WIN95 下虚拟设备驱动程序设计开发

成都西南电子电信技术研究所六处三室(610041) 刘其锋

摘 要：介绍虚拟设备驱动程序开发的基本知识以及 VxDs 与 WIN32 应用程序通讯的几种常用方法，并给出了用 VtoolsD 开发 VxDs 的具体实例。

关键词：虚拟设备 虚拟设备驱动程序(VxDs) DDK VTOOLSD

Windows 自面世以来，即以其强大而友好的图形界面占据了操作系统(尤其是微机操作系统)的霸主地位。但是由于 Windows 采取的保护措施屏蔽了系统的底层操作，对用户而言，已不能象在 DOS 下一样直接操纵使用系统的硬件资源如内存、I/O 端口、中断及 DMA 等。这在很大程度上保证了操作系统的安全稳定性能，但却给广大硬件及底层软件开发人员带来了困难。当需要直接操纵控制具体的硬件设备时，往往需要编写设备驱动程序。

1 VMM、VMs 和 VxD

Win95 操作系统支持多线程、多任务应用，正是依靠虚拟机管理器 VMM(Virtual Machine Manager)和虚拟设备驱动程序 VxDs(Virtual Device Drivers)一起来实现设备虚拟化，从而使多个应用能够同时执行。设备的虚拟化是建立在虚拟机假想的基础上的，假想认为每一个 VM 都可以对所有的硬件设备(如内存、I/O 端口、中断等)进行独占的控制，就如 DOS 下的应用程序一样。VxDs 即为实现此假想而产生。

Windows 中有两种虚拟机 VMs：DOS VM 和 System VM。每运行一个 DOS 程序将产生一个 DOS VM，即不同的 DOS 程序运行在各自的 DOS VM 中，而所有的 Windows 应用程序(包括 WIN16 和 WIN32 应用)都运行于系统虚拟机中。每个 VM 有自己的地址空间、I/O 端口、中断向量表，VM 为应用提供内存保护、虚拟内存和权限检查等，只是在 WIN16 中应用程序运行在同一地址空间中，而每个 WIN32 应用程序都拥有自己的地址空间。所有的 VM 都接受 VMM 的统一调度管理，虽然 VMM 本身不是一 VM，但却充当着激活 VMs 和 VxDs 的主要管理员(例如，VMM 要处理在运行 VMs 时的抢占时间片工作)。VMM 是操作系统的核心，VMM 提供单线程、抢先多任务处理。它是一个 32 位保护模式操作系统，它的主要任务是产生、运行、控制和终止虚拟机 VMs，它让多个 VMs 共享 CPU 的时间以使多应用能同时运行，其实 VMM 本身即是一些 VxDs 的集合，VMM 不可重入。

VxDs 在很大程序上支持 Windows 3.X 和 Windows9X，可以将它看作是运行在系统特权级上的特殊的 DLL，VxDs 可直接操纵系统的硬件资源，甚至可以虚拟根本不存在的硬件，Windows 利用虚拟设备来管理软件及硬件设备以确保应用程序不互相干扰。大多数虚拟设备用来管理硬件设备，有些虚拟设备也提供软件服务，而不对应于具体的硬件设备。如 1999 年初猖狂蔓

延的CIH病毒，之所以能破坏硬件资源（通过攻击主板的Flash Memory，达到破坏硬件的目的），正是因为它利用了VxD技术，运行在系统Ring 0级。

在Win3.X中VxDs是静态装载的，也就是说，当Windows启动时要装载所有要用的VxDs，它们将在Windows执行生命期间一直处于活动状态，Win9X（以及Windows for workgroups 3.11）在支持静态VxDs的同时允许动态装载卸下VxDs。当一应用程序用CreateFile()函数存取一VxD时，系统会跟踪每个VxD打开了多少句柄。当应用程序终止时，它要调用函数CloseHandle()释放这个VxD所打开的句柄。当应用程序终止时，它要调用函数CloseHandle()释放这个VxD的句柄，这样即在需要时加载相应的VxD、用完后卸下，提高了系统资源的利用率。当然在进程消亡时，与其相联的句柄会被自动释放。

