D

Debugging Support

The *WINDBG.EXE* debugger can be used for source-level debugging of Windows NT kernel-mode drivers. You can use this debugger in one of two ways:

- To interactively debug a live Windows NT system
- To perform analysis of a previously obtained crash dump (also known as post-mortem analysis)

Unfortunately, *WINDBG* has more than a few reliability problems (unexplained application crashes) and also behaves in an eccentric fashion occasionally. However, more often than not, *WINDBG* will work reasonably well and should be a valuable tool in helping you debug your kernel-mode code.

Interactive Debugging of a Live System

You will need two machines, each running Windows NT, in order to use *WINDBG* to debug a live system.* The machine that executes the driver(s) to be debugged is called the *target machine.* The machine on which *WINDBG* will execute is called the *host machine.* The two machines communicate using a null-modem serial cable which you will need to purchase. One end of this null-modem serial cable must be connected to the serial (COM) ports on each of the two systems (the host and target). If the machines that you use have multiple available COM ports, you can choose any one that you prefer; by default on x86 systems, *WINDBG* expects to use COM2 but this default can easily be overridden using an appropriate option (/DEBUGPORT=PortWame) on the target machine boot command line.t

* It is possible to do things such as debugging an Alpha target using an x86 machine as the host (or vice versa). However, I have tried to avoid this whenever possible.

t You need not use the same COM port on both the host and target machines.

Use the sequence of steps given below to quickly get started with debugging your file system or filter driver:

1. Install either a checked or a free version of the Windows NT operating system on the target machine.

   If you install a checked build, the target machine will execute a lot slower than it will with a free build. However, you may benefit from the assertions and/or any debug print statements that the operating system contains.

TIP If you have sufficient space available on the boot partition of the target system, you could install both the free and the checked builds in separate boot directories and thereby retain the flexibility to boot using either type of operating system build.

---

2. Install a free version of the Windows NT operating system on the host machine.

   You should note that the version of the operating system on both the target and host systems do not need to be the same. However, you will require the particular version of *WINDBG* executing on the host supplied with the SDK associated with the version of the operating system executing on the target.

   Therefore, if you wish to debug a target machine executing Version 3-51 of the Windows NT operating system, you must use the *WINDBG* application that was supplied with the SDK for Version 3.51 of Windows NT. If, however, you wish to debug a target machine executing Version 4.0 of the operating system, you cannot use the same *WINDBG* application that you use to debug Version 3.51; you must use the debugger supplied with the SDK for Version 4.0 of the operating system instead.

3. Install the appropriate SDKs on the host system.

   If you intend to debug multiple operating system releases, install each of the appropriate SDKs on the host system. For example, you could install the SDK for both Version 3.51 of the operating system and Version 4.0 of the operating system on a single host system running Version 4.0. This would give you the capability of debugging drivers on both operating system releases using the single host system.

   At the time of writing this book, I worked with both the 3.51 and 4.0 versions of the Windows NT operating system. Therefore, on most of the host systems that I used for debugging purposes, I created a ... *\MSTOOLS-40* directory to contain the SDK for the 4.0 version of the operating system and a

...*MSTOOLS-351* directory to contain the SDK for the 3.51 version of the operating system.*

4. Copy the target system's debug symbol files to the host machine.

   This symbolic information is required to get meaningful stack traces on the host system. The checked and free versions of the operating system have different symbol files associated with them. These symbol files are supplied with the Windows NT operating system distribution CD. They can typically be found in the *\SUPPORT\DEBUG\<platform-type>\SYMBOLS* directory on the distribution CD. I would advise that you retain the subdirectory layout used on the distribution CD when you copy the symbol files to the host system (use the XCOPY */S source-path target-path* command to achieve this).

   In order to successfully debug both types (retail/free and checked) of operating system binaries, you can create separate CHECKED and FREE subdirectories on the host system that can contain the appropriate symbol files. For example, you can create the following layout:

   — Create the ... *\DEBUG-40\CHECKED\SYMBOLS*† path to contain the debug symbol files for the checked build of the 4.0 release of the Windows NT operating system.

   — Create the ... *\DEBUG-40\FREE\SYMBOLS* path to contain the debug symbol files for the free build of the 4.0 release of the Windows NT operating system.

   — Create the ... *\DEBUG-351\CHECKED\SYMBOLS* path to contain the debug symbol files for the checked build of the 3.51 release of the Windows NT operating system.

   — Create the ... *\DEBUG-351\FREE\SYMBOLS* path to contain the debug symbol files for the free build of the 3.51 release of the Windows NT operating system.

   Debug symbol files change with every new service pack of the operating system. Be aware of this fact and copy over the appropriate new debug symbol files whenever you install a new Windows NT service pack.

5. On the target symbol, modify the [operating systems] section of the *boot.ini* file‡ to enable debugging of the target. Add the following options to

---

* You can similarly install the appropriate versions of the DDK. Since I often use the host system as a compile-link-debug machine, installing the SDK and the DDK is a requirement for me.

† You can create this subtree anywhere you like on the host system; you can specify this search path to *WINDBG* using the Options—> User DLLs—>Symbol Search Path textbox.

‡ This file has the hidden and system attributes set. You will need to remove the hidden and system attributes before you modify the file and then reset them after modifications have been completed.

the appropriate boot command for either or both of the free and checked versions you may have installed:

/DEBUG
: This option enables kernel-mode debugging of the target. The /NODEBUG option disables such debugging (and any of the options given below such as /DEBUGPORT, /BATJDRATE, etc. are ignored).

