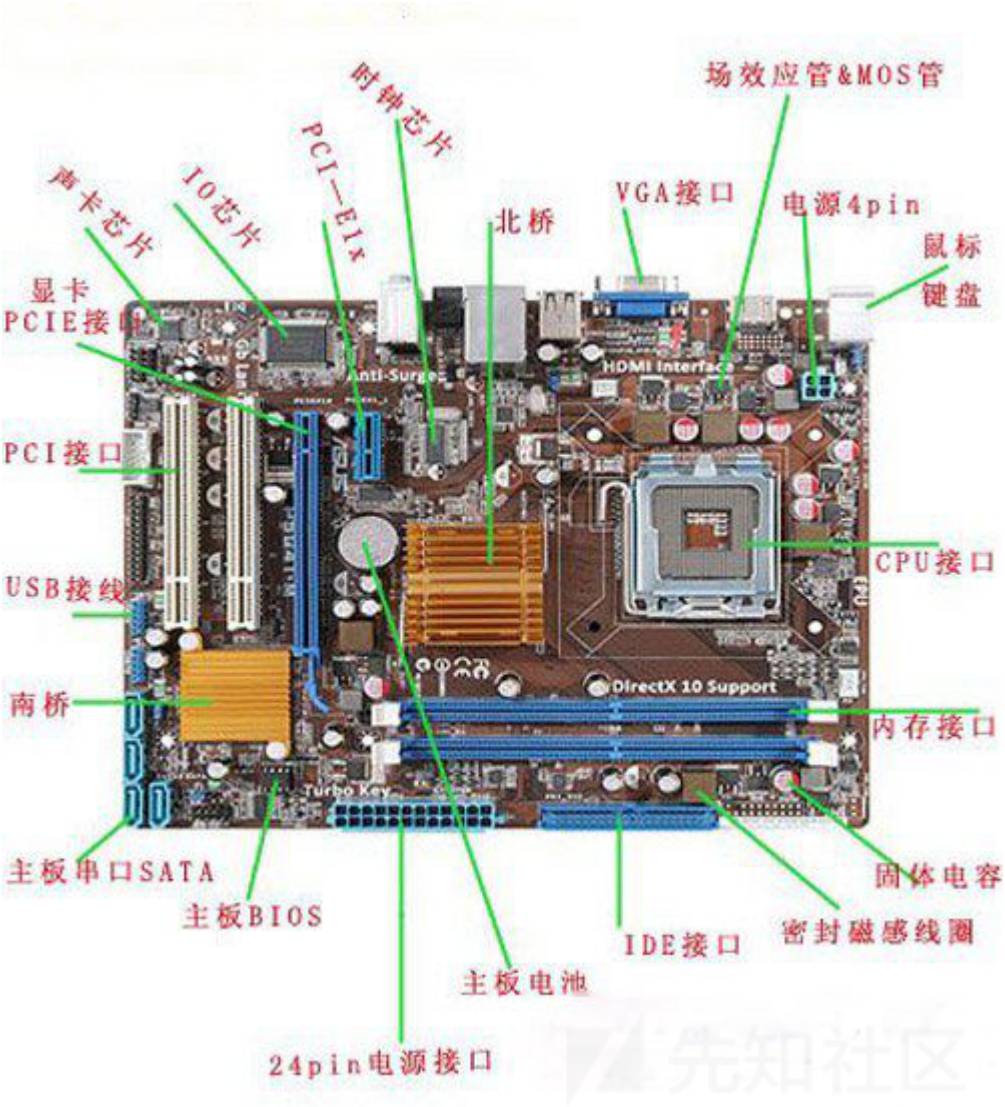
Windows驱动编程之串口过滤杂谈

一半人生 / 2019-10-10 09:08:02 / 浏览数 7588 安全技术 二进制安全 顶(1) 踩(0)