VxD处理硬件设备时一个经常用到的硬件资源就是硬件中断，VxD处理硬件中断时，虚拟设备很少直接截取中断，而是依赖虚拟可编程中断控制器（VPICD），VPICD实现了物理可编程中断控制器（PPIC）的虚拟化，VPICD发送硬件中断信号给其他虚拟设备，并向虚拟设备提供请求中断、模仿硬件中断等服务。

2 VxD设计开发

2.1 开发工具

开发VxD需要专门的开发工具，目前应用广泛的工具主要有两大类，一类是Microsoft提供的对应于不同版本的Windows的DDK(Device Driver Kit)，其中包含了开发VxD所需的各种类库及汇编工具等。由于用DDK开发VxD大多使用汇编语言，所以开发起来相对比较困难。另一类是Vireo Software提供的VtoolsD，VtoolsD开发包提供了对VxD编程的全线C++类库支持，利用VtoolsD中的Quick VxD工具可以快速生成VxD的代码框架，开发者可以在此基础上根据各自的需要添加自己的代码，使用VtoolsD可以不需要DDK。比较而言，使用VtoolsD要比使用DDK简单、快捷。另外，由于VxD运行在Ring()级，开发VxD时好的调试工具也很重要，常用的有NuMega公司的SoftICE和Microsoft公司的WDEB386等。

2.2 VxD与WIN32应用间的通讯

运行在系统特权级的VxD在处理系统底层设备的同时，大多要为上层应用程序提供服务或调用，于是VxD与应用程序之间的相互通讯在设计开发过程中显得尤为重要。VxD与应用程序之间的通讯包括从VxD到应用程序和从应用程序到VxD的双向通讯。WIN32应用程序可以通过调用DeviceIoControl函数向VxD发送W32_DeviceIoControl消息，VxD中的消息处理函数OnW32DeviceIoControl处理此消息，其中消息参数可用来传递双向信息。反过来，由VxD向应用程序的通讯就相对复杂得多，其中常用的几种方法有（以下的应用程序皆指WIN32应用程序）：

a.使用SHELL服务Shell_PostMessage或Shell_CallAtAppTime等。Shell_CallAtAppTime使得系统在应用程序时间事件触发时调用事先设定的Ring3回调过程，此方法可用来调用动态链接库DLL中的函数，并且Shell_CallAtAppTime可以在中断处理时调用；Shell_PostMessage可以用来从VxD中向Ring3应用发送消息，此方法简单实用，只是发送给应用程序的消息因处于线程的消息循环中可能造成一定的时延，Shell_PostMessage不能在中断处理时调用，替代的方法是在全局事件(GlobalEvent)函数中调用Shell_PostMessage向Ring3应用程序发送消息，需要传给VxD的有消息处理窗口句柄及消息标识。

b.使用APC(asynchronous procedure call)异步过程调用。使用APC调用方法，应用程序将回调函数的地址传给VxD，然后应用程序执行SleepEx(或WaitForMultipleObjectsEx、WaitForSingleObjectEx)使其处于警觉等待(alertable wait)状态，这样当VxD调用VWIN32_QueueUserApc服务时即可触发Ring3应用的回调函数，同样此服务不能在中断服务中调用，但可以在事件服务中使用。

c.使用WIN32事件。WIN32事件在多线程应用程序中很常见，主要用来在多个线程间实现通讯和资源同步。这一方法同样可以用来在VxD和Ring3应用程序间通讯，在VxD中利用VWIN32_ResetWin32Event、VWIN32_SetWin32Event、VWIN32_WaitSingleObject等服务来对特定的事件进行控制，从而唤醒等待的WIN32应用线程或者等待被应用程序唤醒。应用程序可用一子线程来等待VxD的事件，其使用方法类似于应用程序中的多线程处理，只是VxD和应用程序分别使用的是Ring0级和Ring3级事件句柄，所以应用程序不能直接将应用级事件句柄传给VxD，而是要用WIN32 API函数OpenVxDHandle将应用级事件句柄转换成Ring0级事件句柄，然后将此句柄传给VxD使用。由于OpenVxDHandle函数是未公开的API，所以使用时涉及到DLL的调用等稍微复杂的方法。

