使用 HOOK 来监控文件操作的方法有很多，可以在 SSDT 上 HOOK 一堆和 FILE 有关的函数，也可以对 FSD 进行 IRP HOOK，不过这些方法既不方便，也不安全。微软推荐的文件操作过滤方法是使用过滤驱动，在 VISTA 之后，推荐使用 MINIFILTER（字面翻译是迷你过滤器）。MINIFILTER 基于标准的文件过滤驱动，但是微软把它封装得很好，使得大家不用注意太多细节，而专注于对 IRP 的过滤。不过，使用 MINIFILTER 有点麻烦，它只能在 WDM 驱动里使用。如果在普通 NT 驱动里使用，注册回调的函数会失败。而 WDM 驱动则需要使用 INF 文件进行加载，直接使用 NtLoadDriver 加载会失败（这个跟 NtLoadDriver 没什么关系，跟注册表有关系，具体原因很复杂）。下面开始一步步讲解。

不过 WDM 驱动在编写上并没有什么不同。而在驱动里使用 MINIFILTER 则需要修改 SOURCE 文件，把 DRIVERTYPE 设置为 FS。使用 MINIFILTER 其实很简单，主要步骤就三个：1.设置你要过滤的 IRP；2.使用 FltRegisterFilter 注册过滤器；3.使用 FltStartFiltering 开启过滤器。在驱动卸载历程（DriverUnload）里，使用 FltUnregisterFilter 卸载过滤器。不过如果加上通信的话，就稍显复杂了。跟 MINIFILTER 通信并不使用传统的 DeviceIoControl 机制，而是有专门的通信函数，不过这个跟监控文件无关，本文略过不讲。注册和卸载 MINIFILTER 的代码如下：

```
//注册 MINIFILTER
NTSTATUS DriverEntry
(
        __in PDRIVER_OBJECT DriverObject,
        __in PUNICODE_STRING RegistryPath
)
{
        NTSTATUS status = STATUS_UNSUCCESSFUL;
        //注册过滤器
        status = FltRegisterFilter( DriverObject,
                                                  &FilterRegistration,
                                                  &g_pFilterHandle );
        if (NT_SUCCESS( status ))
        {
                //开启过滤器
                status = FltStartFiltering( g_pFilterHandle );
                if (!NT_SUCCESS( status ))
                {
                        FltUnregisterFilter( g_pFilterHandle ); //失败则卸载过滤器
                }
        }
_Exit_DriverEntry:
        if (!NT_SUCCESS( status ) )
        {
                if( NULL != g_pFilterHandle )
                {
                        FltUnregisterFilter( g_pFilterHandle ); //失败则卸载过滤器
                        g_pFilterHandle = NULL;
                }
        }
        DbgPrint("[MiniFilter][DriverEntry]status:%x\n", status);
        return status;
}
```

```
//卸载 MINIFILTER
NTSTATUS NPUnload
(
    __in FLT_FILTER_UNLOAD_FLAGS Flags
)
{
    UNREFERENCED_PARAMETER( Flags );
    PAGED_CODE();
    FltUnregisterFilter( g_pFilterHandle ); //卸载
    DbgPrint("[MiniFilter][DriverUnload]\n");
    return STATUS_SUCCESS;
}
```

在注册函数里，用到了两个很重要的结构体常量：FilterRegistration 和 Callbacks。Callbacks 包含在 FilterRegistration 里。在 Callbacks 里设置你要关注的 IRP，以及对应的处理函数。PreXXX 是执行前的处理，PostXXX 是执行后的处理：

```
//operation registration
const FLT_OPERATION_REGISTRATION Callbacks[] =
{
    {
        IRP_MJ_CREATE,
        0,
        NPPreCreate,
        NPPostCreate
    },
    {
        IRP_MJ_SET_INFORMATION,
        0,
        NPPreSetInformation,
        NPPostSetInformation
    },
    {
        IRP_MJ_READ,
        0,
        NPPreRead,
        NPPostRead
    },
    {
        IRP_MJ_WRITE,
        0,
        NPPreWrite,
        NPPostWrite
    },
    {
        IRP_MJ_OPERATION_END
    }
};

//This defines what we want to filter with FltMgr
const FLT_REGISTRATION FilterRegistration =
{
    sizeof( FLT_REGISTRATION ),             //    Size
    FLT_REGISTRATION_VERSION,               //    Version
    0,                                      //    Flags
    NULL,                                   //    Context
    Callbacks,                              //    Operation callbacks
    NPUnload,                               //    MiniFilterUnload
    NULL,                                   //    InstanceSetup
    NULL,                                   //    InstanceQueryTeardown
    NULL,                                   //    InstanceTeardownStart
    NULL,                                   //    InstanceTeardownComplete
    NULL,                                   //    GenerateFileName
    NULL,                                   //    GenerateDestinationFileName
    NULL                                    //    NormalizeNameComponent
};
```