北桥：芯片处理高速信号，如cpu、ram、AGP等

南桥：芯片负责I/O总线直接通信，PCI、SATA、USB、LAN、音频、键盘控制等

串口概念：

串口也就是通常说的COM接口，采用串行通信方式扩展接口，数据bit传输，通信线路简单，用一个传输线就能双向的传输，成本低且适用于远距离通信，但是速度就比较慢

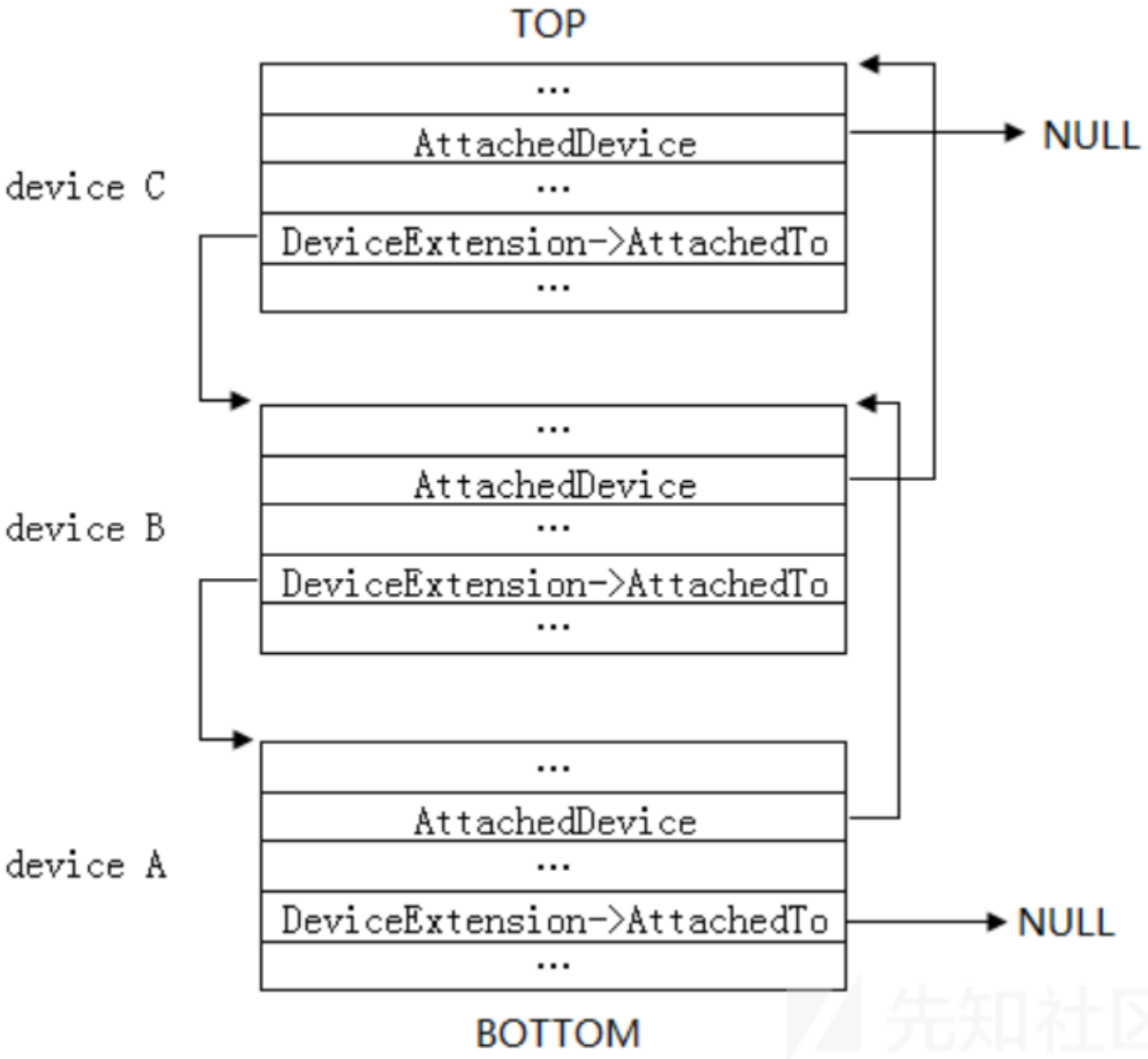
USB不是串口，这是一个最基本的概念，如上图中USB接线与串口SATA是两个东西，然而很多人却容易搞混，当然我们编写的.sys驱动并非对硬件直接操作，设备驱动程序中

