MARCH 1995

NOVELL® RESEARCH

Black Screen of Death Explained

BEN HENDRICK
Product Support Engineer
Novell Technical Services

This Application Note takes an in-depth look at the "Black Screen of Death" (BSOD) issue which arises in the Microsoft Windows environment. The symptoms are that the PC locks up and the screen goes black except for a blinking underline cursor in the upper left-hand corner. This AppNote details the results of Novell's research into this problem. It shows that the BSOD symptom is not specific to the NetWare DOS/Windows client software, but can be caused by a misbehaved driver or Windows application that directly interacts with a driver. The AppNote also presents troubleshooting inform-ation to help customers resolve BSOD problems in NetWare/ Windows client configurations.

Related AppNotes

Mar 95 "Support Issues for the NetWare DOS Requester (VLM) 1.2"

Introduction

"Black Screen of Death" (BSOD) is a somewhat melodramatic term often used to describe a particular error condition experienced by users running MS-DOS and Microsoft Windows. The name reflects the characteristic symptoms of the condition--the workstation locks up and the screen goes completely black except for a flashing cursor in the upper left-hand corner of the screen. This can occur at any time while the user is in Windows, when working in a Windows application, when launching a DOS box, or when exiting Windows. BSOD is often interpreted in the media as a Windows/NetWare compatibility problem, when in fact there are situations that can result in the BSOD symptom without the Novell client being loaded.

Novell became involved in the BSOD issue because many of our common customers with Microsoft were encountering this problem. In this AppNote, we will establish three facts:

- The BSOD issue is not specific to the NetWare client software.
- The BSOD state can occur as the result of any misbehaved driver or application that directly interacts with a driver.
- Since Windows offers no information as to what caused the BSOD state to occur, the problem is difficult to troubleshoot.

In addition, we will discuss three categories of problems which occur when the BSOD symptom is specific to the NetWare client running in the Windows environment.

What Causes the Black Screen of Death?

After looking into the BSOD issue, Novell has concluded that this symptom can occur for many different reasons. We have found there is no single cause of the BSOD. The causes can vary--from virtual device drivers running in Ring 0 that "hiccup" and cause the system to crash, to misbehaved applications making incorrect calls to drivers.

Misbehaved Virtual Device Drivers

Various vendors incorporate virtual device drivers (VxDs) into their Windows-based software. As the Windows community moves more and more to VxD-based solutions, we should be prepared to see more of these types of problems--unfortunately with very limited protection, troubleshooting, and debugging capabilities. An article by Ed Bott in *PC Computing* magazine reinforced this point by stating: "Because VxDs attach themselves to Windows, even a tiny bug can crash your entire system. Finding out whether the problem is VxD related can be a tedious chore."

VxD crashes that occur in Ring 0 are the type of errors that occur at the heart of the machine or CPU, bypassing diagnostic utilities such as Dr.Watson. (Dr. Watson is a tool Microsoft provides to detect faults that occur in Ring 3.) In the Intel environment, these errors are often defined as *exceptions*. Typically, what happens is that one piece of code in memory gets corrupted. The next routine or module down the line trusts that the area is not corrupted. When the module calls that corrupted memory, it results into a exception error on the 80x86 processor.

As a crude analogy, the VxD problem might be compared to an assembly line. Each member along the line assumes that all work is accomplished before the item moves down the line. As a result, the assembly line workers are dependent on each other. If any member of the line doesn't complete his or her part correctly, it will eventually bring the whole production line to a halt.

Misbehaved Windows Applications

Windows applications that contain bugs or perform invalid calls can also cause the BSOD symptom. For the purpose of illustration, we wrote a simple "misbehaved" Windows application which causes the BSOD symptom. This application demonstrates that a BSOD can occur without the NetWare client even being loaded. (Refer to Appendix A for the source code to this Sample BSOD program.)

The reason for the BSOD is that the application makes an invalid call. However, if someone unfamiliar with the code were to run this sample application in Windows, the cause of the problem would be a complete mystery. Is it Windows? An application? A device driver?

Note: If Dr.Watson is running, it does not detect that an error has occurred with this particular application.

For the user at large, a very tedious and time consuming investigation process begins at this point. It requires an expert in assembly language and a software debugger to unravel the mystery, isolate the offending module or modules, and determine exactly what caused the exception to occur.

Difficulties in Troubleshooting

Troubleshooting these types of problems would be easier if Windows provided some helpful information on the screen at the time the problem occurred. The Windows environment could certainly benefit from more robust error handling mechanisms and more debugging capabilities.