要进行监控的话，通常在 PreXXX 里处理；而要进行监视的话，则通常在 PostXXX 里处理（当然监视在 PreXXX 里处理也行）。以下例子演示了监视文件读写、删除、改名和改属性的操作，并且禁止对 README.TXT 做任何修改。其基本逻辑也很简单，在传入的参数里获取文件名，并打印出来，如果发现是被保护的文件，就返回拒绝操作即可。

```
//过滤 IRP_MJ_CREATE
FLT_PREOP_CALLBACK_STATUS NPPreCreate
(
       __inout PFLT_CALLBACK_DATA Data,
       __in PCFLT_RELATED_OBJECTS FltObjects,
       __deref_out_opt PVOID *CompletionContext
)
{
        UNREFERENCED_PARAMETER( FltObjects );
        UNREFERENCED_PARAMETER( CompletionContext );
        PAGED_CODE();
        {
            UCHAR MajorFunction = 0;
```

```
		ULONG Options = 0;
		PFLT_FILE_NAME_INFORMATION nameInfo;
		MajorFunction = Data->Iopb->MajorFunction;
		Options = Data->Iopb->Parameters.Create.Options;
		//如果是 IRP_MJ_CREATE，且选项是 FILE_DELETE_ON_CLOSE，并且能成功获得文件名信息
		if( IRP_MJ_CREATE == MajorFunction && FILE_DELETE_ON_CLOSE == Options &&
			NT_SUCCESS( FltGetFileNameInformation( Data, FLT_FILE_NAME_NORMALIZED |
FLT_FILE_NAME_QUERY_DEFAULT, &nameInfo ) ) )
		{
			//如果解析文件信息成功
			if( NT_SUCCESS( FltParseFileNameInformation( nameInfo ) ) )
			{
				WCHAR pTempBuf[ 512 ] = { 0 };
				WCHAR *pNonPageBuf = NULL, *pTemp = pTempBuf;
				if( nameInfo->Name.MaximumLength > 512 )
				{
					pNonPageBuf = ExAllocatePool( NonPagedPool,
nameInfo->Name.MaximumLength );
					pTemp = pNonPageBuf;
				}
				RtlCopyMemory( pTemp, nameInfo->Name.Buffer, nameInfo->Name.MaximumLength );
				DbgPrint("[MiniFilter][IRP_MJ_CREATE]%wZ", &nameInfo->Name);
				_wcsupr( pTemp );
				if( NULL != wcsstr( pTemp, L"README.TXT" ) )   // 检查是不是要保护的文件
				{
					//DbgPrint( "\r\nIn NPPreCreate(), FilePath{%wZ} is forbided.",
&nameInfo->Name );
					if( NULL != pNonPageBuf )
						ExFreePool( pNonPageBuf );
					FltReleaseFileNameInformation( nameInfo );
					return FLT_PREOP_COMPLETE;
				}
				if( NULL != pNonPageBuf )
					ExFreePool( pNonPageBuf );
			}
			FltReleaseFileNameInformation( nameInfo );
		}
	}
	return FLT_PREOP_SUCCESS_NO_CALLBACK;
}
```

```
//过滤 IRP_MJ_SET_INFORMATION
FLT_PREOP_CALLBACK_STATUS NPPreSetInformation
(
	__inout PFLT_CALLBACK_DATA Data,
    __in PCFLT_RELATED_OBJECTS FltObjects,
    __deref_out_opt PVOID *CompletionContext
)
{
                    UNREFERENCED_PARAMETER( FltObjects );
                  UNREFERENCED_PARAMETER( CompletionContext );
                            PAGED_CODE();
                                {
                                  UCHAR MajorFunction = 0;
                                  PFLT_FILE_NAME_INFORMATION nameInfo;
                                  MajorFunction = Data->Iopb->MajorFunction;
                                  //如果操作是 IRP_MJ_SET_INFORMATION 且成功获
得文件名信息
                                  if( IRP_MJ_SET_INFORMATION == MajorFunction &&
                                      NT_SUCCESS(FltGetFileNameInformation(Data,
FLT_FILE_NAME_NORMALIZED | FLT_FILE_NAME_QUERY_DEFAULT, &nameInfo)) )
                                  {

                if( NT_SUCCESS( FltParseFileNameInformation( nameInfo ) ) )
