Windows WDM 驱动开发

我们的技术是您的！

Wyouken & O4icwin

2005 年 11 月

Rev. 0.1

Revision History

Revision	Comments	Issue Date	Author
V0.1	第一部分发布	11/10/2005	Wyouken & O4icwin

目录

序言

驱动开发是多数程序员的梦想，但是，由于硬件条件和工作环境的限制，很多朋友都没能真正的安下心来写一个驱动。

80 年代以前的程序员们个个身怀绝技，软硬兼通，从最顶层的应用到最底层的 IO 控制都可能是一个人实现的。正是他们那种需要高智商和高投入的工作让很多人都不能挤进他们的行业，所以程序员的名声就由他们打响的。

面向对象的编程让我们开发程序变得容易，程序员越来越多，工作的压力越来越大，让人们随时都有危机感。所以一部分人开始转向嵌入式开发，硬件逻辑设计等等方面。

现在由几颗芯片加起来就是产品的功能非常的有限，模块化的设计思想和用户多元化的需求让开发商们不得不做出更多接口丰富的产品。

驱动开发让很多人感觉复杂，望而生畏，其实只要你只要用心的去做，掌握驱动开发的方法与技巧，您就能感觉到：驱动程序的开发其实跟开发普通应用程序一样简单容易。

实践出真知，我们写这篇文章的目的很简单，也没有利益的驱动，我们只是将我们学习驱动的经历浅薄的写出来，只要您是有心人，此文将能最大限度的让您快速入门。

入门后的造化都靠您们自己，我们欢迎大家一起讨论和进步。

如果您有好的文章或建议，请 Email 给我们！

Wyouken， O4icwin
2005-10-13

第一章 概述

1.1 本教程的规划：

本教程由 icwin 的 wyouken 和 o4icwin 编写。
目的就是对 windows 驱动开发感兴趣的朋友给出一个入门教程，本教程是一个系统的循序渐进的教程。在目前国内这方面的资料能让开发者很快拿起来就开发得资料比较少，同时很多人没有时间或不原意共享他们学习过程中的点滴。

本文的过程是 icwin 在学习过程中的一些经验心得，icwin 将从实用应用系统开发的角度来一步步的去讲解如何实现 WDM 驱动的开发。

如果你对 FPGA 开发感兴趣，让我们从实际的开发应用开始吧，只有实际的开发，读者才能真正快速的了解和开发出自己希望开发出来的东西。

阅读本书的条件：（阅读者必须具备的能力）

*熟练使用 VC，熟练设计基于 C 的程序

Icwin 书写本文的目的：

*阅读者能在一天内了解驱动程序的基本架构，应用程序调用驱动的方法，安装与卸载驱动，驱动运行的流程

*并 DEBUG 一个 WDM 驱动 DEMO 程序，掌握驱动执行流程

第二章 WDM 驱动程序的运行

这篇文章主要从速成方面指导读者快速拿起驱动开发的武器。

2.1 WDM 驱动程序的基本调用流程：

WDM 驱动程序的的调用其实也是基于消息的。IRP_MJ_*， IRP_MN_*等都是可以理解成消息的（我觉得功能码的叫法不利于我们搞 windows 程序开发的啊，呵呵）。

2.1.1 驱动程序何时从何处开始执行？

2.1.1.1 第一次安装好驱动程序：

驱动程序都可从正确安装成功或更新后就会被操作系统调用，并从 DriverEntry()开始执行。

但是热拔插的设备是在插拔时才装载驱动程序，为什么呢？

*热插拔的设备都有一个 ID，比如 USB 的设备由 VID,PID

*当设备插入时，计算机会枚举设备要求设备提供 VID，PID，配置描述符等

*根据 VID 和 PID 计算机会查找是否已经装过驱动程序，如果没有装过，就提示新设备要装驱动程序了；如果已经装了，就由系统装载驱动程序并开始从 DriverEntry（）执行（如果有正确的调用 DbgPrint 就可以看到驱动程序执行的流程能了）

2.1.1.2 驱动程序正常运行：

一般情况下，驱动程序从 windows 检查到设备的拔插开始，由操作系统调用执行，进入我们驱动代码的入口是 DriverEntry()

2.1.2 DriverEntry()大约做些什么？

*DriverEntry	初始驅動程式之進入點, 建立回呼函式 (Callback Routines) 位址對應表
*AddDevice	加入一個新的裝置到系統中 (Device Object)
*Unload	移除驅動程式
StartIo	序列化處理 IRP
*Dispatch Routines	處理 IRP_MJ_XXX, IRP_MJ_PNP, IRP_MJ_POWER等命令
ISR	中斷服務函式
IoCompletion Routine	當低層驅動程式完成一個IRP工作時, 可呼叫上層所預設的一個函式

DriverEntry()相当于 C 程序的 main（）,Winmain()等函数，也相当于 DLL 的 DLLMain（）函数，它是驱动程序运行的入口，不由用户调用而由操作系统调用。

DriverEntry()函数主要注册一些方便我们的应用驱动程序调用的接口（也即指针函数—俗称回调函数）

如代码：

```
#pragma code_seg("INIT") // start INIT section
extern "C"
NTSTATUS DriverEntry(   IN PDRIVER_OBJECT DriverObject,
                        IN PUNICODE_STRING RegistryPath)
{
    NTSTATUS status = STATUS_SUCCESS;
#if DBG
    DebugPrintInit("Wdm1 checked");
#else
    DebugPrintInit("Wdm1 free");
#endif
    DebugPrint("RegistryPath is %T",RegistryPath);
    // Export other driver entry points...
    DriverObject->DriverExtension->AddDevice = Wdm1AddDevice;
    DriverObject->DriverUnload = Wdm1Unload;

    DriverObject->MajorFunction[IRP_MJ_CREATE] = Wdm1Create;//for
CreateFile
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = Wdm1Close; //for
CloseHandle

    DriverObject->MajorFunction[IRP_MJ_PNP] = Wdm1Pnp;
    DriverObject->MajorFunction[IRP_MJ_POWER] = Wdm1Power;
```

```
    DriverObject->MajorFunction[IRP_MJ_READ] = Wdm1Read;//for ReadFile
    DriverObject->MajorFunction[IRP_MJ_WRITE] = Wdm1Write;//for
WriteFile
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
Wdm1DeviceControl;

    DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] =
Wdm1SystemControl;

    //  Initialise spin lock which protects access to shared memory buffer
    KeInitializeSpinLock(&BufferLock);

    DebugPrintMsg("DriverEntry completed");

    return status;
}
#pragma code_seg() // end INIT section
```

如上面的代码所时。
回调函数分为 主功能 和 次功能 两种
主功能消息有：

IRP 功能代碼 (See WDM.H)

2.1.2.1 IRP 主功能码（Major Function Code）

主功能代碼 (Major Function Code)

- #define IRP_MJ_CREATE 0x00
- #define IRP_MJ_CLOSE 0x02
- #define IRP_MJ_READ 0x03
- #define IRP_MJ_WRITE 0x04
- #define IRP_MJ_DEVICE_CONTROL 0x0e
- #define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
- #define IRP_MJ_SHUTDOWN 0x10
- #define IRP_MJ_POWER 0x16
- #define IRP_MJ_PNP 0x1b
- #define IRP_MJ_PNP_POWER IRP_MJ_PNP

2.1.2.2 IRP_MJ_PNP 次功能码（Minor Function Code）

IRP_MJ_PNP 次功能代碼

- 次功能代碼 (Minor Function Code)
- #define IRP_MN_START_DEVICE 0x00
- #define IRP_MN_QUERY_REMOVE_DEVICE 0x01
- #define IRP_MN_REMOVE_DEVICE 0x02
- #define IRP_MN_CANCEL_REMOVE_DEVICE 0x03
- #define IRP_MN_STOP_DEVICE 0x04
- #define IRP_MN_QUERY_STOP_DEVICE 0x05
- #define IRP_MN_CANCEL_STOP_DEVICE 0x06
- #define IRP_MN_SURPRISE_REMOVAL 0x17

2.1.2.3 IRP_MJ_POWER 次功能码（Minor Function Code）

IRP_MJ_POWER 次功能代碼

- #define IRP_MN_WAIT_WAKE 0x00
- #define IRP_MN_POWER_SEQUENCE 0x01
- #define IRP_MN_SET_POWER 0x02
- #define IRP_MN_QUERY_POWER 0x03

2.1.3 驱动程序与应用程序相关的功能码如何调用？

这里仅以 CreateFile（）函数的调用举例，其他的在后面已表格的形式列出

2.1.3.1 DriverEntry()中您必须要注册回调函数

说明：CreateFile（）将向操作系统发送 IRP_MJ_CREATE 消息。

```
DriverObject->MajorFunction[IRP_MJ_CREATE] = Wdm1Create;
```

上面的代码是在 DriverEntry()实现的，它定义了响应操作系统 IRP_MJ_CREATE 消息码的消息处理函数 Wdm1Create（）：

Wdm1Create（）的声明如下，都有标准的接口的，您也不用害怕
NTSTATUS Wdm1Create(IN PDEVICE_OBJECT fdo,IN PIRP Irp);

2.1.3.2 在您的应用程序中正确调用 CreateFile（）

如下：

```
HANDLE rv = CreateFile( ifDetail->DevicePath,
    GENERIC_READ | GENERIC_WRITE,
  FILE_SHARE_READ | FILE_SHARE_WRITE,
  NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

这一调用将触发 IRP_MJ_CREATE 消息，进而就可以调用到 Wdm1Create 了。

2.1.3.3 应用层调用驱动的消息参照：

Win32函式	功能說明	I/O 管理員發送之 IRP 命令
CreateFile(...)	開啓I/O裝置	IRP_MJ_CREATE
CloseHandle(...)	關閉I/O裝置	IRP_MJ_CLOSE
ReadFile(...)	讀取I/O裝置之資料	IRP_MJ_READ
WriteFile(...)	將資料寫入I/O裝置	IRP_MJ_WRITE
DeviceIoControl(...)	對I/O裝置進行特殊控制	IRP_MJ_DEVICE_CONTROL

2.1.3.4 IoControl 调用：

*Iocontrol 是可以自己定义的消息，用 CTL_CODE 宏编码成一个 32bit 的数
*应用程序调用 DevieceIoControl（）
*驱动程序会响应 IRP_MJ_DEVICE_CONTROL 并取得 CTL_CODE
从而转去执行对应的处理代码

*CTL_CODE的定义举例

```
#define IOCTL_WDM1_ZERO_BUFFER CTL_CODE(  \
        FILE_DEVICE_UNKNOWN,        \
        0x801,                      \
```

```
            METHOD_BUFFERED,                \
            FILE_ANY_ACCESS)
    *CTL_CODE的应用程序调用举例
        if( !DeviceIoControl( hWdm1, IOCTL_WDM1_ZERO_BUFFER,
                              NULL, 0,    // Input
                              NULL, 0,    // Output
                              &BytesReturned, NULL))
        {
            ……..
        }
```

- CTL_CODE在驱动中的处理IRP_MJ_DEVICE_CONTROL举例

驱动程序回调Wdm1DeviceControl ()

```
      NTSTATUS Wdm1DeviceControl(    IN PDEVICE_OBJECT fdo,
                                 IN PIRP Irp)
{
    PIO_STACK_LOCATION IrpStack = IoGetCurrentIrpStackLocation(Irp);
    NTSTATUS status = STATUS_SUCCESS;
    ULONG BytesTxd = 0;

    ULONG ControlCode = IrpStack->Parameters.DeviceIoControl.IoControlCode; //这里
取得 ControlCode
    ULONG InputLength = IrpStack->Parameters.DeviceIoControl.InputBufferLength;
    ULONG OutputLength = IrpStack->Parameters.DeviceIoControl.OutputBufferLength;

    DebugPrint("DeviceIoControl: Control code %x InputLength %d OutputLength %d",
                ControlCode, InputLength, OutputLength);

    // Get access to the shared buffer
    KIRQL irql;
    KeAcquireSpinLock(&BufferLock,&irql);
    switch( ControlCode)
    {
    ////// Zero Buffer
    case IOCTL_WDM1_ZERO_BUFFER: //处理此 CTL_CODE
        // Zero the buffer
        if( Buffer!=NULL && BufferSize>0)
            RtlZeroMemory(Buffer,BufferSize);//在这里实现您想实现的处理功能
        break;
    }
}
```

好了，基本的我就介绍这么多，在你慢慢的了解中，你很快知道驱动不过如此了

下面我们就进入实战了，如果你在上面的内容有别的好的 idea 请 emailto

icwin@icwin.net

我们共同讨论进步吧！

第三章 开始编写 WDM 驱动程序

3.1 得到一个 Demo 工程：

我们以《windows WDM 设备驱动程序开发指南》的
wdm1 工程开始吧，你可以用你可以使用的任何方法获得此源码包！
文件大概如下：

```
    |   DIRS
    |   Readme.txt
    |   file.txt
    |
    ├──sys
    |   |   DebugPrint.c
    |   |   Dispatch.cpp
    |   |   GUIDs.h
    |   |   Wdm1.opt
    |   |   Ioctl.h
    |   |   MAKEFILE
    |   |   Makefile.inc
    |   |   DebugPrint.h
    |   |   ReadReg.cpp
    |   |   Sources
    |   |   Wdm1.dsp
    |   |   Wdm1.h
    |   |   Wdm1.rc
    |   |   Wdm1checked.inf
    |   |   Wdm1free.inf
    |   |   resource.h
    |   |   Wdm1.001
    |   |   Init.cpp
    |   |   Wdm1.dsw
    |   |   Wdm1.ncb
    |   |   buildchk.log
    |   |   Pnp.cpp
    |   |   Wdm1.plg
    |   |   buildfre.log
    |   |
    |   ├──objfre
    |   |   └──i386
    |   |           wdm1.res
```

