层的软件或者下层的真是驱动程序，就加入了新的功能。

- ① 设备绑定的内核API。进行过滤的最主要的方法是对一个设备对象（Device Object）进行绑定。通过编程可以生成一个虚拟设备对象，并“绑定”（Attach）在一个真实的设备上。一旦绑定，则本来操作系统发送给真实设备的请求，就会首先发送到这个虚拟设备。

**IoAttachDevice、IoAttachDeviceToDeviceStack、IoAttachDeviceToDeviceStackSafe**

- ② 生成过滤设备并绑定。在绑定一个设备之前，先要知道如何生成一个用于过滤的过滤设备。函数IoCreateDevice被用于生成设备。

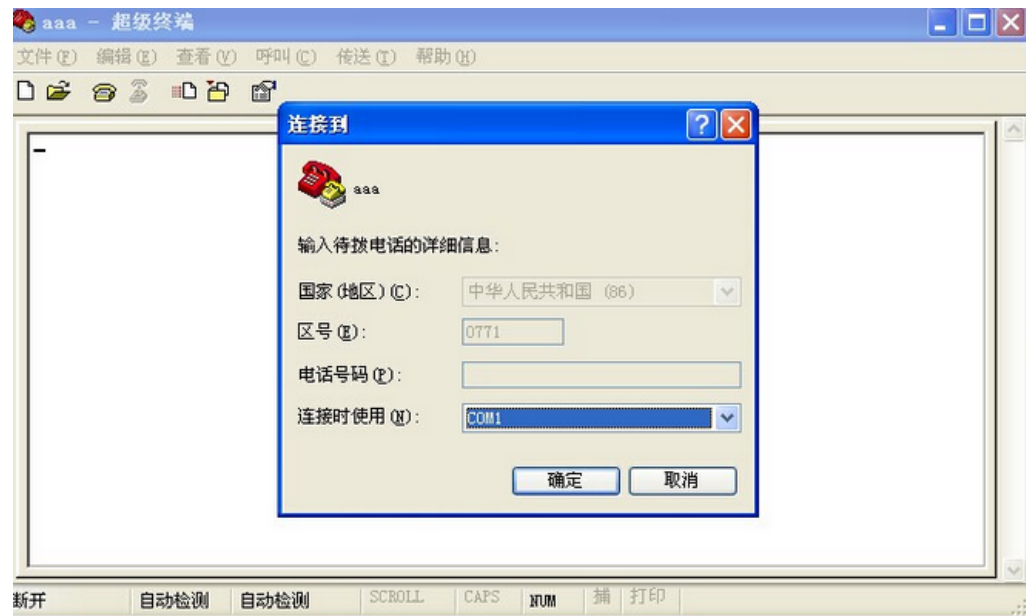
- ③ 从名字获得设备对象。在知道一个设备名字的情况下，IoGetDeviceObjectPointer可以获得这个设备对象的指针。此函数实现用于打开一个端口设备。

- ④ 绑定所有串口。

- ③ 请求处理。（1）请求被允许通过。（2）请求直接被否决。（3）过滤完成这个请求。（分发函数，主功能号，次功能号）

# 串口过滤驱动程序

- 使用串口。打开”开始“菜单——》”所有程序“——》”附件“——》”通讯“——》”超级终端“，然后任意建立一个连接。如下图所示



注意选择COM1，然后在输入框中键入字符串。其中对话框默认是不回显输入的字符的，这个可以点击”文件“——》”属性“——》”设置“——》ASCII码设置，选择回显即可。

DbgView输出：

```
212    26479.92773438 [1136] <CSync:BeginUnload
213    26479.92773438 [1136] <CSync:BeginUnload
214    26485.58007813 comcap: Send Data: 66
215    26490.58007813 comcap: Send Data: 64
216    26495.58203125 comcap: Send Data: 73
217    26502.66210938 [1136] <CSync:BeginUnload
218    26502.66210938 [1136] <CSync:BeginUnload
```



