Windows Device Driver Development

1. Setup Driver Development Environment

Step 1: Install following packages on the development computer

1. Platform Software Development Kit (SDK),
2. Micorsoft Visual C (must use the version specified in DDK, newer version might not be working),
3. Device Driver Kits (DDK).

Step 2: Test driver development system

To compile a driver, one uses "build.exe" program from DDK. BUILD reads inputs from files "*dirs*" or "*sources*" and generates outputs "build.log". Suppose DDK is installed in c:\ntddk directory. Open a command window and type

```
c:\NTDDK\bin\setenv.bat c:\NTDDK free
```

This setup an environment to build drivers of free version (no debug). In this command window, type the following commands to build a sample driver,

```
cd c:\NTDDK\src\general\portio
build
```

If everything goes well, "genport.sys" should appear in "C:\NTDDK\src\general\portio\sys\objfre\i386" folder and "gpdread.exe" and "gpdwrite.exe" are also generated.

2. Setup Driver Debug Environment

Install *Checked Build* (Debug Build) Windows on a driver test system. Add a line in [operation systems] section of "boot.ini" file. For example

```
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Debug Windows 2000" /debug /debugport=com1
/baudrate=192000
```

On the development system, create a dedicated debugging directory "c:\Symbols" and copy OS symbol files from Windows CD in "\support\debug\i386\Symbols", especially "hal.dbg" and "ntoskrnl.dbg" into this directory. Also, put the newly created driver (e.g., genport.sys) in this directory because it symbol information for the driver to be tested.

On the development system, start "WinDbg.exe". From View->Option, select Kernel Debugger tab to enable kernel debugger, set communication port and baud rate. Note, the baud rate must match that specified in the test machine. Select Symbols tab and set the symbol search path to "c:\Symbols".

Connect the development machine and test machine with a null modem cable. Copy the newly created driver (e.g., genport.sys) to the test system (normally in %SystemRoot%\System32\Drivers). Create necessary registry entries for this driver (under \HKLM\System\CurrentControlSet\Servics\driver_name). It might be easier to install/uninstall a device driver using a setup program.

DbgPrint() is a macro in Kernel mode equivalent of C's printf(). Often encloded in #if DBG and #endif conditionals. ASSERT() macro only works in checked build.

Start value in registry: (1) 0x00 -- Boot, (2) 0x01 – System, (3) 0x02 – Automatic, (4) 0x03 – Manual, (5) 0x04 – Disabled.