By contrast, an example of an operating system that handles these types of exceptions is OS/2. OS/2 takes control of misbehaved applications and offers more information and options when an exception occurs. Note:All versions of OS/2 2.0 and higher, including OS/2 Warp, can catch these types of exceptions because they run Windows in Ring 3.

As an example, when we run our Sample BSOD application in the OS/2 environment, we find that it behaves differently. When the exception occurs, we see the following information on the screen:

```
WINOS2.COM
==========================================================================
SYS3176: A program in this session encountered a problem and cannot continue.
==========================================================================
End program/command/operation
Display register information
Redisplay error message
```

If we select **Redisplay error message**, we learn the following:

```
WINOS2.COM
==========================================================================
EXPLANATION: An illegal instruction exception was generated when an attempt was
```

```
made to execute an instruction whose operation was defined for the host machine
architecture. On the Intel 80386** processor, this corresponds to the invalid
opcode fault (#6), caused by an invalid instruction set.

ACTION: If you purchased this program, contact the supplier of the program. If
you are the developer of this program, refer to the information in the register.
===============================================================================
End program/command/operation
Display register information
Redisplay error message
```

For beginning debugging purposes, we can get some additional technical information by next selecting **Display register information**:

```
WINOS2.COM
===============================================================================
A program executed an illegal instruction at 000ffc79.
EAX=0000ce10         EBX=00000246 ECX=0000e144 EDX=00000806
ESI=0000026a         EDI=0000fbd4
DS=8da4     DSACC=****     DSLIM=********
ES=c000     ESACC=****     ESLIM=********
FS=0000     FSACC=****     FSLIM=********
GS=0000     GSACC=****     GSLIM=********
CS:EIP=ff83:00000449       CSACC=****     CSLIM=********
SS:ESP=2506:0000400        SSACC=****     SSLIM=********
EBP=000022a0         FLG=00020206
===============================================================================
End program/command/operation
Display register information
Redisplay error message
```

At this point, we can choose **End program/command/operation** and continue to work in other programs without further glitches.

The progression from Windows 3.0 to 3.1x saw an increase in the reliability and error handling capabilities of the Windows environment. With the arrival of Windows 95, it is expected that users will see more protection for these types of issues in that environment.

To review what we have covered so far:

1. We established the fact that BSOD state can occur as a result of a misbehaved driver or application.

2. We have seen that, because Windows offers no information as to what caused the BSOD state to occur, troubleshooting the problem is difficult.

3. We have established the fact that the BSOD issue is not specific to the NetWare client software.

Due to the nature of today's NetWare workstation environment-- which is basically a conglomeration of hardware and software from multiple vendors that relies on Novell's DOS/Windows client software--there will be times when users will encounter the BSOD symptom while using the Novell client software. The next portion of this document discusses some of the discoveries Novell has made in trying to solve the BSOD riddles that puzzle our common customers in this environment.

Three Categories of BSOD Problems

In a mixed NetWare/Windows client configuration, Novell has identified three categories or types of problems that can cause a workstation to end up in the BSOD state. These are:

- Windows time-outs that lead to a deadlock state
- Critical exceptions inside Windows
- Time-outs when exiting Windows

Windows Timeouts = Deadlocks

Virtual Timer Device (VTD). There is a common definition of BSOD upon which Microsoft and Novell both agree. In this scenario, a user is working inside of Windows and has the NetWare client software running. The user double-clicks on the MS-DOS icon and the machine displays the BSOD symptom.

In this "classic" case of BSOD, the problem is related to the Windows Virtual Timer Device (VTD) inside the Windows kernel. Microsoft and Novell working together found a problem in the VTD driver which would cause a deadlock with the NetWare shell or redirecter (NETX or VLM) loaded. The deadlock or BSOD state occurs as a result of the shell or redirector waiting for the timer tick to advance. When a NetWare Core Protocol (NCP) request is sent, the shell or redirecter starts its retry time-out countdown.

This is essentially a loop waiting for the timer tick value to advance to a certain point. A deadlock occurs if the tick value never increments and thus leaves the shell or redirecter in an infinite loop. The reason the tick count is never incremented is that, under certain circumstances, the timer interrupt is not allowed to be simulated into V86 mode by the VTD.