串口过滤：

过滤：简单说不做任何改变，而在过程中添加一层过滤设备。早些年农夫山泉广告，每一滴水源于深山，层层工艺过滤。深山中的水就是源头，通过工具对水进行净化，把水

设备驱动过滤也是一样，不改变原封装的接口，我们只需要在中间加一层过滤设备，来对数据进行操作即可，不过对于设备栈来说，你添加的过滤设备是在最顶层。一个设备

ReactOS 系统中是用pnp管理器来对设备拔插通知做处理，Pnp会对每类的设备都创建一个根root device，向上扩展，形成一个设备栈。



功能梳理：

1. 如何创建一个过滤设备，或说创建一个设备？IoCreateDevice()

函数原型：

```
NTSTATUS
IoCreateDevice(
   _In_  PDRIVER_OBJECT DriverObject,
   _In_  ULONG DeviceExtensionSize,
   _In_opt_ PUNICODE_STRING DeviceName,
   _In_  DEVICE_TYPE DeviceType,
   _In_  ULONG DeviceCharacteristics,
   _In_  BOOLEAN Exclusive,
   _Outptr_result_nullonfailure_
   _At_(*DeviceObject,
       __drv_allocatesMem(Mem)
       _When_((((_In_function_class_(DRIVER_INITIALIZE))
              ||(_In_function_class_(DRIVER_DISPATCH)))),
            __drv_aliasesMem))
   PDEVICE_OBJECT *DeviceObject
   );

The IoCreateDevice routine creates a device object for use by a driver.
```

我们挑几个参数说一下：

DriverObject指向调用者的驱动程序对象的指针，简单点传入你当前驱动的指针就好。

DeviceExtensionSize扩展设备大小，没有扩展设备传入0就好了

DeviceName设备名，过滤设备一般是没有名称的，这个地方可以是NULL

DeviceType这个要与绑定的设备类型一样的

DeviceObject返回新创建的设备对象指针

示例：

```
PDEVICE_OBJECT fltobj = NULL;
status = IoCreateDevice(pDriver,
                        0,
                        NULL,
                        oldobj->DeviceType,
                        0,
                        FALSE,
                        &fltobj);
if(!!NT_SUCCESS(STATUS))
   return status;
oldobj->DeviceType■■■IoGetDeviceObjectPointer■■■■,■■■■■■■■■■■■■
```

2. 创建了如何绑定，也就是关联过滤的设备呢？IoAttachDevice()或IoAttachDeviceToDeviceStack()

两种情况：

一、有设备名称绑定IoAttachDevice()

函数原型：

```
NTSTATUS
IoAttachDevice(
   _In_ _When_(return==0, __drv_aliasesMem)
   PDEVICE_OBJECT SourceDevice,
   _In_  PUNICODE_STRING TargetDevice,
   _Out_ PDEVICE_OBJECT *AttachedDevice
   );
```

参数SourceDevice则是我们创建的过滤设备指针，TargetDevice则是绑定的设备名称指针，AttachedDevice返回绑定成功的设备栈指针。

二、没有设备名称绑定IoAttachDeviceToDeviceStack()，官方建议使用IoAttachDeviceToDeviceStackSafe()因为兼容性更高，更安全。

函数原型：

```
PDEVICE_OBJECT IoAttachDeviceToDeviceStack(
 PDEVICE_OBJECT SourceDevice,
 PDEVICE_OBJECT TargetDevice
);
```

SourceDevice参数同样是我们创建过滤设备对象指针，TargetDevice是绑定的设备对象指针。

`SourceDevice`

Pointer to the caller-created device object.

`TargetDevice`

Pointer to another driver's device object, such as a pointer returned by a preceding call to IoGetDeviceObjectPointer.

3. 串口是在主板上焊好的，如何获取串口的驱动呢？否则也没法绑定，IoGetDeviceObjectPointer()

函数原型：

```
NTSTATUS IoGetDeviceObjectPointer(
 PUNICODE_STRING ObjectName,
 ACCESS_MASK     DesiredAccess,
 PFILE_OBJECT    *FileObject,
 PDEVICE_OBJECT  *DeviceObject
);
```

ObjectName设备名称，DesiredAccess掩码访问权限，FileObject文件对象指针，DeviceObject逻辑、虚拟或物理设备的设备对象的指针，该函数会让内核的引用计数+2