```
    |   |           debugprint.obj
    |   |           init.obj
    |   |           dispatch.obj
    |   |           pnp.obj
    |   |           Wdm1.sys
    |   |           Wdm1.pdb
    |   |
    |   ├──objchk
    |   |   └──i386
    |   |           Wdm1.sys
    |   |           wdm1.res
    |   |           debugprint.obj
    |   |           init.obj
    |   |           dispatch.obj
    |   |           pnp.obj
    |   |           Wdm1.pdb
    |   |
    |   ├──obj
    |   |   |   _objects.mac
    |   |   |
    |   |   └──i386
    |   |       ├──free
    |   |       |       Wdm1.sys
    |   |       |
    |   |       └──checked
    |   |               Wdm1.sys
    |   |
    |   └──Free
    └──exe
        |   Wdm1Test.rc
        |   resource.h
        |   Wdm1Test.001
        |   Wdm1Test.dsw
        |   Wdm1Test.ncb
        |   Wdm1Test.aps
        |   Wdm1Test.plg
        |   Wdm1Test.cpp
        |   Wdm1Test.dsp
        |   Wdm1Test.opt
        |
        ├──Release
        |       Wdm1Test.exe
```

```
    |
    └──Debug
            Wdm1Test.res
            vc60.idb
            vc60.pdb
            Wdm1Test.sbr
            Wdm1Test.pch
            Wdm1Test.obj
            Wdm1Test.bsc
            Wdm1Test.ilk
            Wdm1Test.exe
            Wdm1Test.pdb
```

3.2 在 VC 下配置 DDK 的开发环境

VC 的 IDE 能极大地帮助我们快速完成开发！

如果我们不配置 VC 的环境，VC 不能完成 wdm1\sys 下的驱动源码的编译

3.2.1 我的目录

我的机器当前驱动源码的目录 ： H:\driverDev\ （根据您自己的填写）
我的机器 XP_DDK 的目录 ： D:\winddk\2600（根据您自己的填写）

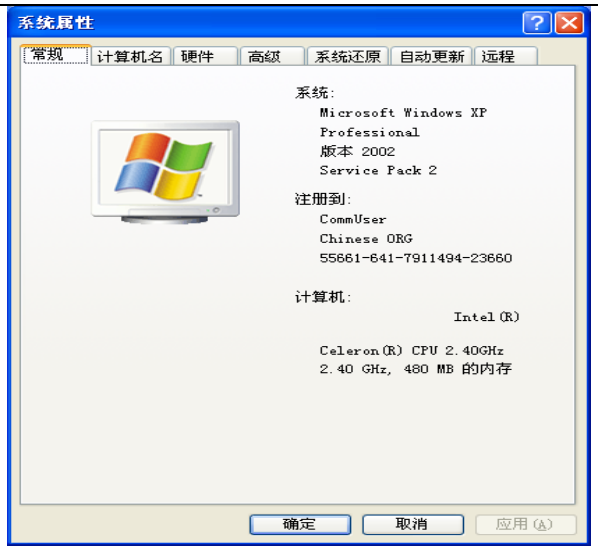
3.2.1.1 我们应该在系统环境变量里设置

设置“驱动源码的目录”和”DDK 的目录”
我的机器当前驱动源码的目录 ： H:\driverDev\ （根据您自己的填写）
我的机器 XP_DDK 的目录 ： D:\winddk\2600（根据您自己的填写）

方法：

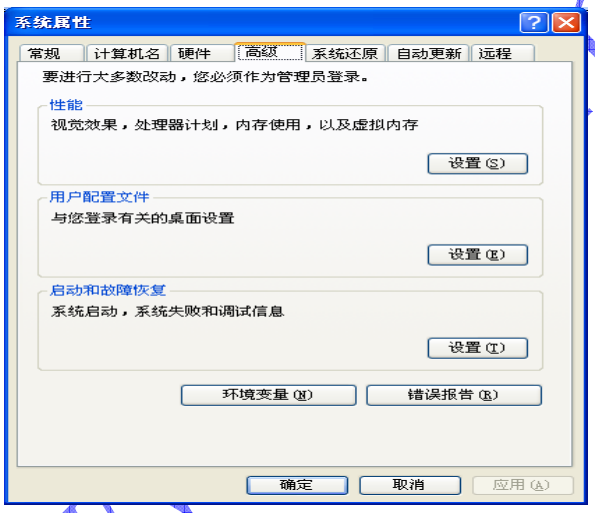
A:

在电脑的桌面上 - 〉 右击“我的电脑” -〉按“R”或点击“属性”出现属性对话框：



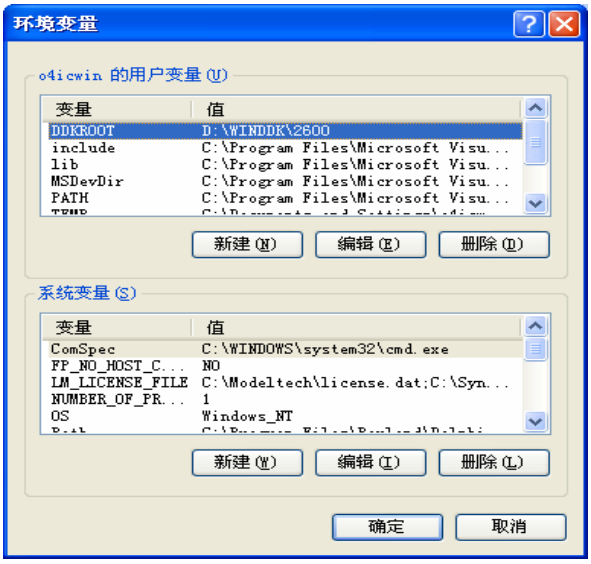
B:

点击高级



C：

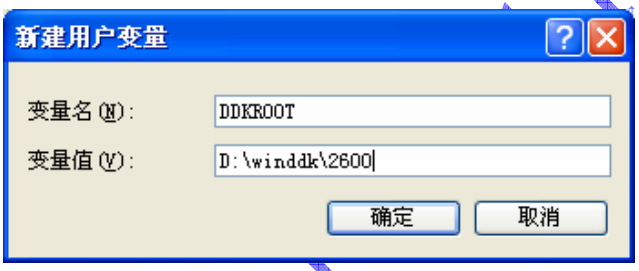
点击“环境变量”



D:

在上面的 用户变量 区点击 “新建”并填写成如下后 按确定

注：变量值是你的 DDK 的目录（bin 的上层目录）



E:

重复 D 的过程 新建 源码的目录的变量，然后一直确定直到关闭属性对话框

3.2.2 安装 VC6

您需要安装 VC6 或其他的 IDE 工具，本文只讨论 VC6 的情况

VC6 的安装很容易，不用多讲了

3.2.3 打开 wdm1\sys\Wdm1.dsp 工作区文件

提示你这个工程被老版本的开发工具生成，要转换成新的格式
你愿意转换吗？

点击 “是”

3.2.4 修改 H:\driverDev\MakeDrvr.bat 文件

为了了解 bat 文件的使用情况，我们可以修改
H:\driverDev\MakeDrvr.bat 修改成：

```
@echo off
@echo DDK_DIR:%1
@echo SRC_ROOT:%2
@echo SRC_PATH:%3
@echo ----------------------------

if "%1"=="" goto usage
if "%3"=="" goto usage
if not exist %1\bin\setenv.bat goto usage
call %1\bin\setenv %1 %4
%2
cd %3
build -b -w -cef %5 %6 %7 %8 %9
goto exit


:usage
echo usage     MakeDrvr  DDK_dir  Driver_Drive  Driver_Dir  free/checked
[build_options]
echo eg      MakeDrvr %%DDKROOT%% C: %%WDMBOOK%% free -cef
:exit
```

在 VC6 界面按下 F7 编译会出现如下信息:

```
--------------------Configuration: Wdm1 - Win32 Free--------------------
'MakeDrvr' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
Error executing c:\windows\system32\cmd.exe.

Wdm1.sys - 1 error(s), 0 warning(s)
```

3.2.5 设置 VC 的环境

3.2.5.1 前面的内容编译时出了错误（配置'MakeDrvr'）

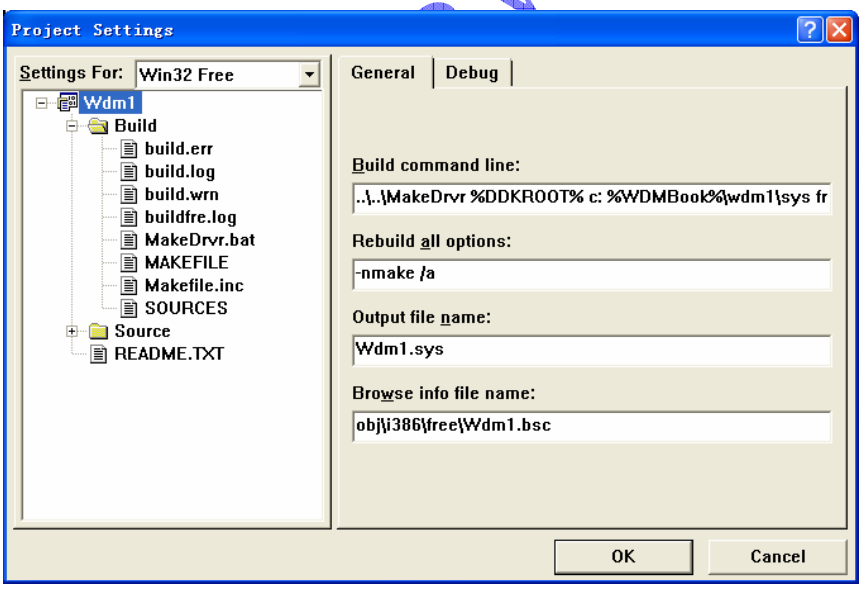
错误提示'MakeDrvr' 找不到时因为 'MakeDrvr'的路径不对，

我们在 VC6 中常做的 有两种解决的办法：

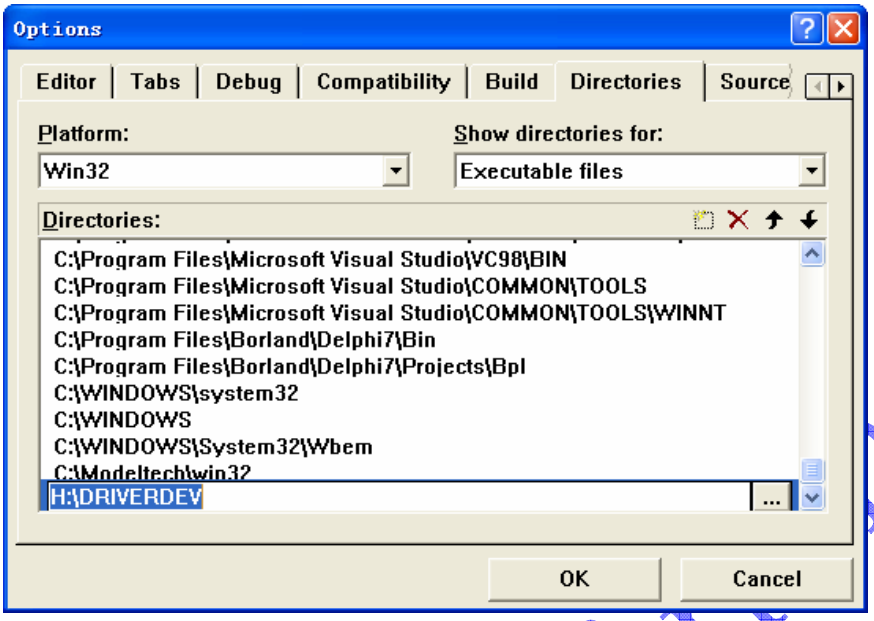
注意：后面的 3.2.5.1.1 和 3.2.5.1.2 的办法可以同时使用不影响，但我推荐用 3.2.3.1.1 的方式

3.2.5.1.1 在 project -> settings 中设置成如下：

把 MakeDrvr.bat 的相对路径加上就可以了



3.2.5.1.2 还可以在 Tools-> Options-〉directories 中选择“Executable files”并添加 MakeDrvr.bat 的目录即可

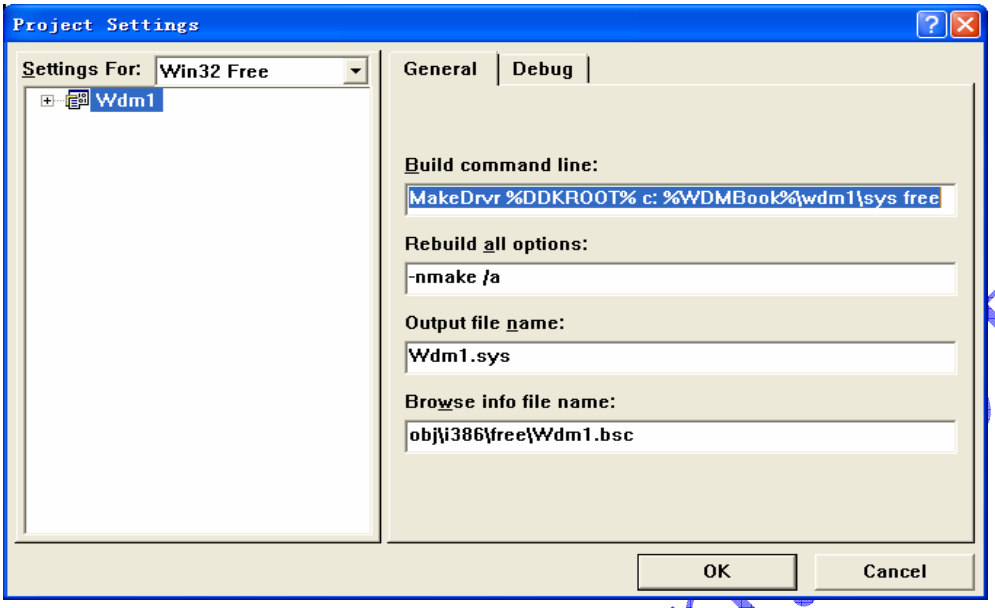


3.2.5.1.3 再按 F7 编译 有编译提示