                                      {
                                          WCHAR pTempBuf[ 512 ] = { 0 };
                                          WCHAR *pNonPageBuf = NULL, *pTemp =
pTempBuf;
                                          if(   nameInfo->Name.MaximumLength   >
512 )
                                          {
                                              pNonPageBuf                    =
ExAllocatePool( NonPagedPool, nameInfo->Name.MaximumLength );
                                              pTemp = pNonPageBuf;
                                          }
                                          RtlCopyMemory(                   pTemp,
nameInfo->Name.Buffer, nameInfo->Name.MaximumLength );

          DbgPrint("[MiniFilter][IRP_MJ_SET_INFORMATION]%wZ", &nameInfo->Name);
                                          _wcsupr( pTemp );
                                          if(     NULL     !=     wcsstr(     pTemp,
L"README.TXT" ) )   // 检查是不是要保护的文件
                                          {
                                              //DbgPrint(                  "\r\nIn
NPPreSetInformation(), FilePath{%wZ} is forbided.", &nameInfo->Name );
                                              if( NULL != pNonPageBuf )
                                                  ExFreePool( pNonPageBuf );

                    FltReleaseFileNameInformation( nameInfo );
```

```
                                                                        return
FLT_PREOP_DISALLOW_FASTIO;
                                                                    }
                                                                    if( NULL != pNonPageBuf )
                                                                        ExFreePool( pNonPageBuf );
                                                                }
                                                                FltReleaseFileNameInformation( nameInfo );
                                                            }
                                                        }
                                    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}
```

```
//过滤 IRP_MJ_READ
FLT_PREOP_CALLBACK_STATUS NPPreRead
(
    __inout PFLT_CALLBACK_DATA Data,
    __in PCFLT_RELATED_OBJECTS FltObjects,
    __deref_out_opt PVOID *CompletionContext
)
{
    UNREFERENCED_PARAMETER( FltObjects );
    UNREFERENCED_PARAMETER( CompletionContext );
    PAGED_CODE();
    {
        PFLT_FILE_NAME_INFORMATION nameInfo;
        //直接获得文件名并检查
        if( NT_SUCCESS( FltGetFileNameInformation( Data, FLT_FILE_NAME_NORMALIZED |
FLT_FILE_NAME_QUERY_DEFAULT, &nameInfo ) ) )
        {
            if( NT_SUCCESS( FltParseFileNameInformation( nameInfo ) ) )
            {
                WCHAR pTempBuf[ 512 ] = { 0 };
                WCHAR *pNonPageBuf = NULL, *pTemp = pTempBuf;
                if( nameInfo->Name.MaximumLength > 512 )
