Debugging Tools for Windows

Bug Check Code Reference

This section contains descriptions of the common bug checks, including the parameters passed to the blue screen.

It also describes how you can diagnose the fault which led to the bug check, and possible ways to deal with the error.

If a specific bug check code does not appear in this reference, use the **!analyze** extension command (in kernel mode) with the following syntax:

```
kd> !analyze -show Code
```

This will display information about the specified bug check code. If your default radix is not 16, you should prefix *Code* with "0x".

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1: APC_INDEX_MISMATCH

The APC_INDEX_MISMATCH bug check has a value of 0x00000001. This indicates that there has been a mismatch in the APC state index.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the system function (system call)
2	The value of the following bit computation: Thread->**ApcStateIndex** << 8 | Previous **ApcStateIndex**
3	The value of Thread->**KernelApcDisable**
4	The value of the previous **KernelApcDisable**

Cause

The most common cause of this bug check is when a file system has a mismatched sequence of **KeEnterCriticalRegion** calls and **KeLeaveCriticalRegion** calls.

Comments

This is a kernel internal error which can occur only on a checked build. This error occurs on exit from a system call.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2: DEVICE_QUEUE_NOT_BUSY

The DEVICE_QUEUE_NOT_BUSY bug check has a value of 0x00000002.

This bug check appears very infrequently.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3: INVALID_AFFINITY_SET

The INVALID_AFFINITY_SET bug check has a value of 0x00000003.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4: INVALID_DATA_ACCESS_TRAP

The INVALID_DATA_ACCESS_TRAP bug check has a value of 0x00000004.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x5: INVALID_PROCESS_ATTACH_ATTEMPT

The INVALID_PROCESS_ATTACH_ATTEMPT bug check has a value of 0x00000005. This generally indicates that the thread was attached to a process in a situation where that is not allowed. For example, this bug check could occur if **KeAttachProcess** was called when the thread was already attached to a process (which is illegal), or if the thread returned from certain function calls in an attached state (which is invalid),

This bug check appears very infrequently.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The pointer to the dispatcher object for the target process, or if the thread is already attached, the pointer to the object for the original process.
2	The pointer to the dispatcher object of the process that the current thread is currently attached to.
3	The value of the thread's APC state index.
4	A non-zero value indicates that a DPC is running on the current processor.

Comment

This bug check can occur if the driver calls the **KeAttachProcess** function and the thread is already attached to another process. It is better to use the **KeStackAttachProcess** function. If the current thread was already attached to another process, the **KeStackAttachProcess** function saves the current APC state before it attaches the current thread to the new process.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6: INVALID_PROCESS_DETACH_ATTEMPT

The INVALID_PROCESS_DETACH_ATTEMPT bug check has a value of 0x00000006.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7: INVALID_SOFTWARE_INTERRUPT

The INVALID_SOFTWARE_INTERRUPT bug check has a value of 0x00000007.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x8: IRQL_NOT_DISPATCH_LEVEL

The IRQL_NOT_DISPATCH_LEVEL bug check has a value of 0x00000008.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9: IRQL_NOT_GREATER_OR_EQUAL

The IRQL_NOT_GREATER_OR_EQUAL bug check has a value of 0x00000009.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA: IRQL_NOT_LESS_OR_EQUAL

The IRQL_NOT_LESS_OR_EQUAL bug check has a value of 0x0000000A. This indicates that Microsoft Windows or a kernel-mode driver accessed paged memory at DISPATCH_LEVEL or above.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write
4	Address which referenced memory

Cause

This bug check is issued if paged memory (or invalid memory) is accessed when the IRQL is too high.

The error that generates this bug check usually occurs after the installation of a faulty device driver, system service, or BIOS.

If you encounter bug check 0xA while upgrading to a later version of Windows, this error might be caused by a device driver, a system service, a virus scanner, or a backup tool that is incompatible with the new version.

Resolving the Problem

If a kernel debugger is available, obtain a stack trace.

To resolve an error caused by a faulty device driver, system service, or BIOS

1. Restart your computer.
2. Press F8 at the character-based menu that displays the operating system choices.
3. Select the **Last Known Good Configuration** option from the Windows **Advanced Options** menu. This option is most effective when only one driver or service is added at a time.