3. Sample Driver

---------------FileName: MyDriver.c ----------------

```
#include <ntddk.h>
//====================================================================
//  Author: Jialong He
//  Date:   Dec. 25, 2002
//====================================================================
#define NativeDriverName L"\\Device\\MyDriver"
#define DosDriverName    L"\\DosDevices\\MyDriver"


//====================================================================
//  I/O Manager calls this routine to unload the device driver
//====================================================================
VOID MyUnload (IN PDRIVER_OBJECT pDriverObject) {

      UNICODE_STRING  DeviceLinkUnicodeString;
      NTSTATUS status;

      //----------------------------------------------------------------
      // "IoDeleteSymbolicLink" removes a symbolic link
      //----------------------------------------------------------------
      RtlInitUnicodeString (&DeviceLinkUnicodeString, DosDriverName);
      status = IoDeleteSymbolicLink (&DeviceLinkUnicodeString);

      //----------------------------------------------------------------
      // "IoDeleteDevice" removes a device object
      //----------------------------------------------------------------
      if (NT_SUCCESS(status))
            IoDeleteDevice(pDriverObject->DeviceObject);

}  // End of DriverUnload

//====================================================================
// Handles "CreateFile" Win32 call from user program
//====================================================================
NTSTATUS MyCreate (IN PDEVICE_OBJECT pDriverObject, IN PIRP pIrp) {

      pIrp->IoStatus.Status = STATUS_SUCCESS;
      pIrp->IoStatus.Information = 0;     // no bytes xfered
      IoCompleteRequest(pIrp, IO_NO_INCREMENT);
      return STATUS_SUCCESS;

} //End of MyCreate

//====================================================================
// Handles "CloseHandle" Win32 call from user program
//====================================================================
NTSTATUS MyClose (IN PDEVICE_OBJECT pDriverObject, IN PIRP pIrp) {

      pIrp->IoStatus.Status = STATUS_SUCCESS;
      pIrp->IoStatus.Information = 0;     // no bytes xfered
      IoCompleteRequest(pIrp, IO_NO_INCREMENT);
      return STATUS_SUCCESS;

} //End of MyClose

//====================================================================
// Handles "ReadFile" Win32 call from user program
//====================================================================
NTSTATUS MyRead (IN PDEVICE_OBJECT  pDriverObject, IN PIRP pIrp) {

      pIrp->IoStatus.Status = STATUS_SUCCESS;
      pIrp->IoStatus.Information = 0;           //no bytes xfered
      IoCompleteRequest(pIrp, IO_NO_INCREMENT );
      return STATUS_SUCCESS;

} // End of MyRead

//==================================================================
// Handles "WriteFile" Win32 call from user program
//==================================================================
NTSTATUS MyWrite (IN PDEVICE_OBJECT  pDriverObject, IN PIRP pIrp) {

      pIrp->IoStatus.Status = STATUS_SUCCESS;
      pIrp->IoStatus.Information = 0;           //no bytes xfered
      IoCompleteRequest(pIrp, IO_NO_INCREMENT );
      return STATUS_SUCCESS;

} //End of MyWrite

//==================================================================
// Handles "DeviceIoControl" Win32 call from user program
//==================================================================
NTSTATUS MyDeviceControl (IN PDEVICE_OBJECT  pDriverObject, IN PIRP pIrp) {

      pIrp->IoStatus.Status = STATUS_SUCCESS;
      pIrp->IoStatus.Information = 0;           //no bytes xfered
      IoCompleteRequest(pIrp, IO_NO_INCREMENT );
      return STATUS_SUCCESS;

} //End of MyDeviceControl

//==================================================================
//  Each driver must have DriverEntry routine. The I/O Manager
//  calls the DriverEntry routine when it loads the driver.
//
//    pDriverObject - Point to a driver object, passed from I/O Manager
//    pRegistryPath - UNICODE_STRING pointer to
//          \HKLM\System\CurrentControlSet\Services\DriverName
//
//    Return value:
//          STATUS_SUCCESS or an appropriate error status.
//
//==================================================================
NTSTATUS DriverEntry (IN PDRIVER_OBJECT pDriverObject,
                      IN PUNICODE_STRING pRegistryPath) {

      PDEVICE_OBJECT pDeviceObject = NULL;
      UNICODE_STRING  DeviceNameUnicodeString;
      UNICODE_STRING  DeviceLinkUnicodeString;
      NTSTATUS status;

      //------------------------------------------------------------
      //  "RtlInitUnicodeString" function initializes a counted
      //  Unicode string from a zero-terminated Unicode string.
      //------------------------------------------------------------
      RtlInitUnicodeString (&DeviceNameUnicodeString, NativeDriverName);
      RtlInitUnicodeString (&DeviceLinkUnicodeString, DosDriverName);

      //------------------------------------------------------------
      // "IoCreateDevice" allocates memory for and initializes
      // a device object for use by a driver.
      //------------------------------------------------------------
      status = IoCreateDevice(pDriverObject, 0,
                              &DeviceNameUnicodeString,
                              FILE_DEVICE_UNKNOWN,
                              0, FALSE,
                              &pDeviceObject);
      if (!NT_SUCCESS(status)) return status;

      //--------------------------------------------------------------
      // "IoCreateSymbolicLink" sets up a symbolic link between
      // an NT device object name and a user-visible name for the device.
      // user program can only access devices in "\??" object directory
      //--------------------------------------------------------------
      status = IoCreateSymbolicLink (&DeviceLinkUnicodeString, &DeviceNameUnicodeString);

      if (!NT_SUCCESS(status)) {
            IoDeleteDevice(pDeviceObject);
            return status;
      }

      //--------------------------------------------------------------
      // Fill Dispatch routine entry points
      //--------------------------------------------------------------
      pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]  = MyDeviceControl;
      pDriverObject->MajorFunction[IRP_MJ_CREATE] = MyCreate;
      pDriverObject->MajorFunction[IRP_MJ_CLOSE]  = MyClose;
      pDriverObject->MajorFunction[IRP_MJ_WRITE]  = MyWrite;
      pDriverObject->MajorFunction[IRP_MJ_READ]   = MyRead;
      pDriverObject->DriverUnload                 = MyUnload;

      return STATUS_SUCCESS;

}  // End of DriverEntry
```