```
------------------Configuration: Wdm1 - Win32 Free------------------
DDK_DIR:D:\WINDDK\2600
SRC_ROOT:c:
SRC_PATH:H:\driverDev\wdm1\sys
-------------------------------
BUILD: Object root set to: ==> objfre
BUILD: Adding /Y to COPYCMD so xcopy ops won't hang.
BUILD: /i switch ignored
BUILD: Compile and Link for i386
BUILD: Loading D:\WINDDK\2600\build.dat...
BUILD: Computing Include file dependencies:
BUILD: Saving D:\WINDDK\2600\build.dat...
BUILD: Done

Wdm1.sys - 0 error(s), 0 warning(s)
```

还是没有编译成功：
原来 上面的提示 SRC_ROOT:c: 的根部路为 C:
我们应该更改 C： 为我们源代码的 根目录（即把下图的 C： 改成 SRC_PATH:H:\driverDev\wdm1\sys 的根目录 H：）

3.2.5.1.4 搞清楚 MakeDrvr.BAT 文件的功能



对应命令行执行文件的解释如下：

注意上图中蓝色：

Bat 文件参数的传递：

MakeDrvr	%DDKROOT%	c:	%WDMBook%\wdm1\sys	free	
[可执行文件 *.Bat]	DDK 目录	源码根目录	源码工程目录	free/checked	

所以 MakeDrvr.Bat 文件的内容如下：

```
@echo off                    ;关闭显示
@echo DDK_DIR:%1     ;显示 第一个参数 ， 这个是 DDK 的目录
@echo SRC_ROOT:%2        ;显示 第二个参数 ， 这个是 SRC 的 ROOT，方便跳转
目录
@echo SRC_PATH:%3        ; 源码的路径
@echo ------------------------------

if "%1"=="" goto usage
if "%3"=="" goto usage
if not exist %1\bin\setenv.bat goto usage
call %1\bin\setenv %1 %4   ; 这第四个参数是 free 还是 checked 根据你的编
译模式
%2                         ; 跳转到 SRC 的盘，相当于 DOS 下输入 C:〉D:
cd %3                      ; 跳到 当前工程的目录
build -b -w -cef %5 %6 %7 %8 %9 ; 开始 build
goto exit
```

```
        :usage
        echo  usage     MakeDrvr  DDK_dir  Driver_Drive  Driver_Dir  free/checked
        [build_options]
        echo eg      MakeDrvr %%DDKROOT%% C: %%WDMBOOK%% free -cef
:exit
```

3.2.5.2 前面的内容编译时出了错误，让我们看看是什么原因

在 VC6 界面按下 F7，开始编译，得到显示结果为：

```
--------------------Configuration: Wdm1 - Win32 Checked--------------------
DDK: d:\winddk\2600
SRCROOT:c:
SRCPATH:H:\driverDev\wdm1\sys
COMPILE TYPE :checked
COMPILE PARAM :-cef
H:\driverDev\wdm1\sys>if "d:\winddk\2600" == "" goto usage
H:\driverDev\wdm1\sys>if "H:\driverDev\wdm1\sys" == "" goto usage
H:\driverDev\wdm1\sys>if not exist d:\winddk\2600\bin\setenv.bat goto usage
call d:\winddk\2600\bin\setenv d:\winddk\2600 checked
H:\driverDev\wdm1\sys>call d:\winddk\2600\bin\setenv d:\winddk\2600 checked
 build -b -w -cef
BUILD: Object root set to: ==> objchk
BUILD: Adding /Y to COPYCMD so xcopy ops won't hang.
BUILD: /i switch ignored
BUILD: Compile and Link for i386
BUILD: Loading d:\winddk\2600\build.dat...
BUILD: Computing Include file dependencies:
BUILD: Examining H:\driverDev\wdm1\sys directory for files to compile.
    H:\driverDev\wdm1\sys
    H:\driverDev\wdm1\sys - 5 source files (1,824 lines)
BUILD: Saving d:\winddk\2600\build.dat...
BUILD: Compiling H:\driverDev\wdm1\sys directory
Compiling - wdm1.rc for i386
Compiling - debugprint.c for i386
Compiling - init.cpp for i386
Compiling - dispatch.cpp for i386
Compiling - pnp.cpp for i386
Linking Executable - objchk\i386\wdm1.sys for i386
```

```
BUILD: Linking H:\driverDev\wdm1\sys directory
BUILD: Done
    9 files compiled -   912 LPS
    1 executable built