There is a single-byte patch to WIN386.EXE which prevents this situation from occurring by allowing the timer interrupts to be simulated. The fix is in the VTD's control procedure. Without the patch, a certain procedure is called while the machine is in the Create_VM state. With the patch, this procedure is called during the VM_Critical_Init state. The patch is applied at offset 441D8 in WIN386.EXE (544789, 03-01-92, 3:10a). The value at that location is 07 (Create_VM). The value should be changed to 08 (VM_Criti-cal_Init).

If you are experiencing this VTD problem, you will need to obtain an updated VxD (VTDA.386) driver. This patch is contained in the WW0863.EXE file available from Microsoft. In addition to the VTDA update from Microsoft, Novell has prepared enhanced LSL.COM, IPXODI.COM and VIPX.386 drivers which work better in combination with this Microsoft patch. These Novell updates can be found in the latest VLMUP*x*.EXE, WINDR*x*.EXE, and NWDLL*x*.EXE files (where *x* = the current revision number). See Appendix B for information on where to find these update files.

Note: This patch is not needed for Windows for Workgroups because the patch is included in the operating system.

LAN IRQ Timeout. VPICD.386 normally virtualizes the LAN card interrupts (IRQs) when Windows starts. However, the default IRQ handling in VPICD is inadequate for networks. When a LAN IRQ occurs, VPICD determines which Virtual Machine (VM) is to handle the IRQ, boosts that VM's priority to TIME_CRITICAL_ BOOST (that is, interrupt boost), and simulates the IRQ into the V86 mode. The globally-loaded LAN driver interrupt service routine (ISR) runs in the context of the now currently-running VM. However, VPICD imposes a timeout on the interrupt. If a LAN driver takes more than 500 milliseconds to process packets from the LAN card (which can and does happen often), VPICD will timeout the IRQ and give the current running VM a priority of 0.

Windows' primary scheduler then allows a time slice to another VM. If that VM attempts to use NETX or VLM to do a network request, a deadlock will occur because the newly running VM has claimed critical section (that is, it has CRITICAL_SECTION_ PRIORITY_ BOOST) and attempts to process network traffic. The new network request will be queued because the LAN driver knows it is in the context of its ISR. However, the LAN driver cannot continue because the first VM has priority 0, which is less than the CRITICAL_SECTION_PRIORITY_BOOST priority of the second VM.

This deadlock is a limitation of the Windows VPICD.386. Novell provided a workaround in VIPX version 1.17. In version 1.17 or later, VIPX takes over the virtualization of the LAN IRQs and has a much longer timeout for the IRQ than VPICD.386 does. See Appendix B for additional information on VIPX.

Transport Retry. SPX, NetBIOS, and NETX each employ classic transport retry mechanisms. The retry mechanism works as follows. Once a packet is sent, a timer event is scheduled to fire for some timeout value in the future. If an ACK (acknowledgement) is received before the timeout occurs, the timer event is canceled. If there is no ACK before the timer event fires, the timer event procedure assumes the packet was lost and resends the packet. After a certain number of resends, the connection is aborted and no further attempt is made to send the packet.

Unfortunately, this classic transport retry algorithm is prone to deadlocks under Windows. Usually, a packet is sent with the Windows Critical Section owned. Windows allows timer ticks (the engine for timer events) to be simulated into other VMs (Virtual Machines) while another VM owns the Windows Critical Section. If a timer tick causes a timer event to expire in the context of a VM which does not own the Windows Critical Section, the timer event runs and sends a retry. Sending a packet will start a Windows Critical Section. This effectively blocks the second VM because the first VM owns the Windows Critical Section. As a result, the timer event in the second VM never completes and is never rescheduled (that is, it never fires again). The first VM waits for an ACK or for the timer event to abort the connection, neither of which will ever occur.

This deadlock is fixed by increasing the `TimerCriticalSection=` setting in the `[386Enh]` section of the system.ini file to a minimum of 1000, or up to a maximum of 10000. This is why Novell requires the PC configuration to include this parameter when the NetWare DOS/Windows client is being used.

Critical Exceptions Inside Windows

This situation typically involves a user who is in Windows either working in an application or sitting idle, and the machine suddenly drops off into a BSOD state. Novell has found that the cause of this problem can range from an ODI driver acting strangely to a misbehaved network application using SPX. Each scenario can be caused by a different vendor's components. Because the NetWare workstation environment is inherently multivendor in nature, solving one specific problem in this category does not necessarily mean that it will fix all BSOD problems.

In the current Windows/DOS architecture, there is no way for a single fix to resolve all BSOD problems. However, there does exist today some potential to enhance the error handling and information provided when exceptions occur. This makes the troubleshooting process much easier.