-------- FileName: makefile -----------

```
#
# DO NOT EDIT THIS FILE!!!  Edit .\sources. if you want to add a new source
# file to this component.  This file merely indirects to the real make file
# that is shared by all the driver components of the Windows NT DDK
#

!INCLUDE $(NTMAKEENV)\makefile.def
```

-------- FileName: sources ---------------

```
TARGETNAME=MyDriver
TARGETTYPE=DRIVER

TARGETPATH=.
INCLUDES= $(BASEDIR)\inc;.

SOURCES=.\MyDriver.c
```

General Purpose Commands

Macros in SOURCE file

TARGETNAME	(*Required.*) Specifies the name of the library being built, excluding the file extension.
TARGETPATH	(*Required.*) Specifies a directory name that is the destination of all build products (*.exe*, *.dll*, *.lib* files, and so on). BUILD creates platform-specific subdirectories under this directory. Note that BUILD always creates an *\obj* subdirectory under the directory containing the *sources* file.
TARGETPATHLIB	Specifies a file path and name that is the destination for import libraries created by the build operation. If not specified, import libraries are place in the same subdirectory as other build product files (that is, a subdirectory under TARGETPATH).
TARGETTYPE	(*Required.*) Specifies the type of product being built. This is typically DRIVER or DYNLINK (for DLLs).
TARGETEXT	Specifies the file name extension for DLLs, such as *.cpl*. If this macro is not defined, the default file name extension for DLLs is *.dll*.
TARGETLIBS	(Required.) Specifies the set of import libraries with which your product must be linked. Following is an example: TARGETLIBS=$(SDK_LIB_PATH)\kernel32.lib \ $(SDK_LIB_PATH)\advapi32.lib \ $(SDK_LIB_PATH)\user32.lib \ $(SDK_LIB_PATH)\spoolss.lib
INCLUDES	Contains a list of paths to be searched for header files during compilation. Entries in this list are separated by semicolons, and the paths can be absolute or relative to the directory in which the sources file resides.

	BUILD also searches for header files in a default list of directories. If a sources file contains INCLUDES, the specified paths are searched before the default paths.
SOURCES	(*Required.*) Contains a list of source file names with extensions. These source files will compose the library or DLL being constructed. Entries in this list are separated by spaces or tabs. The files must reside in the directory in which the *sources* file resides. (If you are supplying one or more source files that contain a **main** routine, they must be listed using the UMAPPL or UMTEST macro. They are not listed using SOURCES.)
UMTYPE	**windows** – Win32 user mode, **NT** – kernel-mode, **console** – Win32 console application
UMAPPL	Contains a list of named source files containing a **main** function. These file names are specified without extensions and are separated by asterisks. Each file in this list is compiled and linked with the library or DLL generated by the SOURCES line. If you use UMAPPL, BUILD will automatically build executable files. Listing the names of executables in the BUILD command line is unnecessary.
UMTEST	Contains a list of named source files containing a **main** function. These file names are specified without extensions and are separated by asterisks. Each file in this list is compiled and linked with the library or DLL generated by the SOURCES line. If you use UMTEST, you must identify the names of files you want built by listing them in the BUILD command line.
UMAPPLEXT	Specifies the file name extension for executable files, such as *.com* or *.scr*. If this macro is not defined, the default file name extension for executable files is *.exe*.
UMLIBS	Contains a list of path names of libraries to be linked to the files specified by UMTEST or UMAPPL. The library specified by SOURCES should be included here. Entries in this list are separated by spaces or tabs. These path names must be absolute.
NTPROFILEINPUT	Allows you to specify a file listing the order in which the linker should lay out functions in the image. Set this macro as follows: NTPROFILEINPUT=1 If you set this macro, the directory containing sources must also contain a file named TargetName.prf, where TargetName is the name specified by TARGETNAME.
DLLORDER	Allows you to specify a file listing the order in which the linker should lay out functions in the image. The macro must be set to the name of the file containing the order list. You can use this macro instead of NTPROFILEINPUT.
386_WARNING_LEVEL	Specifies the compiler warning level. The default is: 386_WARNING_LEVEL=-W3

Macros in DIRS file

BUILD can be instructed to recursively build an entire source tree using a dirs file. The dirs file resides in a directory that contains subdirectories (for example, at the root of a source code tree or subtree). Each subdirectory can be a source directory or another source tree. A dirs file should exist at the root of each source code subtree, and a sources file should exist in each "leaf" directory (that is, directories containing actual source code).
You can define the following macros in a dirs file:

DIRS	Contains a list of subdirectories to be built by default.
OPTIONAL_DIRS	Contains a list of subdirectories to be built only if they are explicitly specified as BUILD command arguments.