禁用Notepad.exe, Minifilter

# Minifilter——DriverEntry

① **DriverEntry**。与sfilter形成鲜明对比，这个例子的DriverEntry显得非常的简单。关键的函数有两个：**FltRegisterFilter**注册一个微过滤器；另一个用函数**FltStartFiltering**来开始过滤。

```
NTSTATUS DriverEntry (
    __in PDRIVER_OBJECT DriverObject,
    __in PUNICODE_STRING RegistryPath
)
{
    NTSTATUS status;
    UNREFERENCED_PARAMETER( RegistryPath );
    status = FltRegisterFilter( DriverObject,&FilterRegistration,&gFilterHandle );
    ASSERT( NT_SUCCESS( status ) );
    if (NT_SUCCESS( status )) {
        status = FltStartFiltering( gFilterHandle );
        if (!NT_SUCCESS( status )) { FltUnregisterFilter( gFilterHandle );}
    }
    return status;
}
```

# Minifilter——数据结构

① `typedef struct _FLT_REGISTRATION{...}`。注册微过滤器时，我们填写了一个名为微过滤器注册结构。

② 其中，最重要的是**Callbacks**。这是一个回调函数数组，在其中可以处理所有的请求。但是处理方式和以前做的请求过滤时有所不同，以前处理的是**IRP**，其实有两种处理：一种是在请求完成之前就进行处理；另一种是用事件等待请求完成之后，或者在完成函数中进行处理。前一种适合要拦截请求本身的情况，后一种适合要拦截请求之后返回的结果的情况。在**minifilter**中，这两种过滤杯截然地分在两个会回调函数中，一个称为预操作回调（**Pre-Operation Function**），另一个称为后操作回调（**Post-Operation Function**）。下面是一个例子。

```
CONST FLT_OPERATION_REGISTRATION Callbacks[] = {
    {
        IRP_MJ_CREATE,           // 主功能号
        0,
        NPPreCreate, // 生成预操作回调函数
        NPPostCreate // 生成后操作回调函数
    }
}
```

# Minifilter——回调例程

① 卸载回调例程。在卸载回调函数中，应该注销我们曾经注册过的微过滤器，这可以通过调用内核API函数FltUnregisterFilter来实现。除此之外，在这个回调函数中，读者可以完成以前在传统型的文件过滤驱动中驱动卸载函数所完成的所有工作。但是本例非常简单，仅仅是调用FltUnregisterFilter而已。这个函数也只有一个参数，就是微过滤器句柄。

② 预操作回调例程。

```
FLT_PREOP_CALLBACK_STATUS NPPreCreate(  
    __inout PFLT_CALLBACK_DATA Data,  
    __in PCFLT_RELATED_OBJECTS FltObjects,  
    __deref_out_opt PVOID *CompletionContext  
)
```

③ 预操作回调例程。

```
FLT_POSTOP_CALLBACK_STATUS  
NPPostCreate(  
    __inout PFLT_CALLBACK_DATA Data,  
    __in PCFLT_RELATED_OBJECTS FltObjects,  
    __in_opt PVOID CompletionContext,  
    __in FLT_POST_OPERATION_FLAGS Flags  
)
```

# Minifilter——与用户层通信

考虑到内核态和用户态之间的互动，以前的做法是使用用户态的API函数DeviceIoControl结合在内核模块中的处理控制请求来实现双方数据的传递。但是在minifilter中则不同，minifilter有内建支持的API提供给开发者使用，这里就先针对这些API来做介绍。

这个方法有个称呼叫做通信端口（Communication Port），顾名思义，就是先定义一个通道名称，通过双边已经定义好的通信端口来做数据上的沟通；使用上很像socket或管道(pipe)之类的通信程序设计。