                {
                    pNonPageBuf = ExAllocatePool( NonPagedPool,
nameInfo->Name.MaximumLength );
                    pTemp = pNonPageBuf;
                }
                RtlCopyMemory( pTemp, nameInfo->Name.Buffer, nameInfo->Name.MaximumLength );
                DbgPrint("[MiniFilter][IRP_MJ_READ]%wZ", &nameInfo->Name);
                /*_wcsupr( pTemp );
                if( NULL != wcsstr( pTemp, L"README.TXT" ) )   // 检查是不是要保护的文件
                {
                    //DbgPrint( "\r\nIn NPPreWrite(), FilePath{%wZ} is forbided.",
&nameInfo->Name );
                    if( NULL != pNonPageBuf )
                        ExFreePool( pNonPageBuf );
                    FltReleaseFileNameInformation( nameInfo );
                    return FLT_PREOP_DISALLOW_FASTIO;
                }*/
                if( NULL != pNonPageBuf )
                    ExFreePool( pNonPageBuf );
            }
            FltReleaseFileNameInformation( nameInfo );
        }
    }
    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}
```

```
//过滤 IRP_MJ_WRITE
FLT_PREOP_CALLBACK_STATUS NPPreWrite
(
    __inout PFLT_CALLBACK_DATA Data,
    __in PCFLT_RELATED_OBJECTS FltObjects,
    __deref_out_opt PVOID *CompletionContext
)
{
    UNREFERENCED_PARAMETER( FltObjects );
    UNREFERENCED_PARAMETER( CompletionContext );
    PAGED_CODE();
    {
        PFLT_FILE_NAME_INFORMATION nameInfo;
        //直接获得文件名并检查
        if( NT_SUCCESS( FltGetFileNameInformation( Data, FLT_FILE_NAME_NORMALIZED | 
FLT_FILE_NAME_QUERY_DEFAULT, &nameInfo ) ) )
        {
            if( NT_SUCCESS( FltParseFileNameInformation( nameInfo ) ) )
            {
                WCHAR pTempBuf[ 512 ] = { 0 };
                WCHAR *pNonPageBuf = NULL, *pTemp = pTempBuf;
                if( nameInfo->Name.MaximumLength > 512 )
                {
                    pNonPageBuf = ExAllocatePool( NonPagedPool, 
nameInfo->Name.MaximumLength );
                    pTemp = pNonPageBuf;
                }
                RtlCopyMemory( pTemp, nameInfo->Name.Buffer, nameInfo->Name.MaximumLength );
                DbgPrint("[MiniFilter][IRP_MJ_WRITE]%wZ", &nameInfo->Name);
                _wcsupr( pTemp );
```

```
                    if( NULL != wcsstr( pTemp, L"README.TXT" ) )   // 检查是不是要保护的文件
                    {
                        //DbgPrint(      "\r\nIn      NPPreWrite(),      FilePath{%wZ}      is      forbided.",
&nameInfo->Name );
                        if( NULL != pNonPageBuf )
                            ExFreePool( pNonPageBuf );
                        FltReleaseFileNameInformation( nameInfo );
                        return FLT_PREOP_DISALLOW_FASTIO;
                    }
                    if( NULL != pNonPageBuf )
                        ExFreePool( pNonPageBuf );
                }
                FltReleaseFileNameInformation( nameInfo );
            }
        }
        return FLT_PREOP_SUCCESS_NO_CALLBACK;
}
```