Wdm1.sys - 0 error(s), 0 warning(s)
```

上面列出的结果表明编译成功，已经 LINK 生成了 wdm1.sys

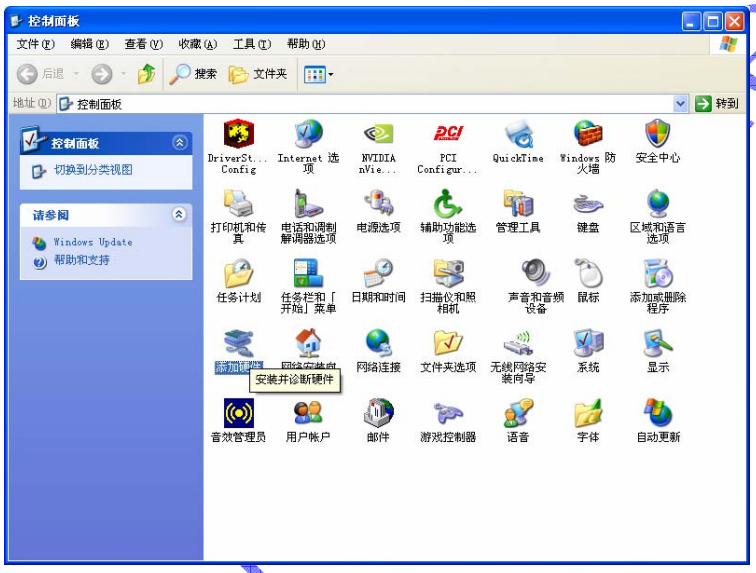
```
Linking Executable - objchk\i386\wdm1.sys for i386
```

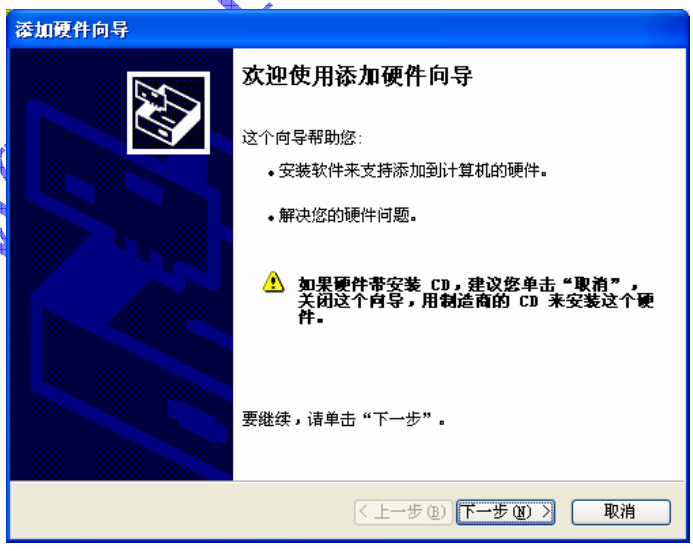
第四章 安装 DebugPrintMonitor 驱动程序

DebugPrintMonitor 是 WDMBook 教程提供的比较好的监视工具，很多公司都使用此作为模板进行开发。

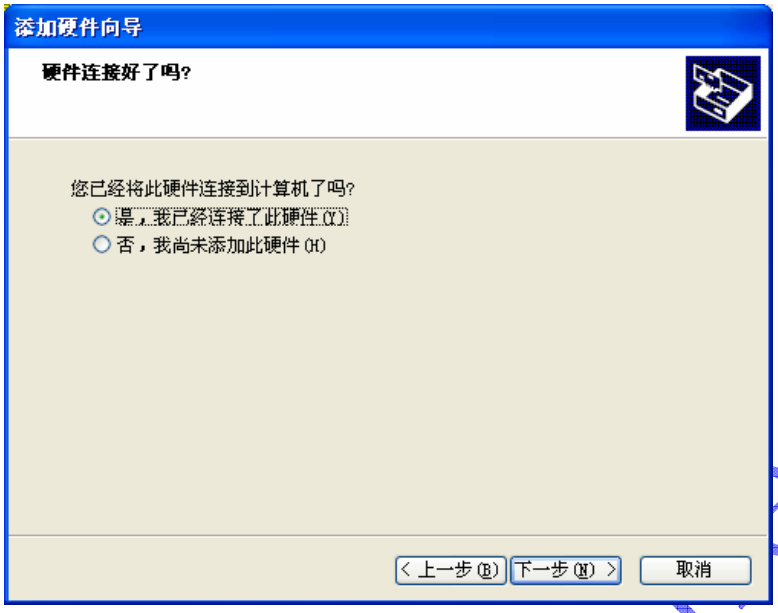
4.1 用控制面板安装 DebugPrintMonitor

打开控制面板，双击 “添加硬件”

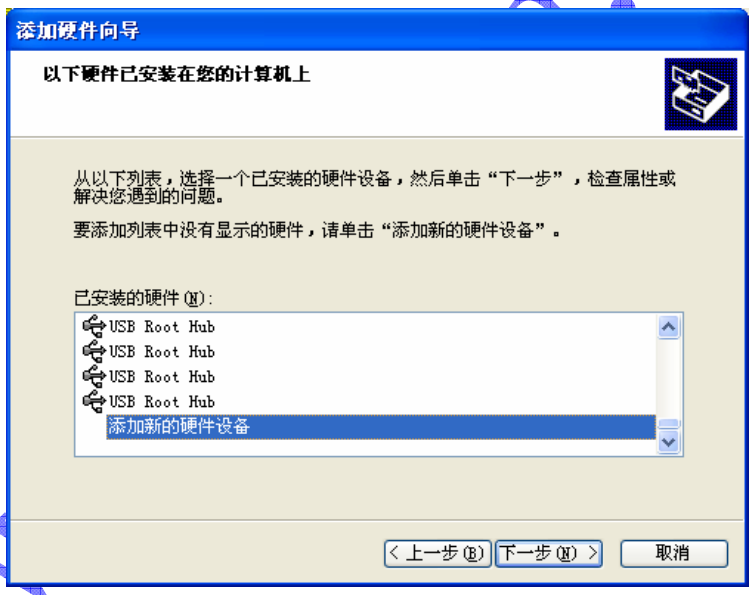




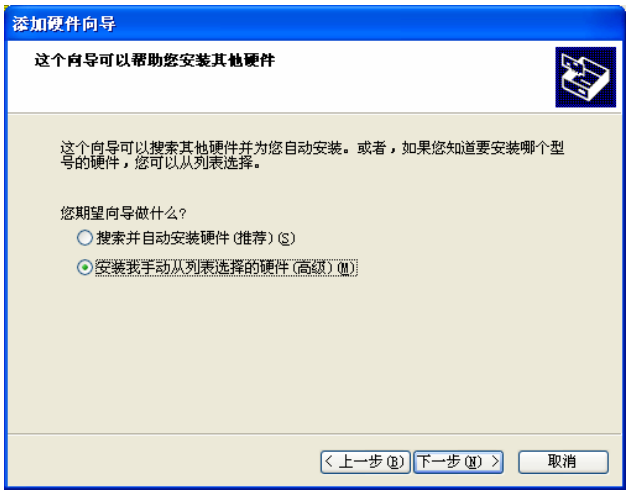
点击下一步 让其查找



选择 “是，我已经连接了次硬件（Y）”
然后点击 “下一步”

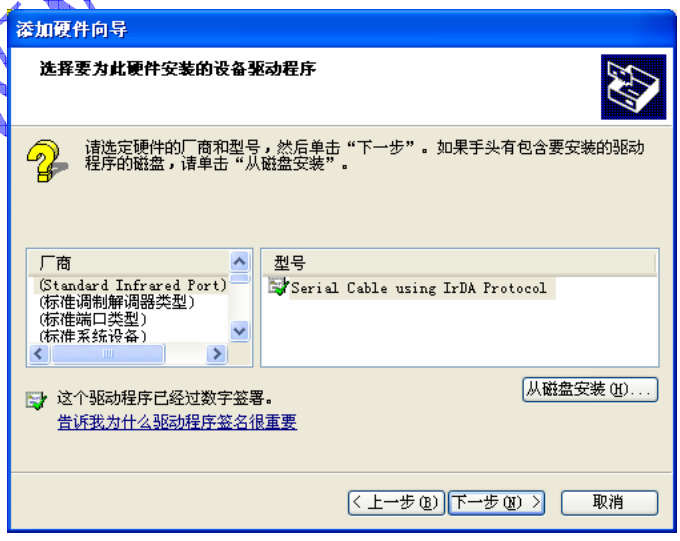


拖动“已安装的硬件” 的列表滑动条至最下边
选择“添加新的硬件设备”

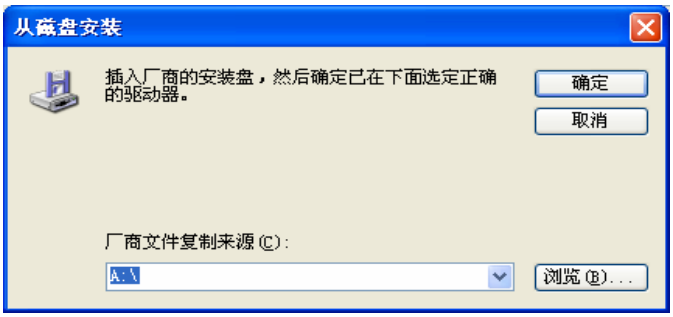


选择 手动安装

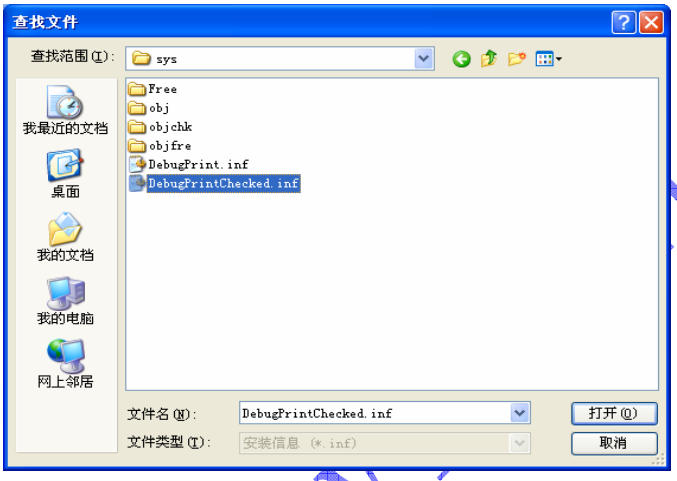
选择 显示所有设备



不用选择 直接 点击“从磁盘安装（H)。。。”

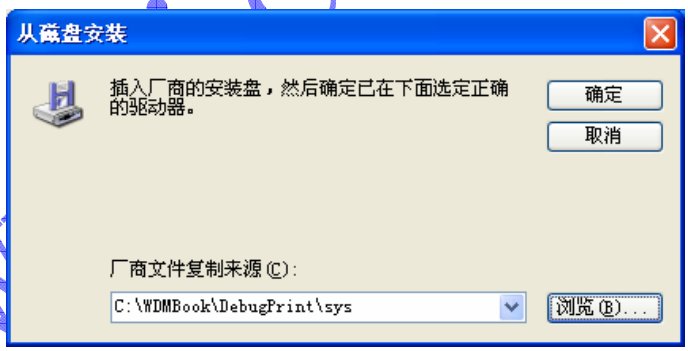


点击 “浏览（B)。。。”

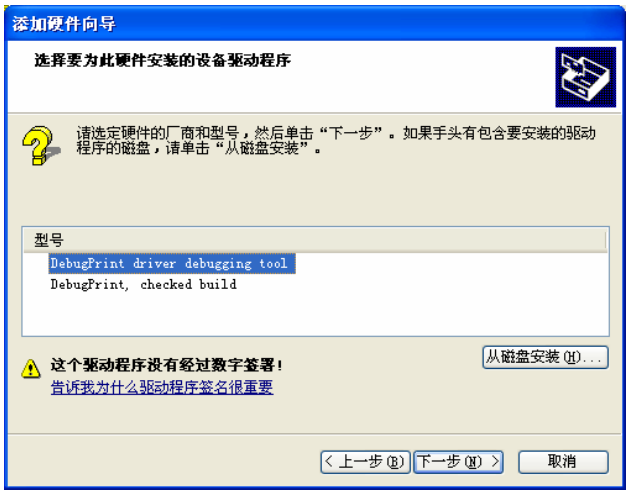


找到 “WDMBook\DebugPrint\sys\DebugPrintChecked.inf”后选中

点击 “打开（0)”

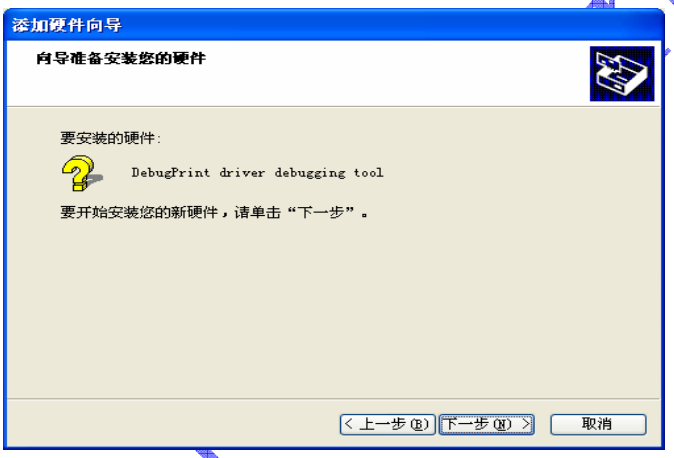


点击确定

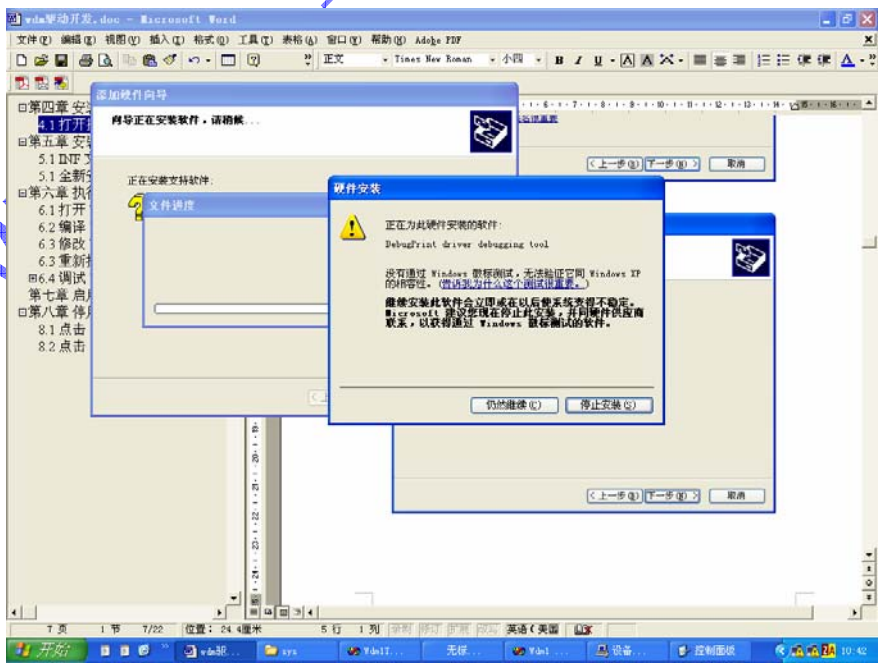


照上图 选择

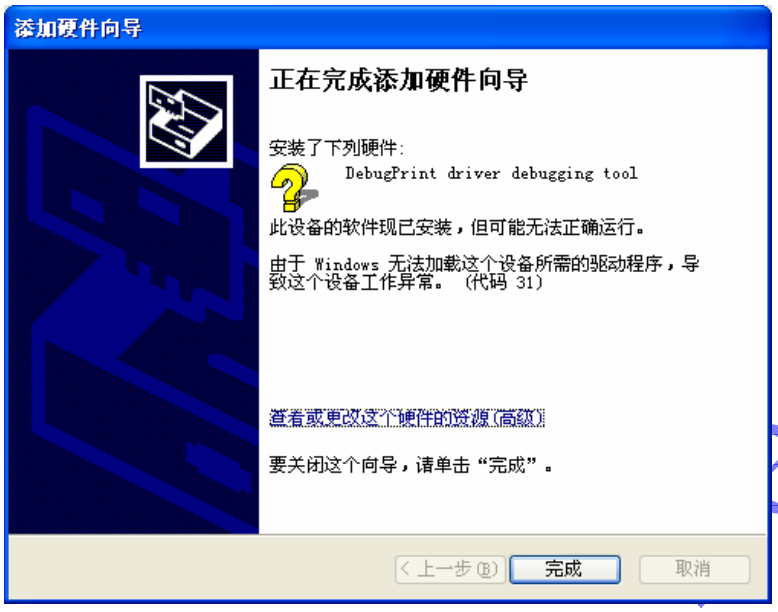
点击 “下一步（N)〉”



点“下一步（N)〉”

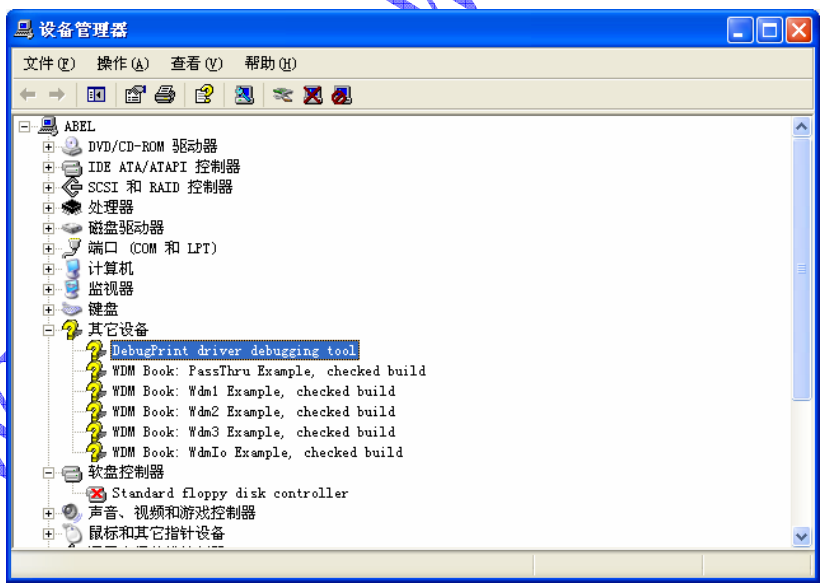


点击 “仍然继续”



点击 “完成” 就完成了

4.2 检查 DebugPrint driver 的安装情况



如图所示

没有 黄色的 警告 和红 X 说明安装成功了

等我们安装完成 wdm1 驱动程序 后我们在测试是不是 安装成功了

第五章 安装 wdm1 驱动程序

*安装 Wdm1 的驱动的方法与 DebugPrint Driver 的方法一样

*打开 DebugPrintMonitor.exe 我们可以监视 wdm1.Sys 的流程
我们编译出来的 wdm1.sys 必须装入内存才能运行，所以我们必须安装它。

安装完成后，DebugPrintMonitor.exe 马上就能得到 wdm1.Sys 的运行情况

5.1 INF 文件

安装驱动程序必须有*.INF 文件，INF 文件主要指出了*.sys 文件相对于*.INF 文件的位置，驱动程序相关的版本、厂商、时间等等 信息。

INF 的书和文章网上有很多，我这里就不写了。

5.1 全新安装驱动

我们的第一个驱动程序需要我们安装，可以做一个安装程序来完成，我们这里用 windows 的设备管理工具来完成。

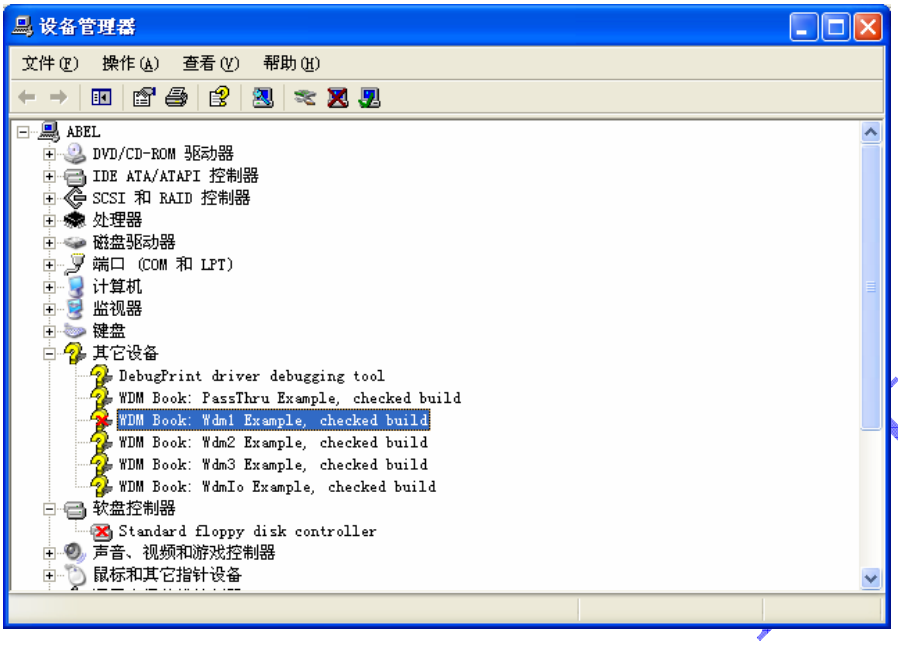
这不是一个硬件设备，我们得用 添加硬件工具来添加我们的驱动程序。硬件设备相关的驱动程序，在硬件连接到计算机的时候，windows 会检测到并检测是否安转过驱动程序，如果已安装 就 直接执行 DriverEntry 函数，如果没有安装则提示安装驱动。

5.1.1 安装驱动 WDM1.SYS

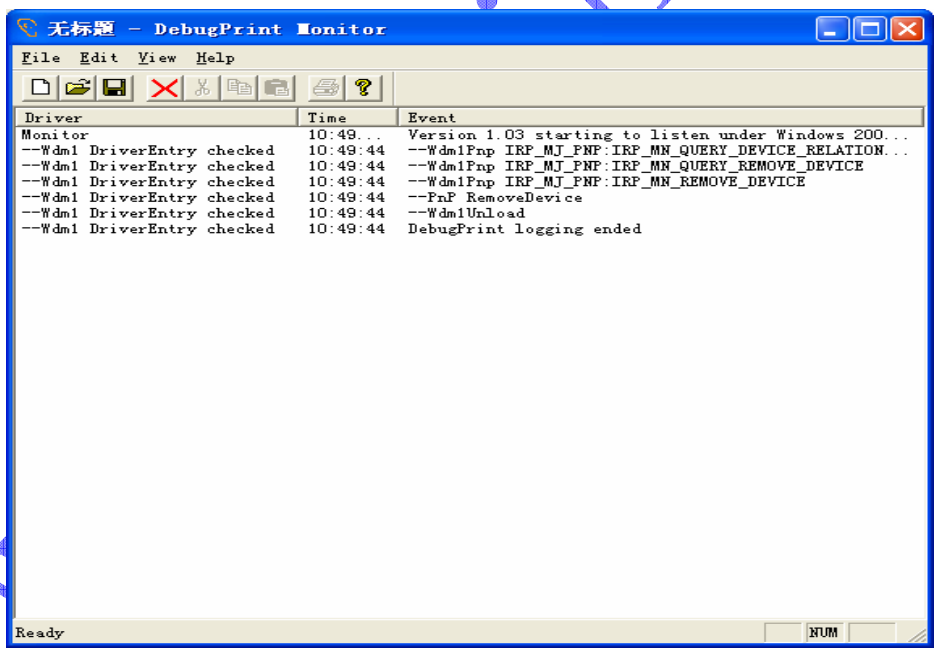
安装 WDM1.SYS 驱动程序与 安装 DebugPrint Driver 的方法一样，参照第四章

5.2 测试 DebugPrintMonitor

安装完成 WDM1.SYS 且每出现黄色警告或红 X 后。
打开 ..\WDMBook\DebugPrint\exe\Release\DebugPrintMonitor.exe



选择停止 WDM1 后，DebugPrint Monitor 得到了 Event，说明 DebugPint Monitor 安装成功了，以后我们就可以用它来理解驱动和得到驱动运行的状态

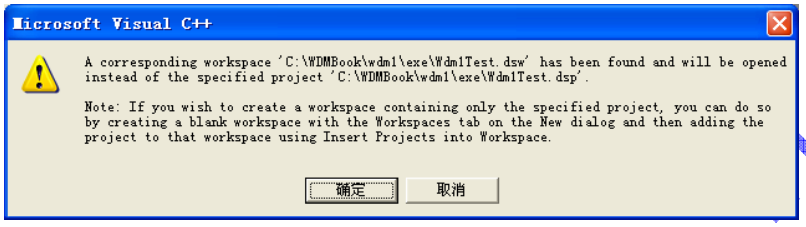


第六章 执行应用程序

wdm1 的应用程序在 wdm1\exe\目录下,根据不同的 DDK 安装目录,我们应该要修改

6.1 打开 Wdm1Test.dsp

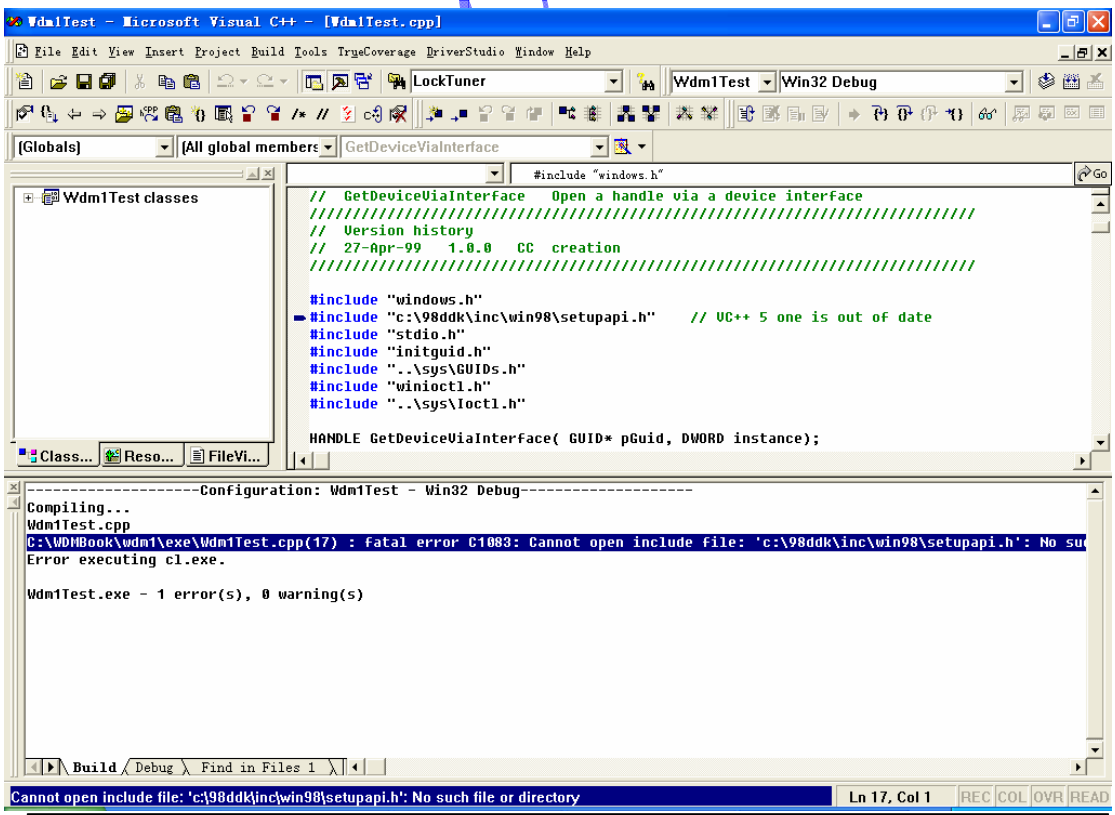
VC6 会提示升级成新格式



我们点击 “是”

6.2 编译 Wdm1Test 工程

按下 F7 进行编译



有错误