以上是VxD与WIN32应用程序通讯的几种主要方法，但在开发中断处理VxD时则要注意，由于硬件中断是异步事件，必须遵守中断处理的各项规则，硬件中断处理时系统服务要慎用，当然可以在VxD的硬件中断事件处理函数(Hw_Int_Proc)中使用全局事件，或者在虚拟中断处理函数(Virt_Int_Proc)中使用适当的方法来实现。

2.3 VxD开发实例

下面给出一个用VtoolsD开发的虚拟设备驱动程序的实例。该驱动程序用来处理中断IRQ10，使用

APC 异步过程调用的方法实现 VxD 和 WIN32 应用之间的通讯。当中断发生时触发全局事件，在全局事件处理函数中使用 APC 方法 VWIN32_QueueUserApc 来调用 Ring3 级应用过程，在 Ring3 级的 APC 过程中可根据需要编写相应的处理代码。该 VxD 是一动态加载的驱动程序，WIN32 应用需要传递给 VxD 的主要参数有 APC 过程的指针。在 WIN32 应用中可以创建一线程专门用来处理与 VxD 的通讯等事务，通过 DeviceIoControl 函数将 APC 过程的指针传给 VxD，尔后线程 SleepEx 即可。My.vxd 的部分源码如下。

(1) My.h - include file for VxD My

```
…
#define My_IRQ 10 //IRQ 中断号
class MyHwInt:public VHardwareInt
{
public:
    MyHwInt():VHardwareInt(MY_IRQ,VPICD_OPT_CAN_SH-
    ARE,0,0){}
        virtual VOID OnHardwareInt (VMHAN-
DLE);
};
class MyDevice:public VDevice
{
public:
    virtual BOOL OnSysDynamicDeviceInit();
    virtual BOOL OnSysDynamicDeviceExit();
    virtual DWORD OnW32DeviceIoControl (PI-
OCTLPAR-
AMS pDIOCParams);
    MyHwInt *pMy IRQ;
};
…
class MyEvent:public VGlobalEvent
{
public:
    MyEvent(VOID);
    VOID handler (VMHANDLE hVM,
CLIENT_STRUCT* pRegs,PVOID refData);
};
```

(2) My.cpp - main module for VxD My

```
…
PVOID OpenFileApc=0;                    //APC 过程指针
THREADHANDLE TheThread=0;//Ring0 线程句柄
MyEvent::MyEvent(VOID):VGlobalEvent(0){}
VOID MyEvent::handler(VMHANDLE hVM,CLIENT_STRUCT*
  pRegs,PVOID refData)
{                                       // 事件处理函数
    VWIN32_QueueUserApc (OpenFileApc,0,
TheThread);
                                        //异步过程调用
}
VOID MyHwInt::OnHardwareInt (VMHANDLE
hVM)
{
    (new MyEvent())->call();        //全局事件调度
    sendPhysicalEOI();//通知 VPICD 中断处理结束
}
…
    BOOL MyDevice::OnSysDynamicDeviceInit()
{
    pMyIRQ=new MyHwInt();
    VEvent::initEvents();           //初始化事件堆
    return TRUE;
}
BOOL MyDevice::OnSysDynamicDeviceExit()
{
    delete pMyIRQ;
    return TRUE;
}
DWORD MyDevice::OnW32DeviceIoControl(PIOCTLPAR-
AMS p){          //与 Win32 应用程序的接口函数
    switch(p->dioc_IOCtlCode)
    {
    …
    case 111:
        OpenFileApc=*(PVOID*)p->dioc_InBuf;
            //从 WIN32 传入的 APC 过程指针
        TheThread=Get_Cur_Thread_Handle();
                        //获得 Ring()级线程句柄
        pMyIRQ->hook();          //挂接中断
        pMyIRQ->physicalUnmask();//解除中断屏蔽
        break;
    }
    return 0;
}
```

参考文献

1 杨 强,李堂秋编著.Win9X 虚拟设备驱动程序编程指南.北京:清华大学出版社,1999

2 (美)Vireo Software 公司著.VTOOLSD User's guide.Concord,MA,Vireo Software 1994~1997

(收稿日期:1999-09-28)