示例：

```
RtlStringCchPrintfW(&name_str, sizeof name_str, L"\\Device\\Serial%d", id);
IoGetDeviceObjectPointer(&name_str, FILE_ALL_ACCESS, &fileobj, &devobj);
```

简单说当设备调用了IoGetDeviceObjectPointer，对象管理就会产生对应的文件对象。引用计数增加，设备就会被占用，当文件被解引用之后，IoGetDeviceObjectPointe

4.停止、卸载过滤？IoDetachDevice（），IoDeleteDevice（）

函数原型：

```
void IoDetachDevice(
 PDEVICE_OBJECT TargetDevice
);

void IoDeleteDevice(
 PDEVICE_OBJECT DeviceObject
);
```

过滤实现：

封装打开串口设备对象:

当然你可以为了绑定全部串口，只需要将下面的Serial0替换成Serial%d，做个循环即可:

```
NTSTATUS prOpenSataobj(PDEVICE_OBJECT *devobj)
{
   NTSTATUS status;
   ULONG i = 0;
   UNICODE_STRING name;
   PFILE_OBJECT fileobj = NULL;
   RtlInitUnicodeString(&name, L"\\Device\\Serial0");
   status = IoGetDeviceObjectPointer(&name, FILE_ALL_ACCESS, &fileobj, devobj);
   if (NT_SUCCESS(status))
       ObDereferenceObject(fileobj);
   return status;
}
```

封装过滤设备创建、绑定：

```
PDEVICE_OBJECT oldobj = NULL;

// ■■■■■■■■
prOpenSataobj(&oldobj);

//
PDEVICE_OBJECT prAttachDevobj(PDRIVER_OBJECT pDriver, PDEVICE_OBJECT *oldobj)
{
   NTSTATUS status;
   PDEVICE_OBJECT fltobj;
   PDEVICE_OBJECT nextobj;

   // 1. ■■■■■■
   status = IoCreateDevice(pDriver, 0, NULL, oldobj->DeviceType, 0, FALSE, fltobj);
   if (!NT_SUCCESS(status))
       return status;

   // ■■■■■■
       // ■■■■■■
   if (oldobj->Flags & DO_BUFFERED_IO)
       fltobj->Flags|= DO_BUFFERED_IO;
   if (oldobj->Flags & DO_DIRECT_IO)
```

```
        fltobj->Flags |= DO_DIRECT_IO;
    if (oldobj->Characteristics &FILE_DEVICE_SECURE_OPEN)
        fltobj->Characteristics |= FILE_DEVICE_SECURE_OPEN;

    // 2. ■■■■■■
    PDEVICE_OBJECT nrtobj = NULL;
    status = IoAttachDeviceToDeviceStackSafe(fltobj, oldobj, &nrtobj);
    if (!NT_SUCCESS(status))
    {
        IoDeleteDevice(fltobj);
        fltobj = NULL;
        status = STATUS_UNSUCCESSFUL;
        return status;

    }

    // ■■■■■■
    fltobj->Flags = fltobj->Flags & ~DO_DEVICE_INITIALIZING;

    // ■■■■■■■■
    return nrtobj;
}
```

封装获取串口数据与过滤下发：

在这之前讲个概念，关于IRP，利用派遣函数了接收了IO请求之后被调用来处理IO请求的函数.

如我们 CreateFileW打开符号链接，三环与驱动建立I/0请求是通过符号链接，而不是驱动设备名称。组 MajorFunction [IRP_MJ_CREATE] 项就就会被触发，派遣函数原型如下：

```
DRIVER_DISPATCH DriverDispatch;

NTSTATUS DriverDispatch(
 _DEVICE_OBJECT *DeviceObject,
 _IRP *Irp
)
```

我们发现有两个参数设备对象与IRP，那么用户层传递的数据其实系统其实已经将这些参数保存在了 IRP 和 IO_STACK_LOCALTION 的结构中。IRP结构保存了用户层传递进来的参数，用来保存处理不同I/O请求类型的数据，所以是个复杂的结构体，MSDN如下所示：

IRP structure

2018/04/30 • 6 分钟阅读时长

The **IRP** structure is a partially opaque structure that represents an *I/O request packet*. Drivers can use the following members of the IRP structure.