One idea that enhances the troubleshooting process for this second category in the current DOS/Windows environment is offered by Novell. This idea involves a function in Novell DOS 7's memory manager (EMM386.EXE) that provides information about exceptions that Microsoft's DOS memory manager does not currently provide.

For example, if you duplicate a BSOD situation by using the Sample BSOD application with Novell DOS 7 running underneath Windows 3.1, you receive the error information shown below (note that it is not as advanced as with OS/2, but at least you receive the same general information as OS/2 does):

This technical information on exception errors could be used to associate specific types of problems with certain patches. However, with the current memory manager (emm386.exe) in MS-DOS, this case study offers no help beyond the flashing cursor in the upper left-hand corner of the screen.

For the sake of time, we will not cover all the known issues related to this category. However, we need to address one issue that was published in *InfoWorld*, May 16, 1994, in an article by Brian Livingston. It was titled, "A bit of promising news on Windows' Black Screen of Death."

This article gives the steps to reproduce one specific case of a BSOD problem when using NETX in the Windows environment (the problem doesn't occur with VLMs). A portion of these steps is as follows:

Step 10: Exit Windows.
Step 11: Do NOT reboot your system.
Step 12: Start Windows again.

Steps 10 12 were the key to understanding the problem in this scenario. An entry point in memory used by NETX was not being reinitialized by NETX when Windows was restarted. Because of this, NETWARE.DRV made a call to an invalid area of memory in a later step (step 18). The result was the BSOD symptom.

Novell has addressed the problem described in *InfoWorld* by providing an updated NETX.EXE. This update can be found in NOVFILES in a file called NET33X.EXE.

Timeout When Exiting Windows

Another timeout scenario can occur when users exit Windows. The machine hangs in a BSOD state and never returns to the DOS prompt. In this situation, no crash or exception error is occurring. Research has shown that this problem is due to an application, device driver, or TSR that has not fully finished up and the machine is waiting for the next instruction. Windows does not timeout; therefore, the machine never returns to the DOS prompt.

Note: This issue is included in the NetWare/Windows category because early versions of VIPX had a problem un-virtualizing the LAN IRQs. This problem was fixed in VIPX v1.18.

For the sake of troubleshooting, it would be helpful if Windows preempted the application or driver after a timeout loop and returned control to DOS. In addition, Windows could pass to DOS some information in a log file noting which device driver or application caused the system to hang.

Conclusion

In conclusion, we reiterate that pinpointing the exact cause of BSOD problems is difficult at best. Windows users are often left not knowing what happened or what caused the BSOD symptom to occur. On the road to Windows 95, we hope to see more help or information for those users who currently end up in a BSOD state under Windows.

Additional references and information resources concerning BSOD problems are listed in Appendix B of this AppNote.

Appendix A: Sample BSOD Program

Here is the source code for the sample BSOD-producing Windows application.

```
/*------------------------------------------------------------------------
----
      BSOD.C -- Tests Black Screen of Death without Novell Client Loaded.

-----------------------------------------------------------------------------
-*/
#include <windows.h>
#include <dos.h>
#pragma argsused
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
      LPSTR lpszCmdParam, int nCmdShow) {
      if (MessageBox(NULL,
            (LPCSTR)"Press OK to generate a  Black Screen of Death'.",
            (LPCSTR)"BSOD Test", MB_OKCANCEL) == IDOK)
            geninterrupt(0x30);      // This is a stab in the dark because
                                  //   register contents are unknown.
      return 0;

}
```

Appendix B: Additional References

Updated NetWare DOS/Windows client software and Windows support drivers can be found in novfiles on CompuServe or on Novell's ftp server on the Internet. As of this writing, the current files are the following:

File Name	Size	Date	Version	Location	Owner
VIPX.386	23855	05-23-94	1.19	WINDR2.EXE	Novell
IPXODI.COM	39353	10-31-94	3.01	VLMUP2.EXE	Novell
LSL.COM	18313	10-11-94	2.14	VLMUP2.EXE	Novell
NETX.EXE	78749	11-22-94	3.32PTF	NET33X.EXE	Novell

The following text files are included with the Novell downloads listed above and are recommended for further information:

- `DEADLOCK.TXT`
- `VIPX.DOC`

In addition, the Network Support Encyclopedia Professional (NSEPro) CD-ROM contains a number of documents that are helpful in troubleshooting Black Screen of Death problems. Simply search for references using the keyword "BSOD" to access these documents from the CD.