有了驱动程序，就说说如何加载 WDM 驱动。加载 WDM 驱动首先需要一个 INF 文件：

```
[Version]
Signature     = "$Windows NT$"
Class          = "ActivityMonitor"                              ;This is determined by the work this filter driver
does
ClassGuid     = {D93806DA-228D-4a25-A065-5F676DEA1C05}       ;This value is determined by the Class
Provider       = %TASOFT%
DriverVer     = 04/10/2012,1.0.0.1
CatalogFile = MFTest.cat



[DestinationDirs]
DefaultDestDir                 = 12
MiniFilter.DriverFiles    = 12                    ;%windir%\system32\drivers

;;
;; Default install sections
;;

[DefaultInstall]
OptionDesc                 = %ServiceDescription%
CopyFiles                  = MiniFilter.DriverFiles

[DefaultInstall.Services]
AddService                  = %ServiceName%,,MiniFilter.Service
```

```
;;
;; Default uninstall sections
;;

[DefaultUninstall]
DelFiles      = MiniFilter.DriverFiles

[DefaultUninstall.Services]
DelService = %ServiceName%,0x200           ;Ensure service is stopped before deleting


;
; Services Section
;

[MiniFilter.Service]
DisplayName          = %ServiceName%
Description          = %ServiceDescription%
ServiceBinary        = %12%\%DriverName%.sys             ;%windir%\system32\drivers\
Dependencies         = "FltMgr"
ServiceType          = 2                                 ;SERVICE_FILE_SYSTEM_DRIVER
StartType            = 3                                 ;SERVICE_DEMAND_START
ErrorControl         = 1                                 ;SERVICE_ERROR_NORMAL
LoadOrderGroup       = "FSFilter Activity Monitor"
AddReg               = MiniFilter.AddRegistry

;
; Registry Modifications
;

[MiniFilter.AddRegistry]
HKR,,"DebugFlags",0x00010001 ,0x0
HKR,"Instances","DefaultInstance",0x00000000,%DefaultInstance%
HKR,"Instances\"%Instance1.Name%,"Altitude",0x00000000,%Instance1.Altitude%
HKR,"Instances\"%Instance1.Name%,"Flags",0x00010001,%Instance1.Flags%


;
; Copy Files
;

[MiniFilter.DriverFiles]
%DriverName%.sys

[SourceDisksFiles]
MFTest.sys = 1,,
[SourceDisksNames]
1 = %DiskId1%,,,

;;
;; String Section
;;

[Strings]
TASOFT                          = "TASOFT"
ServiceDescription          = "MFTest Mini-Filter Driver"
ServiceName                    = "MFTest"
DriverName                      = MFTest
DiskId1                          = MFTest Device Installation Disk"

;Instances specific information.
DefaultInstance              = MFTest Instance"
Instance1.Name                 = MFTest Instance"
Instance1.Altitude         = "270030"
Instance1.Flags              = 0x0                    ; Allow all attachments
```