Syntax

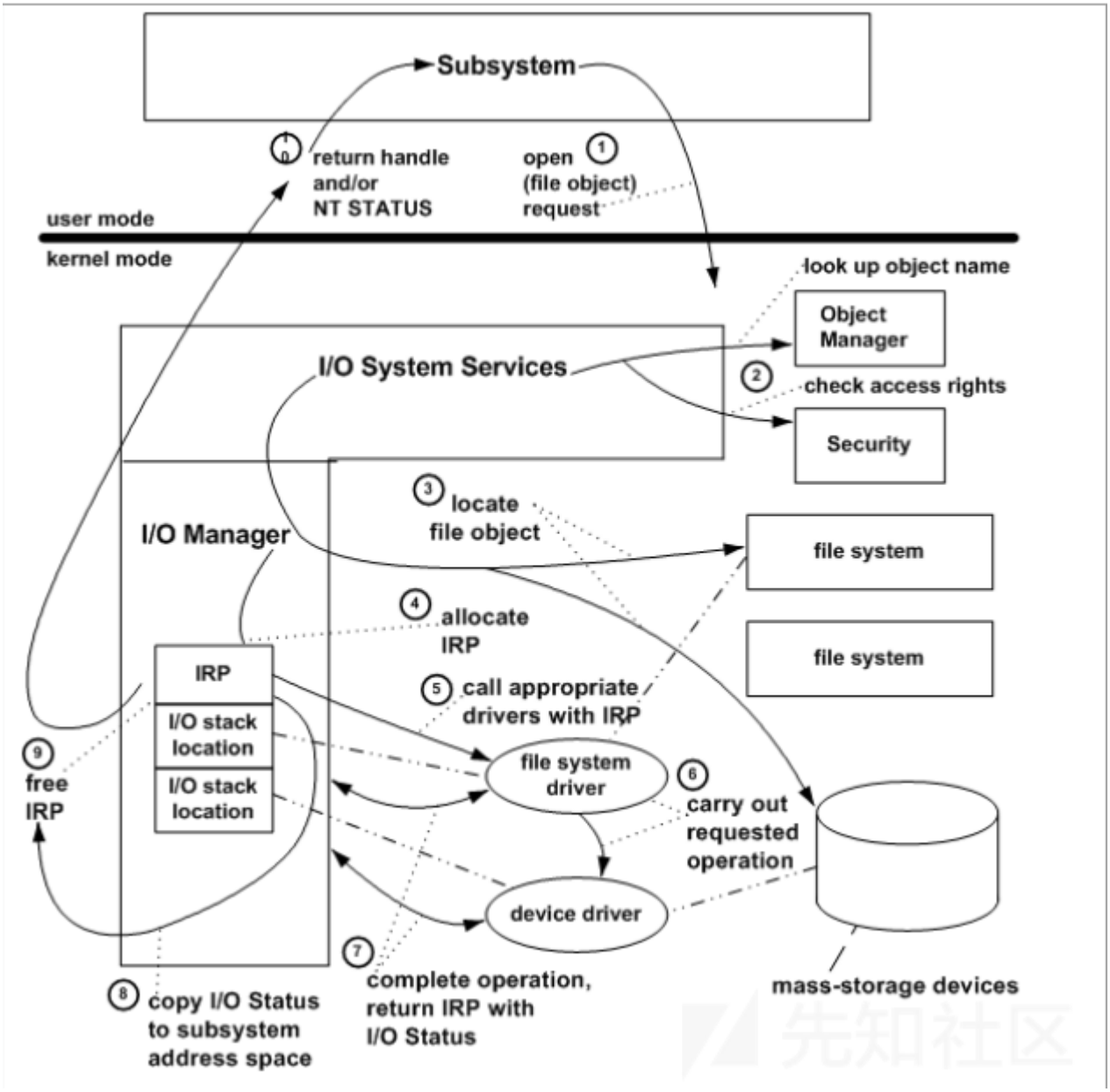
C++ 复制

```
typedef struct _IRP {
  CSHORT                    Type;
  USHORT                    Size;
  PMDL                      MdlAddress;
  ULONG                     Flags;
  union {
    struct _IRP     *MasterIrp;
    __volatile LONG IrpCount;
    PVOID           SystemBuffer;
  } AssociatedIrp;
  LIST_ENTRY                ThreadListEntry;
  IO_STATUS_BLOCK           IoStatus;
  KPROCESSOR_MODE           RequestorMode;
  BOOLEAN                   PendingReturned;
  CHAR                      StackCount;
  CHAR                      CurrentLocation;
  BOOLEAN                   Cancel;
  KIRQL                     CancelIrql;
  CCHAR                     ApcEnvironment;
  UCHAR                     AllocationFlags;
  PIO_STATUS_BLOCK          UserIosb;
  PKEVENT                   UserEvent;
```