FltBuildDefaultSecurityDescriptor 以 FLT\_RORT\_ALL\_ACCESS 权限来产生一个安全性的叙述子，MINISPY\_PORT\_NAME 是刚刚所讲的通信端口定义的名称，通过InitializeObjectAttributes 来初始化对象属性（OBJECT\_ATTRIBUTES），接下来便是注册这个通信端口以及所需要使用到的函数。

```
FltBuildDefaultSecurityDescriptor
```

```
InitializeObjectAttributes
```

```
status = FltCreateCommunicationPort(gFilterHandle,
```

```
    &gServerPort,
```

```
    &oa,
```

```
    NULL,
```

```
    NPMiniConnect,
```

```
    NPMiniDisconnect,
```

```
    NPMiniMessage,
```

```
    1);
```

# Minifilter——安装与加载

**StartType** 被称为启动类型。为3即**DEMAND\_START**，表示当有需求加载时才启动此驱动的功能；**StartType**为0即**BOOT\_START**，表示计算机开机启动时就自动加载此驱动的功能。根据上面的INF文件可以看到，此驱动文件是依附在**FltMgr**服务底下的。

**Altitude** 是微过滤器的层级码。层级码决定过滤层次的上下。根据微软的文档，微过滤器的层级码范围是多个以数字标示的范围区段（例如：2000~429999），正式的商业软件的层级码是需要向微软申请注册的。例如，许多防毒软件都要向微软申请一个特定的层级码，不过我们在测试时可以先给出这个范围内的一个值，这里是先定义 370030，也就是本章示例的微过滤器在过滤层中的位置。

编写这个INF文件后，测试时只要跟**Minifilter.sys**放在同一个目录下，通过鼠标右键选择INF菜单“安装”，便可以成功将**minifilter**安装到系统目录下，安装完毕后也可以去 %windir%\system32\drivers 下查看是否有这个文件的存在。另外，也可以使用**OSR driver loader** 来检查已经安装的**minifilter**在系统上的**Load Group** 顺序。

修改上面的[String]节就可以成为不同的驱动的INF文件。比如本书的例子，服务名为**NPminifilter**，驱动名为**NPminifilter.sys**，那么这一节改为：

[Strings]

**Msft** = "TEST"

**ServiceDescription** = "NPminifilter Mini-Filter Driver"

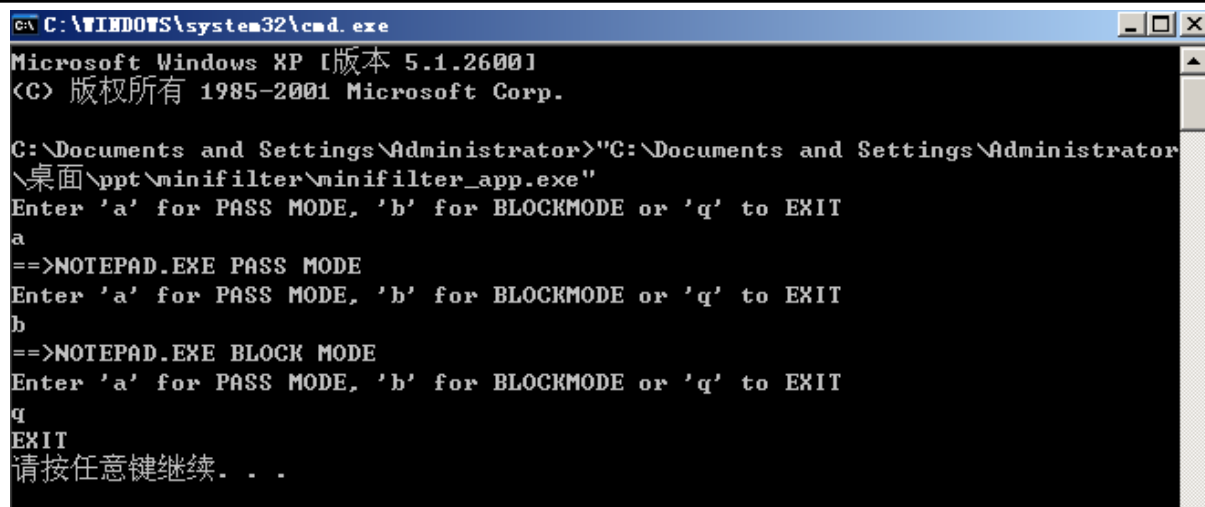
**ServiceName** = "NPminifilter"