```
--------------------Configuration: Wdm1Test - Win32 Debug--------------------
Compiling...
Wdm1Test.cpp
H:\driverDev\wdm1\exe\Wdm1Test.cpp(17) : fatal error C1083: Cannot open include
file: 'c:\98ddk\inc\win98\setupapi.h': No such file or directory
Error executing cl.exe.

Wdm1Test.exe - 1 error(s), 0 warning(s)
```

错误提示为:

Cannot open include file: 'c:\98ddk\inc\win98\setupapi.h': No such file or directory

意思是 ： 在编译 Wdm1Test。CPP 的时候没有找到 setupapi.h 文件

6.3 修改 Wdm1Test .CPP 文件的 setupapi.h 的路径

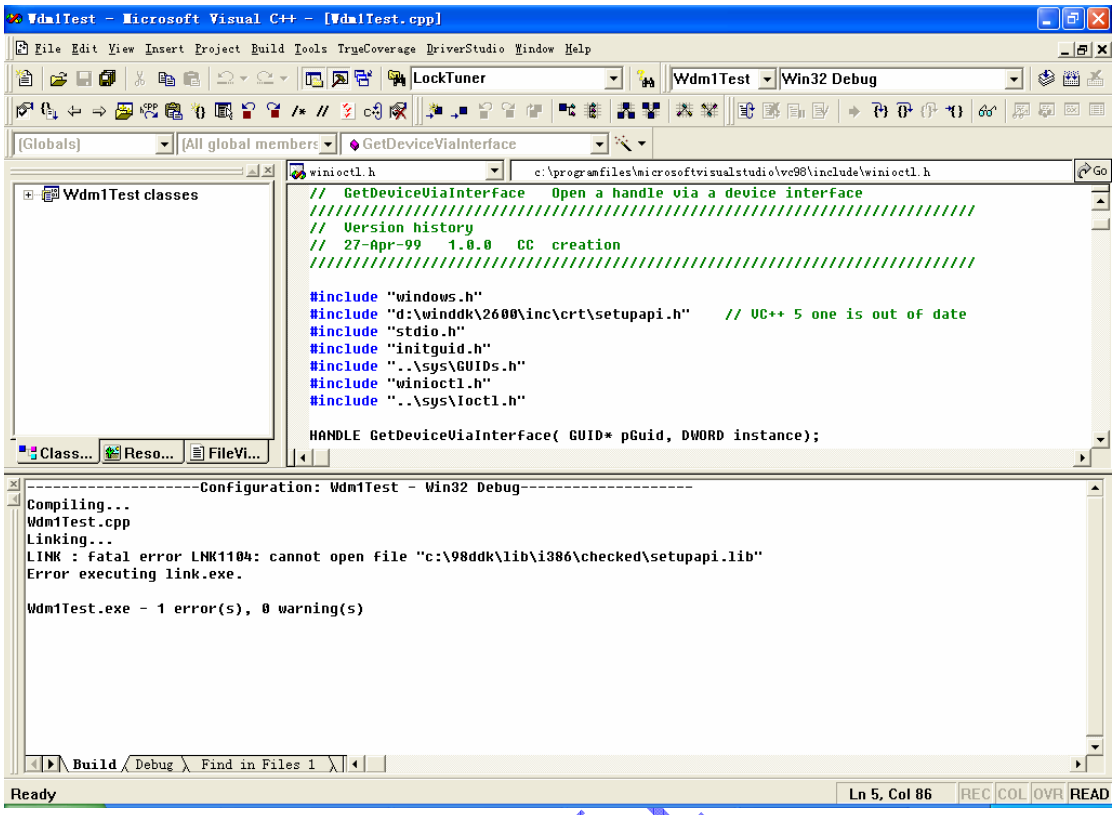
上面的编译错误指出了 setupapi.h 路径错误,这是 DDK 的文件,我们在 DDK 的目录(D:\WINDDK\2600)下搜索 setupapi.h 并把目录设置好

我的 DDK 目录是 D：\winddk\2600

```
#include "d:\winddk\2600\inc\crt\setupapi.h"
```

其实如果这个文件我们如果指向了在 VC 目录下的 setupapi.h 就不会有此问题了

修改好后保存，然后 F7 编译



又有错误？？？

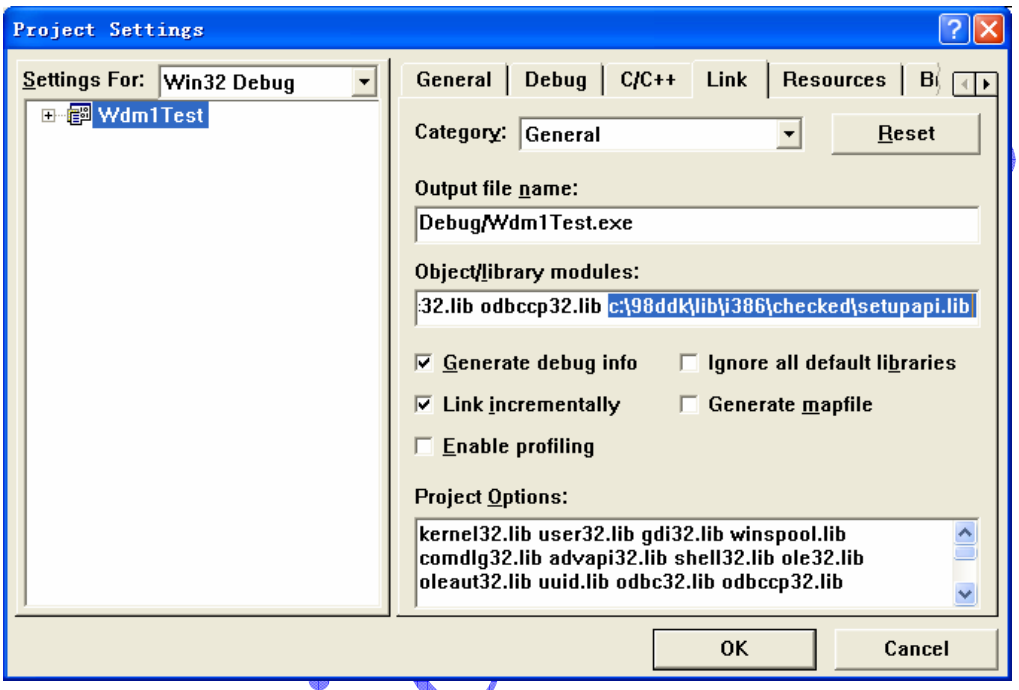
```
--------------------Configuration: Wdm1Test - Win32 Debug--------------------
Compiling...
Wdm1Test.cpp
Linking...
LINK : fatal error LNK1104: cannot open file "c:\98ddk\lib\i386\checked\setupapi.lib"
Error executing link.exe.