任何内核模式程序在创建一个IRP时，同时还创建了一个IO_STACK_LOCATION 数组结构：数组中的每个堆栈单元都对应一个将处理该 IRP 驱动程序。头部有 IO_STACK_LOCATION 数组索引，同时也有一个指向该 IO_STACK_LOCATION 的指针。

索引是从1开始，没有0。索引不会设置成 0，否则系统崩溃，底层驱动不调用 IoCallDriver 。驱动程序准备向次低层驱动程序传递IRP时可以调用 IoCallDriver ，工作是递减当前 IO_STACK_LOCATION 索引，使之与下一层的驱动程序匹配。

IoGetCurrentIrpStackLocation 函数就能过获取到当前设备的IO栈，下图是IRP处理的过程：



用户层通过系统I/O打开文件名，进入内核层由I/O管理器去调用对象管理解析符号链接，这里还会做安全权限检测被打开的对象权限，I/O管理器初始化分配内存IRP，同时创

IRQL:中断请求级别(Interrupt ReQuest Level,IRQL)
用来保证进程的优先级，原子，级别又Dispatch，APC与Passive，这里不做详细说明，后续有时间写一下详细的IRP处理过程.

上述我们已经说了获取串口设备及过滤设备创建与绑定，下面就是获取串口传输的数据，实现自定义的功能，我们如何获取串口数据呢？当然是派遣函数与IRP了。
需要注意:你并知道串口设备使用的那种通讯协议缓冲区进行的数据交换，可能是直接、间接或者 DO_DIRECT_IO
方式。默认是UserBuffer，但是最不安全，一般都会用MdlAddress，但是你需要把所有可能都写上，做好兼容性处理，这是必须做的，否则可能截获不到数据！

```
// ■■■■■IO■
PIO_STACK_LOCATION irpsp = IoGetCurrentIrpStackLocation(irp);
PUCHAR buf = NULL;
// ■■■■■■■■■■■■■■■■■■IRP_MJ_CREATE or IRP_MJ_WRITE
if (irpsp->MajorFunction == IRP_MJ_WRITE)
{
   // ■■■DO_DIRECT_IO ■■■■■■■■■■■■■■■■■■
   if (irp->MdlAddress != NULL)
   {
       buf = (PUCHAR)MmGetSystemAddressForMdlSafe(irp->MdlAddress, NormalPagePriority);
   }
   // irp->AssociatedIrp.SystemBuffer
   // irp->UserBuffer;
   ULONG Writesize = irpsp->Parameters.Write.Length;

   // ■■■■■■■■■■
   for(ULONG i = 0; i < Writesize; ++i)
   {
```

```
        KdPrint(("%2X ", buf[i]))■
        if(!(Writesize % 16))
            KdPrint(("\r\n"))
    }
    IoSkipCurrentIrpStackLocation(irp)
    // ■■■■ IoCallDriver ■■ PoCallDriver ■■■
    // ■■■■■■■■■■■■■■■■■■■■■■
}

// ■■■■
irp->IoStatus.Information = 0;
irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
IoCompleteRequest(irp, IO_NO_INCREMENT);
return STATUS_ERROR;
```

网上串口过滤代码已经很完善，我们这里主要聊的是过程与思路，Windows设备栈基于这种模式去做的，代码大同小异，以前学习的时候也都是参考书籍与帖子，这里不在

很多知识点没有细讲，如IRP的调用过程，MDL映射，通讯机制，IRPQ级别处理机制等等，后续有时间补一篇相关得零散的知识点分享。

参考：
《Windows驱动开发技术详解》
《寒江独钓-Windows内核安全编程》

上一篇：泽少个人渗透系统 7.0版 - 正式发布 下一篇：Joomla 3.4.6 RCE 分析

1. 0 条回复
   - 动动手指，沙发就是你的了！