**DriverName** = "NPminifilter"

**DiskId1** = "NPminifilter Device Installation Disk"

## Minifilter——安装与加载

- **NPminifilter.sys** : 内核模块
- **minifilter\_app.exe** : 应用层控制程序
- **minifilter\_dll.dll** : 内核层-用户层通信DLL
- **NPminifilter.inf** : 右键单击该文件，选择“安装”，即可完成内核模块加载

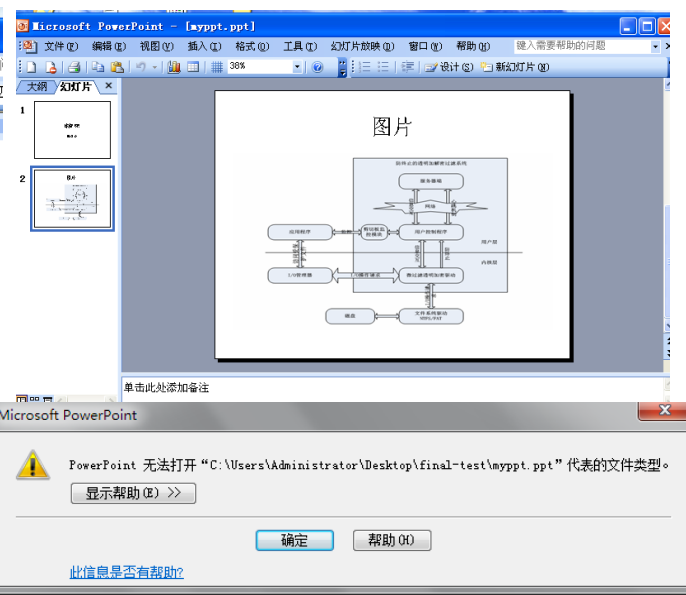
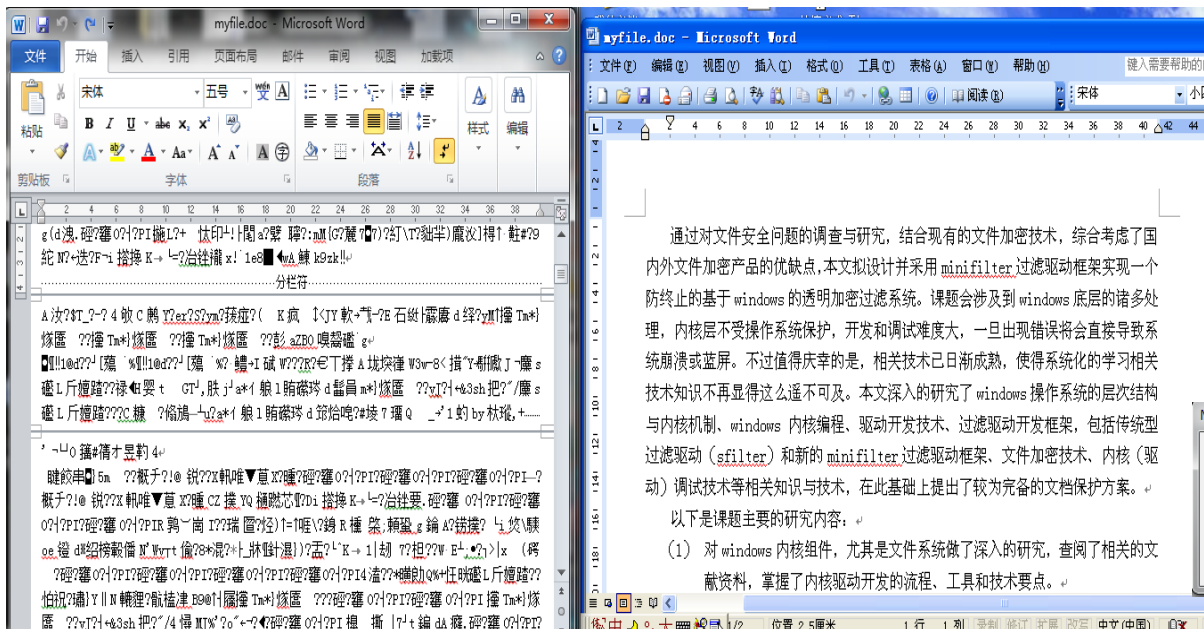