Wdm1Test.exe - 1 error(s), 0 warning(s)
```

LINK : fatal error LNK1104: cannot open file c:\98ddk\lib\i386\checked\setupapi.lib"

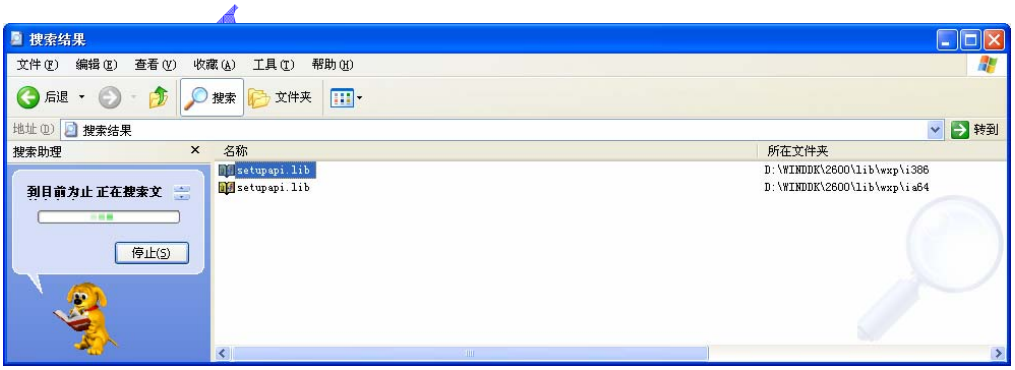
这是说找不到 setupapi.lib

6.3 重新指定 Wdm1Test 工程的 setupapi.lib 的路径

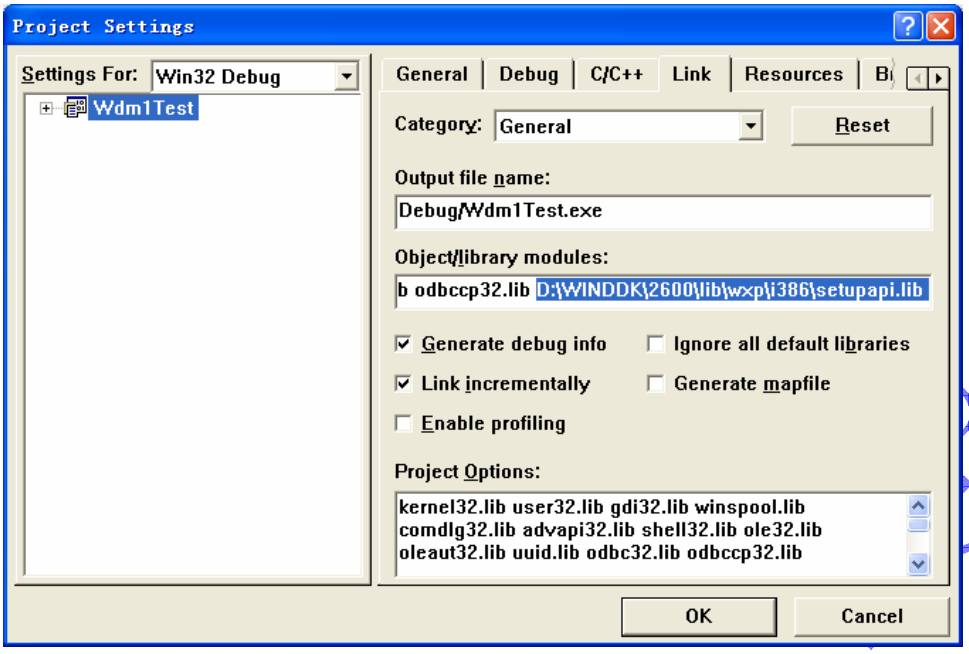
VC6 中 project -> settings -> Link 页面的 Object/library modules 中 setupapi.lib 的
路径确实是：“c:\98ddk\lib\i386\checked\setupapi.lib”



我们在 DDK 目录中搜索它并把 I386 对应的路径拷贝后覆盖 上图蓝色的部分即可



修改后的效果如下：



6.4 类型 DWORD_PTR 和 ULONG_PTR 没定义的错误

```
d:\winddk\2600\inc\crt\setupapi.h(688) : error C2146: syntax error : missing ';' before identifier 'Reserved'
d:\winddk\2600\inc\crt\setupapi.h(688) : error C2501: 'ULONG_PTR' : missing storage-class or type specifiers
d:\winddk\2600\inc\crt\setupapi.h(1482) : error C2146: syntax error : missing ';' before identifier 'PrivateData'
d:\winddk\2600\inc\crt\setupapi.h(1492) : error C2501: 'DWORD_PTR' : missing storage-class or type specifiers
```

找到 typedef PVOID HDEVINFO;
并在他的下方添加， 如下所示：

```
//
// Define type for reference to device information set
//
typedef PVOID HDEVINFO;
// added by o4icwin
typedef unsigned long *ULONG_PTR ;
typedef unsigned long  *DWORD_PTR ;
```

6.5 最后编译，F7

我们现在再编译一下：

```
--------------------Configuration: Wdm1Test - Win32 Debug--------------------
Compiling...
Wdm1Test.cpp
Linking...