在这个 INF 文件里，我们最关心不是那些名称，而是一个名为 Altitude 的属性。Altitude 是一个数值。如果这个数值和别人重名了，会直接引发 BSOD。关于 Altitude 的官方解释如下：http://msdn.microsoft.com/en-us/library/windows/hardware/ff549689(v=vs.85).aspx。而且微软还列出了别人已经使用了的 Altitude：http://msdn.microsoft.com/en-us/library/windows/hardware/dn265170(v=vs.85).aspx。说实话我不明白微软搞这么个东西意义何在，不过也不必太过在乎这个值，只要把这个值在 229998～320000 之间取就行了。

有了 INF，加载/卸载驱动就很方便了。只要三行批处理就行：

```
::安装驱动并开启
InfDefaultInstall INF_FILE_PATH
net start MFTest
::等待
pause
::关闭保护
net stop MFTest
```

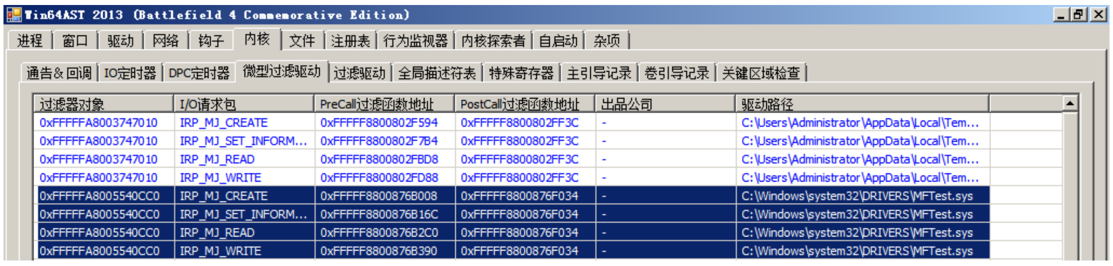
用编程实现如下：

```
#include <stdio.h>
#include <Windows.h>
#pragma comment(lib,"Advapi32.lib")
//连接字符串
char *cs(char *str1, char *str2)
{
	SIZE_T newstrlen=strlen(str1)+strlen(str2)+1;
	char *newstr=(char*)malloc(newstrlen);
	memcpy(newstr,str1,strlen(str1));
	memcpy(newstr+strlen(str1),str2,strlen(str2)+1);
	return newstr;
}
//加载 WDM 驱动
BOOLEAN LoadWdmDrv(char *inf, char *szDrvSvcName)
{
	CHAR exe[]="c:\\windows\\system32\\InfDefaultInstall.exe ";
	STARTUPINFOA si={0};
	PROCESS_INFORMATION pi={0};
	si.cb=sizeof(si);
	char *cmd=cs(exe,inf);
	if(!CreateProcessA(NULL, cmd, NULL, NULL, 0, 0, NULL, NULL, &si, &pi))
		return FALSE;
	WaitForSingleObject(pi.hProcess,INFINITE);
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);
	SC_HANDLE hSCM=OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
	SC_HANDLE hSvcHandle = OpenServiceA(hSCM, szDrvSvcName, SERVICE_ALL_ACCESS);
	if(hSvcHandle)
	{
		CloseServiceHandle(hSCM);
		CloseServiceHandle(hSvcHandle);
		return TRUE;
	}
	else
	{
		CloseServiceHandle(hSCM);
		return FALSE;
	}
}
//开启服务
BOOLEAN net_start(char *szDrvSvcName)
{
	BOOLEAN b=FALSE;
	SC_HANDLE hSCM=OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
	SC_HANDLE hSvcHandle = OpenServiceA(hSCM, szDrvSvcName, SERVICE_ALL_ACCESS);
	if(hSvcHandle)
	{
		b=StartServiceA(hSvcHandle, NULL, NULL);
		CloseServiceHandle(hSvcHandle);
	}
```

```
        CloseServiceHandle(hSCM);
        return b;
}
//停止服务
BOOLEAN net_stop(char *szDrvSvcName)
{
        BOOLEAN b=FALSE;
        SERVICE_STATUS lpSt;
        SC_HANDLE hSCM=OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
        SC_HANDLE hSvcHandle = OpenServiceA(hSCM, szDrvSvcName, SERVICE_ALL_ACCESS);
        if(hSvcHandle)
        {
                b=ControlService(hSvcHandle, SERVICE_CONTROL_STOP, &lpSt);
                CloseServiceHandle(hSvcHandle);
        }
        CloseServiceHandle(hSCM);
        return b;
}
//卸载 WDM 驱动
BOOLEAN UnloadWdmDrv(char *szDrvSvcName)
{
        BOOLEAN b=FALSE;
        SC_HANDLE hSCM=OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
        SC_HANDLE hSvcHandle = OpenServiceA(hSCM, szDrvSvcName, SERVICE_ALL_ACCESS);
        if(hSvcHandle)
        {
                b=DeleteService(hSvcHandle);
                CloseServiceHandle(hSvcHandle);
        }
        CloseServiceHandle(hSCM);
        return b;
}
//调用示例
int main()
{
        BOOLEAN b;
        //copy file
        CopyFileA("setup.inf","c:\\setup.inf",FALSE);
        CopyFileA("mftest.sys","c:\\mftest.sys",FALSE);
        //load WDM DRIVER
        b=LoadWdmDrv("c:\\setup.inf", "mftest");
        printf("LoadWdmDrv: %d\n",b);
        //start driver
        b=net_start("mftest");
```