透明加密过滤驱动，简单异或  
加密



# Minifilter——透明加密过滤驱动

- IRP\_MJ\_READ : 后操作回调中解密, 给用户层返回明文
- IRP\_MJ\_WRITE : 预操作回调中加密, 写入磁盘

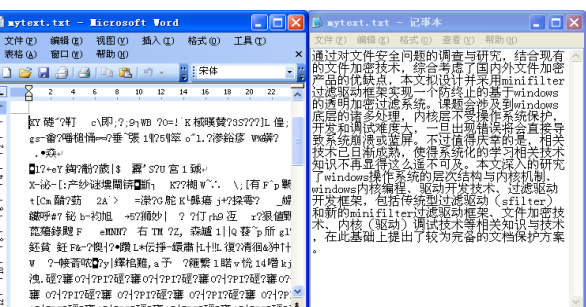


## 主机-虚拟机word文档

A screenshot of a Microsoft Excel window titled 'mydata.xls - Microsoft Excel'. The spreadsheet contains a table with columns labeled '编号' (Number), 'x(m)', 'y(m)', '海拔(m)', '功能区', and '功能区'. The data is garbled.

编号	x(m)	y(m)	海拔(m)	功能区	功能区
1	74	781	5	1	生活区
2	1373	731	11	4	2 工业区
3	1321	1791	28	4	3 山区
4	0	1787	4	2	4 交通区
5	1049	2127	12	4	5 公园绿地区
6	1647	2728	6	2	
7	2883	3617	15	4	
8	2388	3692	7	2	
9	2708	2295	22	4	
10	2933	1767	7	4	
11	4233	895	6	5	
12	4043	1895	14	1	
13	2427	3971	2	1	
14	3526	4357	7	4	
15	5062	4339	5	4	
16	4777	4897	8	1	
17	5868	4904	16	4	

## 主机-虚拟机excel文档



## 主机-虚拟机txt文档

# 入门心得

1. 双机调试环境搭建，windbg就很强大。多跟踪底层处理过程，不要光靠打印排除错误。
2. 《寒江独钓——Windows内核安全编程》，入门还是不错的，涉及到的例子要自己调试，可以的话稍作改动，加入自己的功能。《Windows核心编程》——Jeffrey Richter，很好的一本书。
3. 微软在WDK中的例子，还是很有必要研读的，这是一手资料。
4. 微软的开发文档，这是必须要看的。
5. 理论知识也不能少，有关Windows的框架以及原理还是有必要掌握。
6. 开发环境不必太纠结，我用的是WDK+VS2010。VS2010只是辅助编码，支持用WDK自带编译器进行编译，这样可以发现内核驱动特有的错误或警报。
7. 遇到问题，可以先自己排查，再与同学或老师讨论，其中要学会充分利用网络资源。

资料分享

Everything Windows Driver Development, OSR Online :

<http://osronline.com/>

看雪 :

<http://bbs.pediy.com/>

msdn :

<http://blogs.msdn.com/b/alexcarp/>

微软官网:

[https://msdn.microsoft.com/en-us/library/windows/hardware/ff540402\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff540402(v=vs.85).aspx)

CODE PROJECT:

<http://www.codeproject.com/>

驱网:

<http://bbs3.driverdevelop.com/thread.php?fid-39.html>

CSDN、博客园等也有一些文章。会百度，会google也能找到的。

# Thank you

2015.06.11

The road to success was trial and error development, recompilation, and lots of crashes.  
——共勉