Wdm1Test.exe - 0 error(s), 0 warning(s)
```

说明环境已经配置好了，OK

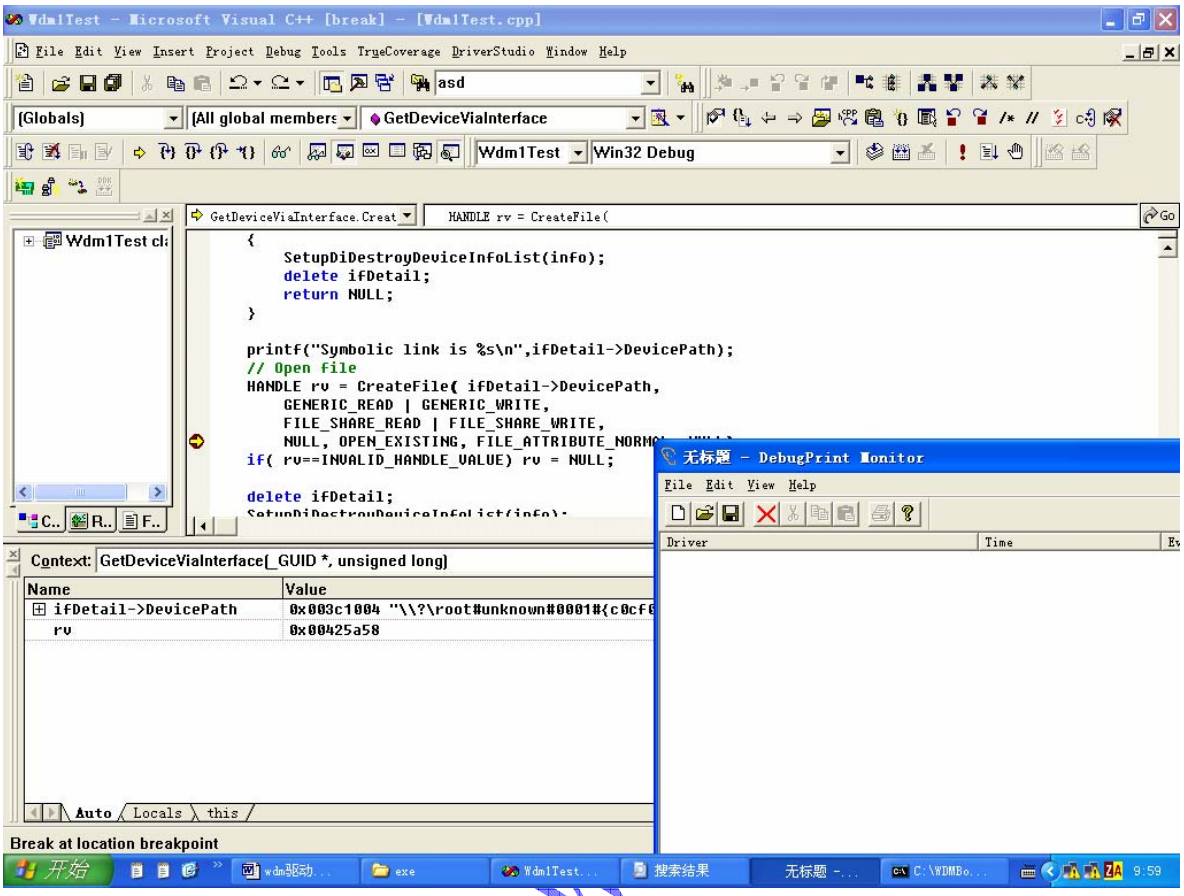
6.4 调试 WdmTest 工程

我们打开 DebugPrintfMonitor 驱动信息监视工具，本工具的配置见 第四章

6.4.1 设置断点

在 VC 搜索　HANDLE rv = CreateFile(ifDetail->DevicePath
把鼠标这里设置断点

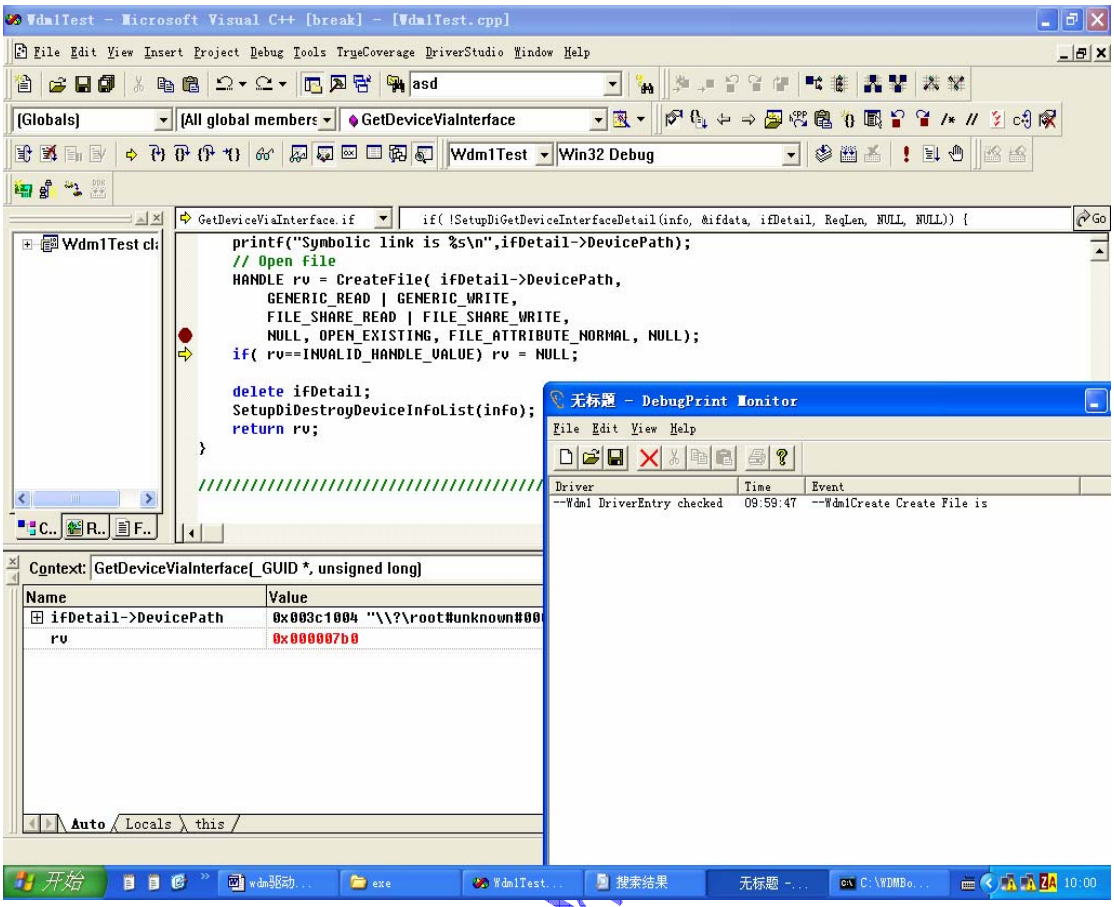
按下 F5 开始进入调试模式，程序会在 HANDLE rv = CreateFile(ifDetail->DevicePath 这个语句停下。



此时 DebugPrintMonitor 中没有信息（我删掉了其他的信息）

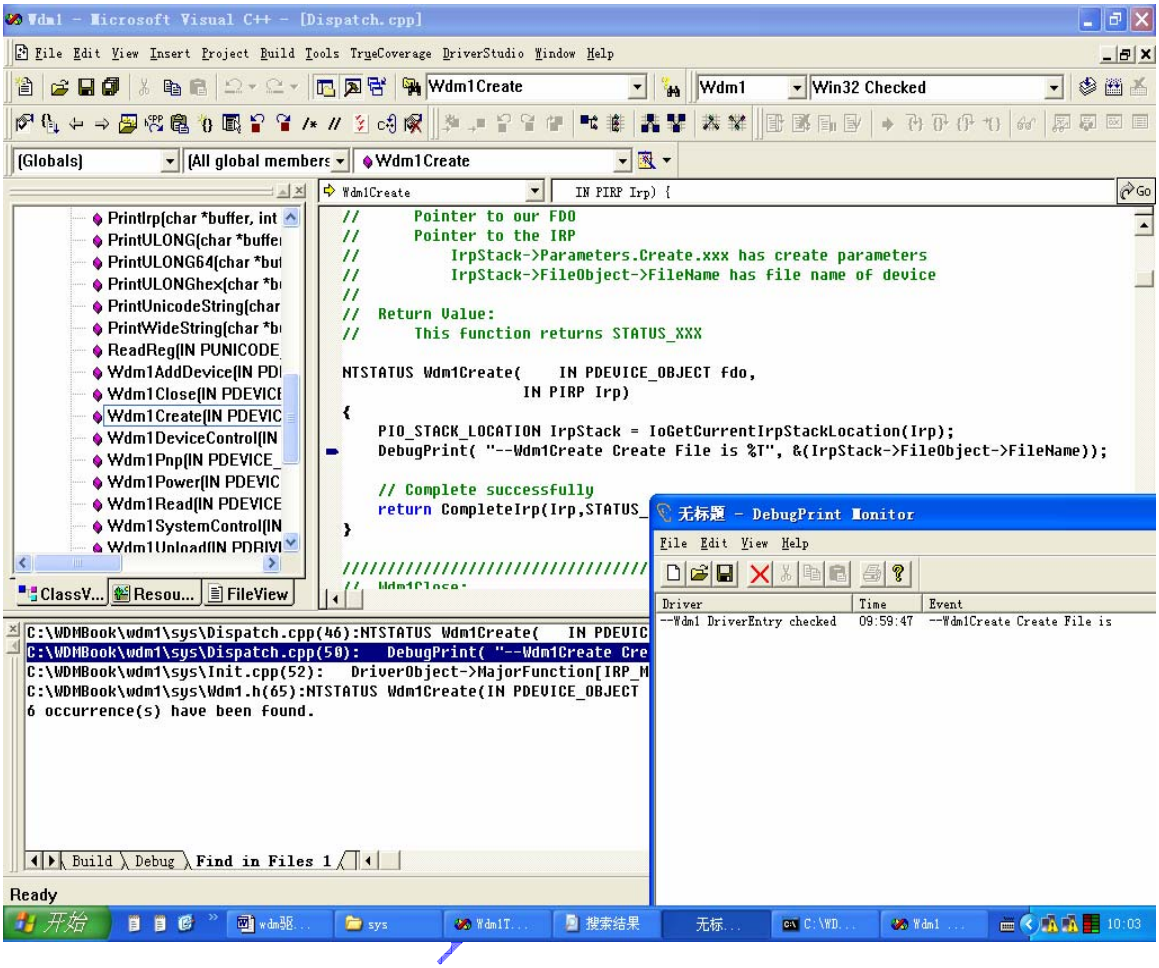
6.4.2 单步执行

我们在 VC 中按下 F10 后情况如图



注意：DebugPrintMonitor 中有了信息，其 EVENT 指出了我们的驱动程序执行了什么消息的处理（回调函数出历）

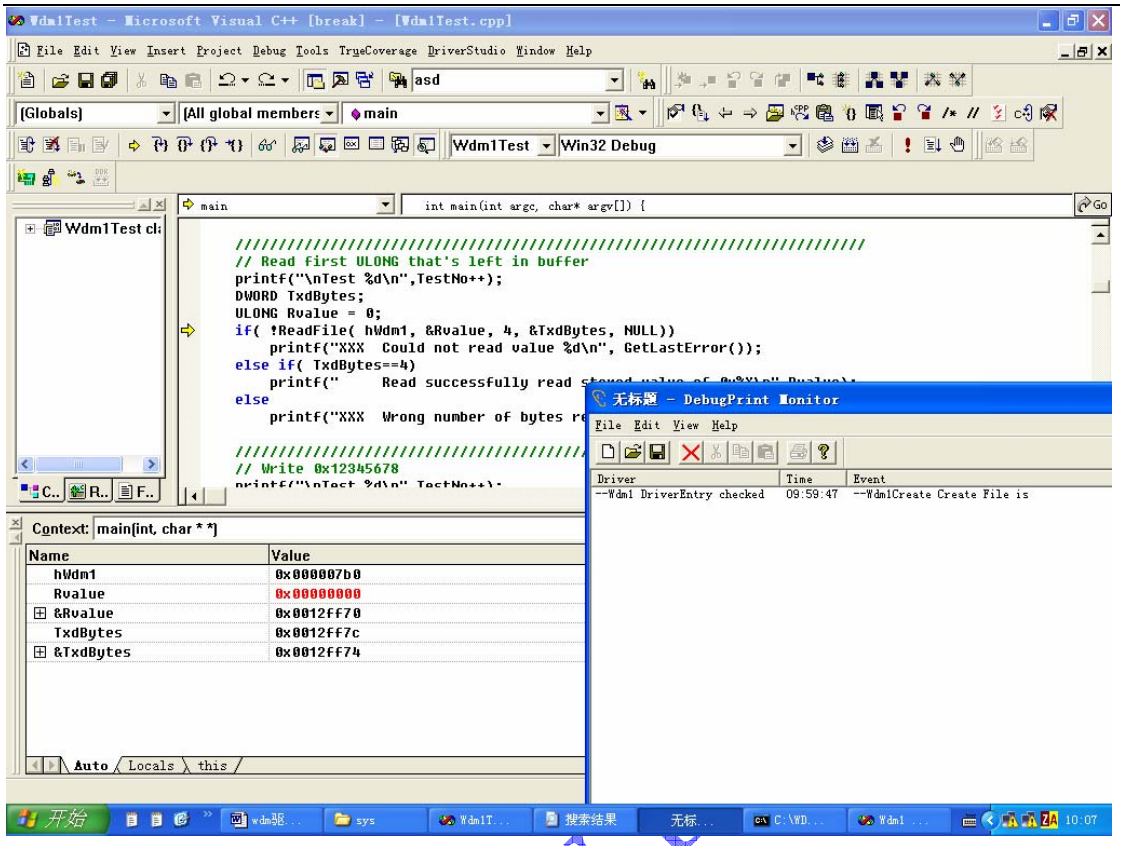
6.4.3 SYS 目录下驱动程序代码对照：



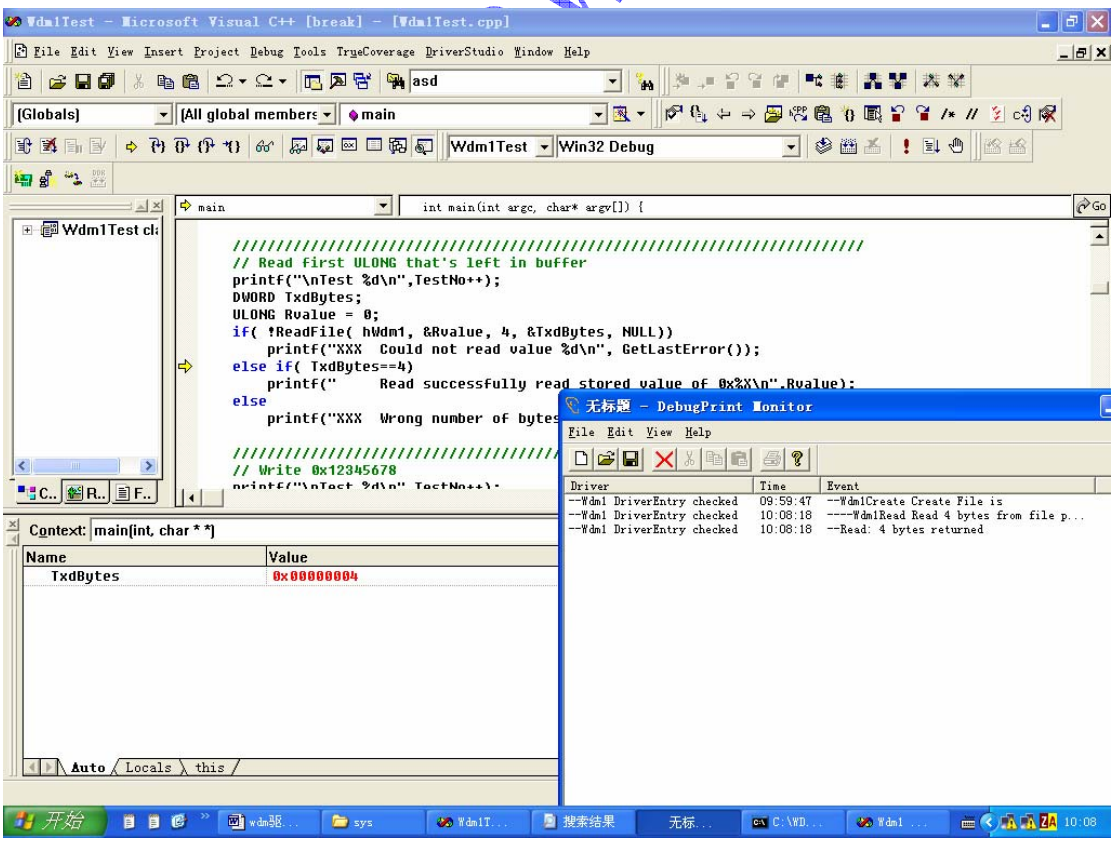
说明我们的 VC 环境配置已经 成功了

6.4.4 EXE 中继续往下执行 ReadFile/WriteFile

6.4.4.1 执行 ReadFile 的情况

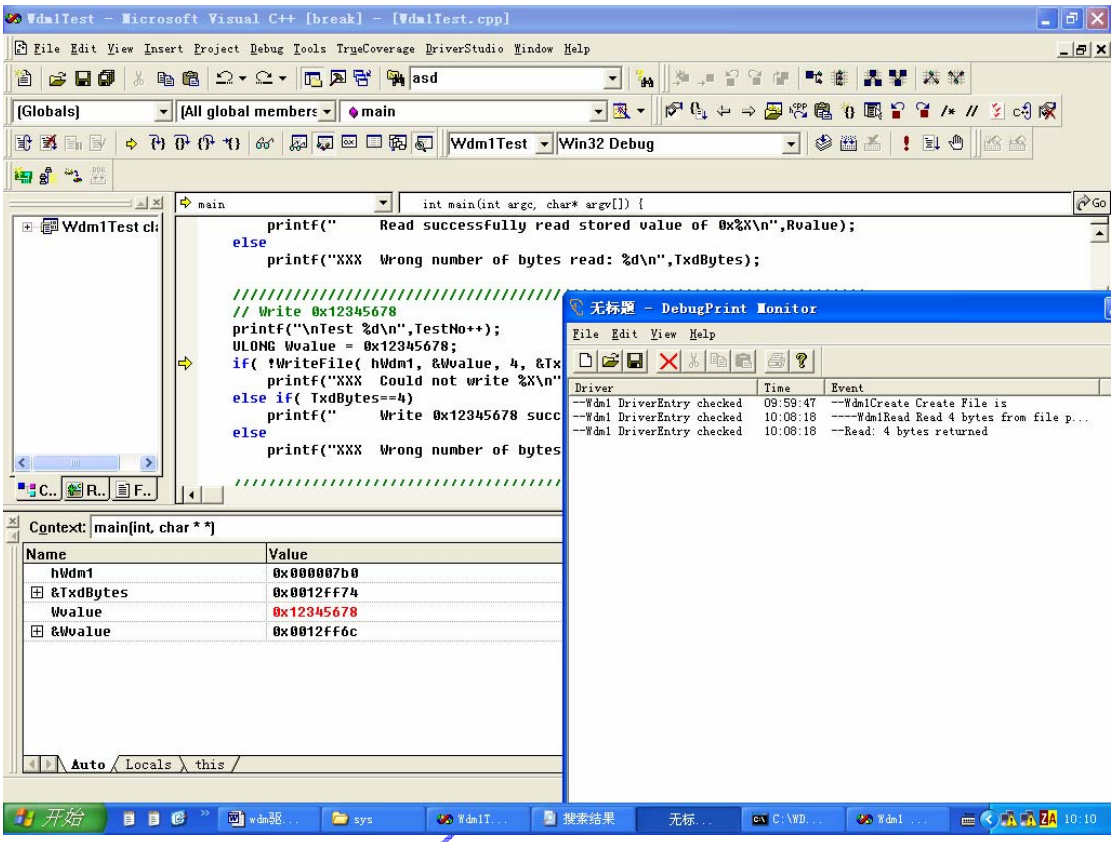


执行 ReadFile 前

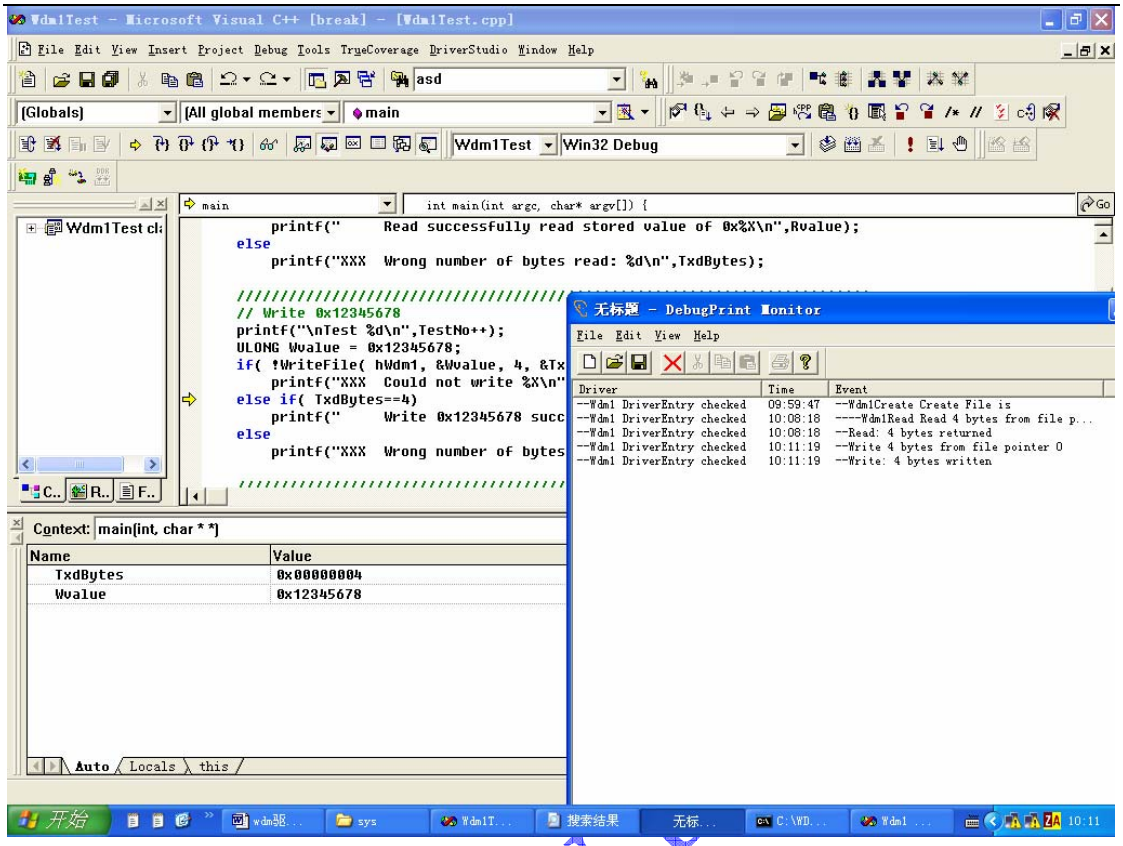


执行 ReadFile 后

6.4.4.2 执行 WriteFile 的情况



执行 WriteFile 前



执行 WriteFile 后

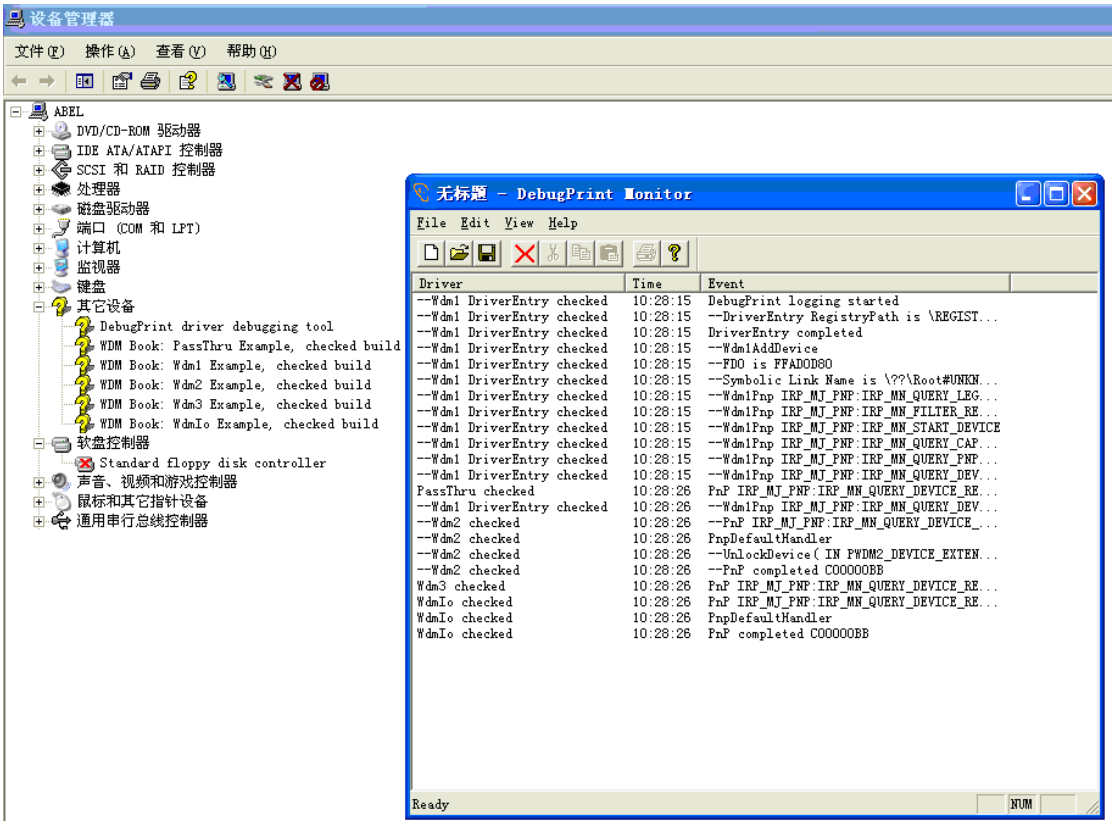
6.4.5 其他的请自己执行

其他的请自己往下执行并观察

第七章 启用 wdm1 驱动程序

我们希望知道启用 wdm1 的过程中，wdm1 驱动程序和操作系统做了些什么

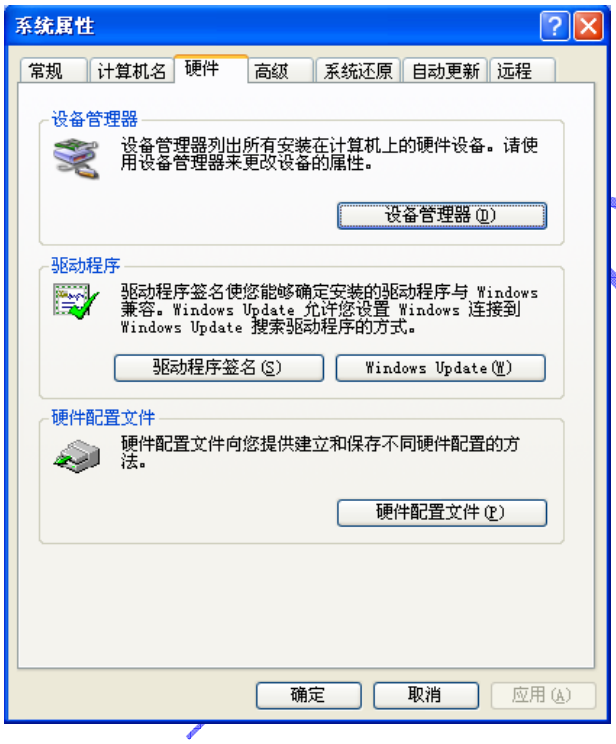
只看 wdm1 标志的啊



第八章 停用 wdm1 驱动程序

我们希望知道停用 wdm1 的过程中，wdm1 驱动程序和操作系统做了些什么

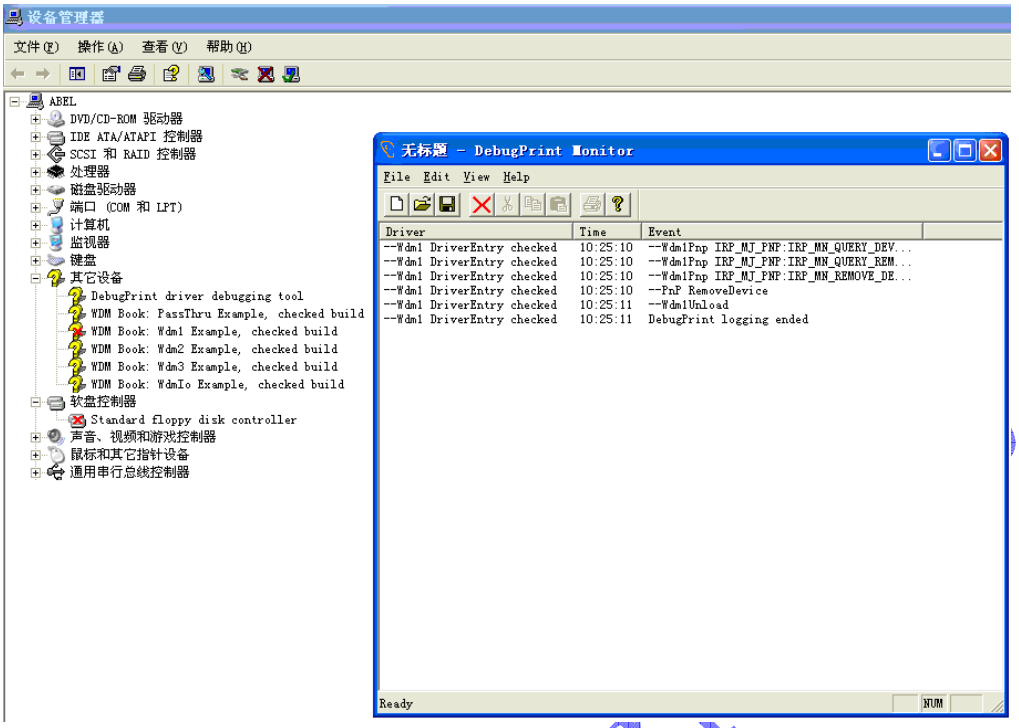
8.1 点击“我的电脑” -〉“属性” -〉“硬件”



8.2 点击“设备管理器”并展开其他设备

选中 wdm1 对应的项 ， 点击 上面工具栏的“ 停止”即可看到 wdm1 的停止消息的处理过程

只看 wdm1 标志的



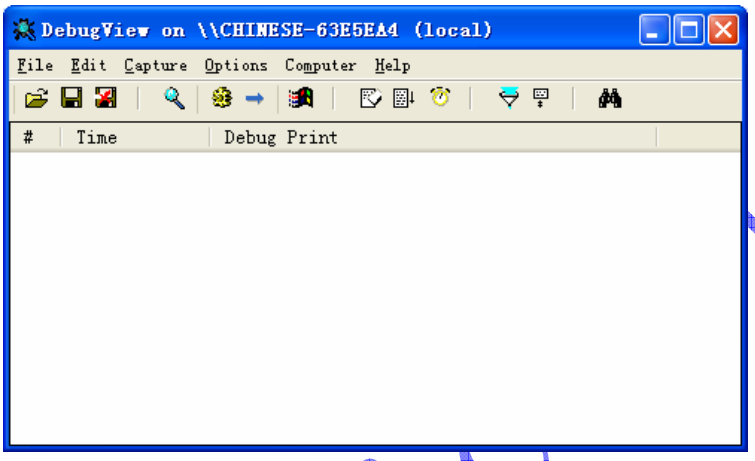
第九章 还有更好的 DebugView.exe

DebugView.exe 驱动打印消息提取工具，功能非常的强大，能完全实现 第四章介绍的 DebugPrintMonitor 的功能.本工具不用安装驱动，一个 EXE 就能实现功能，使用方便异常。

9.1 得到 DebugView.exe

如果您需要，可以EMIL到 icwin@icwin.net 给我们，我们可以给你发送

界面如下：



如下是他的帮助文件主页的内容：