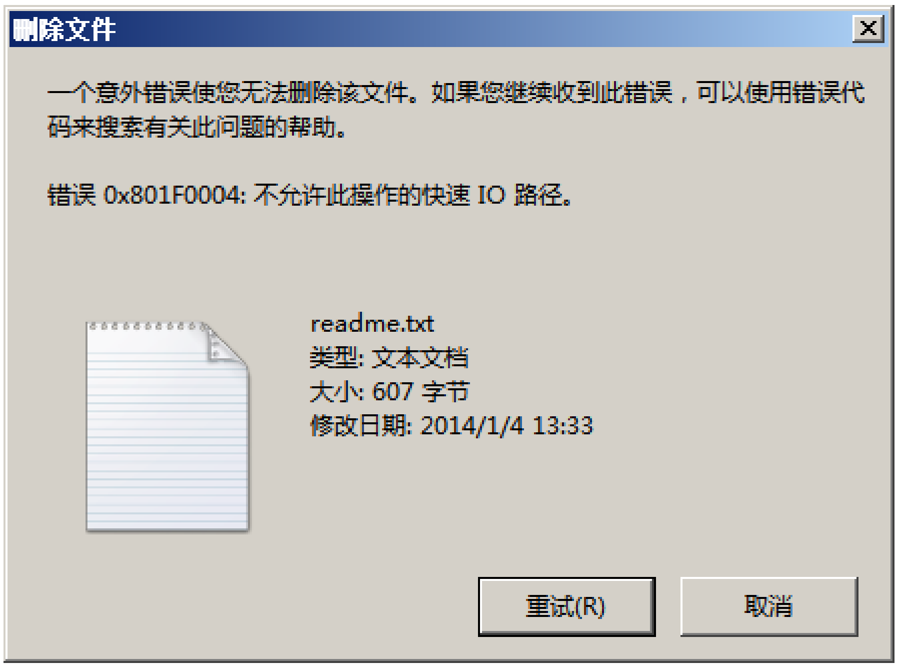
```
	printf("net_start: %d\n",b);
	printf("press ENTER to unload driver...");
	getchar();
	//stop driver
	net_stop("mftest");
	//unload driver
	UnloadWdmDrv("mftest");
	//exit
	printf("press ENTER to exit...");
	getchar();
	//delete file
	DeleteFileA("c:\\setup.inf");
	DeleteFileA("c:\\mftest.sys");
}
```

加载驱动完毕后，WIN64AST 会检测到回调的存在：


Win64AST 2013 (Battlefield 4 Commemorative Edition)
进程 | 窗口 | 驱动 | 网络 | 钩子 | 内核 | 文件 | 注册表 | 行为监视器 | 内核探索者 | 自启动 | 杂项
通告&回调 | IO定时器 | DPC定时器 | 微型过滤驱动 | 过滤驱动 | 全局描述符表 | 特殊寄存器 | 主引导记录 | 卷引导记录 | 关键区域检查

过滤器对象	I/O请求包	PreCall过滤函数地址	PostCall过滤函数地址	出品公司	驱动路径
0xFFFFFA8003747010	IRP_MJ_CREATE	0xFFFFF8800802F594	0xFFFFF8800802FF3C	-	C:\Users\Administrator\AppData\Local\Tem...
0xFFFFFA8003747010	IRP_MJ_SET_INFORM...	0xFFFFF8800802F7B4	0xFFFFF8800802FF3C	-	C:\Users\Administrator\AppData\Local\Tem...
0xFFFFFA8003747010	IRP_MJ_READ	0xFFFFF8800802FBD8	0xFFFFF8800802FF3C	-	C:\Users\Administrator\AppData\Local\Tem...
0xFFFFFA8003747010	IRP_MJ_WRITE	0xFFFFF8800802FD88	0xFFFFF8800802FF3C	-	C:\Users\Administrator\AppData\Local\Tem...
0xFFFFFA8005540CC0	IRP_MJ_CREATE	0xFFFFF8800876B008	0xFFFFF8800876F034	-	C:\Windows\system32\DRIVERS\MFTest.sys
0xFFFFFA8005540CC0	IRP_MJ_SET_INFORM...	0xFFFFF8800876B16C	0xFFFFF8800876F034	-	C:\Windows\system32\DRIVERS\MFTest.sys
0xFFFFFA8005540CC0	IRP_MJ_READ	0xFFFFF8800876B2C0	0xFFFFF8800876F034	-	C:\Windows\system32\DRIVERS\MFTest.sys
0xFFFFFA8005540CC0	IRP_MJ_WRITE	0xFFFFF8800876B390	0xFFFFF8800876F034	-	C:\Windows\system32\DRIVERS\MFTest.sys

如果尝试删除/写入/重命名文件，会报错：



一个意外错误使您无法删除该文件。如果您继续收到此错误，可以使用错误代码来搜索有关此问题的帮助。

错误 0x801F0004: 不允许此操作的快速 IO 路径。

readme.txt
类型: 文本文档
大小: 607 字节
修改日期: 2014/1/4 13:33

课后作业：完善文件监视器，做到类似于 FileMon 的效果。