/DEBUGPORT=*FortName*
: You can use this option to specify an alternate COM port to which you have connected the null-modem serial cable on the target system.*

/BAUDRATE=*BaudRate*
: Specify the highest available baud rate at which both the target and host systems can communicate.

There are other options such as /SOS, /MAXMEM, and /CRASHDEBUG, which are documented in the DDK that you can also specify. These options are not critical, however, to enabling kernel-mode debugging of the target system.

As an example of how to set up boot commands correctly on the target system, study the contents of the *boot.ini* file given below. This is a file that I have set up on one of the target x86 systems I use to debug newly developed kernel-mode drivers:

```
[boot loader]
timeout=30
default=C:\

[Operating Systems]
multi(0)disk(0)rdisk(l)partition(l)\WINNT40="Windows NT Workstation
Version 4.00"
multi(0)disk(0)rdisk(l)partition(l)\WINNT351="Windows NT Workstation
Version 3.51"
multi(0)disk(0)rdisk(2)partition(2)\WINNT40.CKD="Windows NT
Workstation Version
    4.0 (Checked)" /DEBUG /DEBUGPORT=COM1 /BAUDRATE=57600
multi(0)disk(0)rdisk(2)partition(2)\WINNT351.CKD="Windows NT
Workstation
    Version 3.51 (Checked)" /DEBUG /DEBUGPORT=COM1 /BAUDRATE=57600
multi(0)disk(0)rdisk(2)partition(2)\WINNT40.CKD="Windows NT Workstation
    Version 4.00 - Checked"
multi(0)disk(0)rdisk(2)partition(2)\WINNT351.CKD="Windows NT
Workstation
    Version 3.51 - Checked"
C:\="Microsoft Windows 95"
```

6. Configure the *WINDBG* application on the host machine.

---

* To specify an alternate COM port for the host system, you will need to configure the *WINDBG* application settings on the host system as shown in Figure D-l.

You should configure *WINDBG* kernel debugger options to accurately reflect the COM port you are using on the host machine, the baud rate at which you want the host to communicate with the target, and whether you want the initial breakpoint (during target machine startup) to be activated or not. Figure D-l depicts a screen shot of a configuration I've set up.

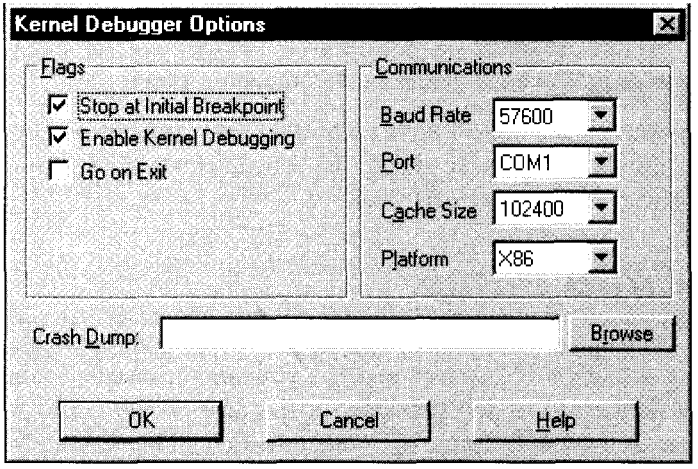

Figure D-l. Configuring the WINDBG kernel debuggeroptions

You may also need to specify the path where you have copied symbols on the host system. Use the User DLLs menu option (from the Options main-menu option list) to specify the path leading to the symbol files.

TIP Once you have configured *WINDBG* correctly, save the program with an appropriate name, to allow you to simply open the same program for subsequent debugging sessions (and avoid having to re-configure each time).

---

7. Copy symbolic information (*your-driver-name.dbg*file or the binary itself) for your driver to the symbol file directory on the host system.
8. Ensure that you have the source files located on the host system for use in source-level debugging sessions.

   If you use your host system as the compile machine as well, then your source files will be easily located by *WINDBG*. If, however, you use some other system to compile and link your binary, ensure that source files are copied onto the host system at the same location where they exist on the compile machine.

TIP If possible, share the source directory tree on the compile machine and access it directly from the host system (using the same drive letter as the one on which they are located on the compile machine).

9. Start *WINDBG* on the host system and open the appropriate program (if you had saved your configuration earlier).
10. Boot the target system using the appropriate boot command option.

The source and target systems will connect with each other and you can proceed with debugging your kernel-mode driver. Read the documentation on using WINDBG that is provided with the Windows NT DDK.

Analyzing a Crash Dump

Occasionally, you may need to determine the root cause of a system crash that may have occurred on a Windows NT system that has your driver installed (and executing), but was not connected to any debugger at the time of the crash. As long as you have access to the crash-dump file, you have a fighting chance of determining the cause of the crash.

NOTE The crash dump is a file that contains the saved system state—including the contents of physical memory—for the machine that experienced a crash.

To analyze a crash dump, configure a Windows NT system exactly as you would otherwise configure a host system for interactive debugging (except that you do not need to physically connect the system via a serial cable to any target machine). Invoke *WINDBG* as follows:

```
WINDBG -y path-to-symbol-flies-directory -z path-to-crash-dump-flie
```

Once you invoke *WINDBG* as shown above, you can pretty much execute the same sequence of steps that you would otherwise execute in debugging a live target system during interactive debugging. The *DUMPCHK* utility shipped with the DDK can be useful in checking the validity of a crash dump file before you use it with *WINDBG.*