Sysinternals DebugView
Copyright © 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com
DebugView is an application that lets you monitor debug output on your local system, or any computer on the network that you can reach via TCP/IP. It is capable of displaying both kernel-mode and Win32 debug output generated by standard debug print APIs, so you don't need a debugger to catch the debug output your applications or device drivers generate, and you don't need to modify your applications or drivers to use non-Windows debug functions in order to view its debug output.
License
You may not redistribute *DebugView* in any form without the express written permission of Mark Russinovich. If you wish to redistribute *DebugView*, please contact licensing@sysinternals.com.

9.2 原理

驱动开发环境 DDk 中，我们会遇到 DBG，DbgPrint()，为我们提供了非常方便的软件级调试功能

9.2.1 DBG

DBG 是 WDM.H 中定义的宏，在编译时由环境选择，Checked 相当于 VC 的 Debug，Free 相当于 VC 的 Release，

DBG 在 Checked 版本时为假 ， 在 Free 版本时为真

9.2.2 DbgPrint（）

DbgPrint（）函数是驱动程序中常用到的函数，可以定义适用的宏，我们只要知道 DbgPrint 函数的使用就可以了，DbgPrint 函数被声明在 wdm.h 文件中，支持格式化输出，使用方法与 C 语言的 printf（）函数基本一样。

9.2.3 如何使用 DbgPrint（）

DbgPrint（）也不用像 wdm1 的*print*一样需要初始化，我们可以象在 C 下使用 Printf（）函数一样，随时随意的使用。

9.2.4 修改 wdm1 工程的例子

要知道如何才可以让 wdm1.sys 打印的消息能在 DebugView 中显示出来：

我们需要用 DbgPrint 替换所有 wdm1 驱动工程的所有***DebugPrin***方面的函数（当然你只替换一两个也是可以的），那两个文件 DebugPrint.h 和 DebugPrint.c 就不要了，删除吧（多余了）

9.3 如何使用 DebugView

DebugView.exe 不需要安装，直接双击 就 可以使用（在驱动安装前后都可以）

	执行监控，有红 X 时停止监控
	为清除消息

其他的就不说了，自己看吧。

第十章 USB 驱动程序的设计详细

10.1 工作忙待续：请注意关注

第十一章 PCI 驱动设计详细

11.1 工作忙待续：请注意关注