BUILD Environment Variables

BUILD uses a set of environment variables. Most of them are set by the DDK's setenv.bat file when the "free" or "checked" environment is being initialized. Some can be referenced within sources files, while others are meant for internal use by BUILD. To reference an environment variable in a sources file, use the following syntax: $(VariableName)

BASEDIR	The base of the build product's source tree.
BUILD_ALT_DIR	Appends specified characters to the *\obj* subdirectory name. The "free" and "checked" build environments use this variable to create *\objfre* and *\objchk* subdirectories.
BUILD_DEFAULT	A list of default parameters to pass to the BUILD utility.
BUILD_DEFAULT_TARGETS	A list of default target switches (such as "-386 -MIPS").
BUILD_MAKE_PROGRAM	The name of the make utility used by BUILD. This must be *nmake.exe*.
CRT_INC_PATH	Path to a directory containing Windows 2000 header files.
CRT_LIB_PATH	Path to a directory containing Microsoft-supplied C import libraries.
DDK_INC_PATH	Path to a directory containing DDK-specific, Microsoft-supplied header files.
DDK_LIB_PATH	Path to a directory containing DDK-specific, Microsoft-supplied C import libraries.
DDK_LIB_DEST	Path to a directory to receive a DDK-specific import library that is a build product.
OAK_INC_PATH	Path to a directory containing Microsoft-supplied header files.
SDK_LIB_DEST	Path to a directory to receive an import library that is a build product.
SDK_LIB_PATH	Path to a directory containing Microsoft-supplied C import libraries.
WDM_INC_PATH	Path to a directory containing Microsoft-supplied, WDM-specific header files.
C_DEFINES	Switches passed to both the C compiler and the MIDL compiler.
NTDEBUG	In the "checked" environment this is set to **ntsd**, which causes BUILD to set the C compiler's **/Zi** option to create symbolic debugging information.
BUILD_OPTIONS	List of optional subdirectories, identified by OPTIONAL_DIRS in a *dirs* file, that should be scanned during a build operation. For more information, see Macros for *dirs* Files.

DEBUG Routines (User Mode)

VOID OutputDebugString(LPCTSTR lpOutputString);	sends a null-terminated string to the debugger of the calling process. When debugging a user-mode driver, **OutputDebugString** displays the string in the **WinDbg** Command window. If the debugger is not running, this routine has no effect. **OutputDebugString** does not support the variable arguments of a **printf** formatted string.
VOID DebugBreak(VOID);	A breakpoint routine causes a breakpoint exception to occur in the current process, so that the calling thread can signal the debugger associated with the calling process. If no debugger is running, this usually results in the termination of the calling process.

DEBUG Routines (Kernel Mode)

DbgPrint	DbgPrint displays output in the debugger window. This routine supports the basic printf format parameters. Only kernel-mode drivers can call DbgPrint. KdPrint is similar to DbgPrint when running on a checked build. On a free build, it has no effect.
DbgBreakPoint	also causes a breakpoint, but it additionally sends a 32-bit status code to the debugger.
DbgBreakPointWithStatus	The **DbgBreakPoint** routine works in kernel-mode code, but otherwise is similar to the **DebugBreak** user-mode routine.
KdBreakPoint	identically to **DbgBreakPoint,** but only effect in checked build version.
assert	causes the debugger to display the failed expression and its location in the program;

4. References

1. Open System Resources (http://www.osr.com/)
This is the authors' web site of the book "**Windows NT Device Driver Development**" – an excellent book.

2. Microsoft® Windows® Driver Development Kit (DDK) home page (http://www.microsoft.com/ddk/)

3. Kernel model device driver FAQ (http://www.cmkrnl.com/faq.html)

4. WinDriver – a generic windows driver that can be customized. (http://www.jungo.com/)