To resolve an error caused by an incompatible device driver, system service, virus scanner, or backup tool

1. Check the System Log in **Event Viewer** for error messages that might identify the device or driver that caused the error.
2. Try disabling memory caching of the BIOS.
3. Run the hardware diagnostics supplied by the system manufacturer, especially the memory scanner. For details on these procedures, see the owner's manual for your computer.
4. Make sure the latest Service Pack is installed.
5. If your system has small computer system interface (SCSI) adapters, contact the adapter manufacturer to obtain updated Windows drivers. Try disabling sync negotiation in the SCSI BIOS, checking the cabling and the SCSI IDs of each device, and confirming proper termination.
6. For integrated device electronics (IDE) devices, define the onboard IDE port as Primary only. Also, check each IDE device for the proper master/subordinate/stand-

alone setting. Try removing all IDE devices except for hard disks.

If the message appears during an installation of Windows, make sure that the computer and all installed peripherals are listed in the Microsoft Windows Marketplace Tested Products List.

Here is a debugging example:

```
kd> .bugcheck       [Lists bug check data.]
Bugcheck code 0000000a
Arguments 00000000 0000001c 00000000 00000000

kd> kb       [Lists the stack trace.]
ChildEBP RetAddr  Args to Child
8013ed5c 801263ba 00000000 00000000 e12ab000 NT!_DbgBreakPoint
8013eecc 801389ee 0000000a 00000000 0000001c NT!_KeBugCheckEx+0x194
8013eecc 00000000 0000000a 00000000 0000001c NT!_KiTrap0E+0x256
8013ed5c 801263ba 00000000 00000000 e12ab000
8013ef64 00000246 fe551aa1 ff690268 00000002 NT!_KeBugCheckEx+0x194

kd> kv       [Lists the trap frames.]
ChildEBP RetAddr  Args to Child
8013ed5c 801263ba 00000000 00000000 e12ab000 NT!_DbgBreakPoint (FPO: [0,0,0])
8013eecc 801389ee 0000000a 00000000 0000001c NT!_KeBugCheckEx+0x194
8013eecc 00000000 0000000a 00000000 0000001c NT!_KiTrap0E+0x256 (FPO: [0,0] TrapFrame @ 8013eee8)
8013ed5c 801263ba 00000000 00000000 e12ab000
8013ef64 00000246 fe551aa1 ff690268 00000002 NT!_KeBugCheckEx+0x194

kd> .trap 8013eee8       [Gets the registers for the trap frame at the time of the fault.]
eax=dec80201 ebx=ffdff420 ecx=8013c71c edx=000003f8 esi=00000000 edi=87038e10
eip=00000000 esp=8013ef5c ebp=8013ef64 iopl=0         nv up ei pl nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010202
ErrCode = 00000000
00000000 ??????????????       [The current instruction pointer is NULL.]

kd> kb       [Gives the stack trace before the fault.]
ChildEBP RetAddr  Args to Child
8013ef68 fe551aa1 ff690268 00000002 fe5620d2 NT!_DbgBreakPoint
8013ef74 fe5620d2 fe5620da ff690268 80404690
NDIS!_EthFilterIndicateReceiveComplete+0x31
8013ef64 00000246 fe551aa1 ff690268 00000002 elnkii!_ElnkiiRcvInterruptDpc+0x1d0
```

Comments

Before upgrading to a new version of Windows, remove all third-party device drivers and system services, and disable any virus scanners. Contact the software manufacturers to obtain updates of these third-party tools.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xB: NO_EXCEPTION_HANDLING_SUPPORT

The NO_EXCEPTION_HANDLING_SUPPORT bug check has a value of 0x0000000B.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC: MAXIMUM_WAIT_OBJECTS_EXCEEDED

The MAXIMUM_WAIT_OBJECTS_EXCEEDED bug check has a value of 0x0000000C. This indicates that the current thread exceeded the permitted number of wait objects.

Parameters

None

Cause

This bug check results from the improper use of **KeWaitForMultipleObjects** or **FsRtlCancellableWaitForMultipleObjects**.

The caller may pass a pointer to a buffer in this routine's *WaitBlockArray* parameter. The system will use this buffer to keep track of wait objects.

If a buffer is supplied, the *Count* parameter may not exceed MAXIMUM_WAIT_OBJECTS. If no buffer is supplied, the *Count* parameter may not exceed THREAD_WAIT_OBJECTS.

If the value of *Count* exceeds the allowable value, this bug check is issued.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD: MUTEX_LEVEL_NUMBER_VIOLATION

The MUTEX_LEVEL_NUMBER_VIOLATION bug check has a value of 0x0000000D.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE: NO_USER_MODE_CONTEXT

The NO_USER_MODE_CONTEXT bug check has a value of 0x0000000E.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF: SPIN_LOCK_ALREADY_OWNED

The SPIN_LOCK_ALREADY_OWNED bug check has a value of 0x0000000F. This indicates that a request for a spin lock has been initiated when the spin lock was already owned.

Parameters

None

Cause

Typically, this error is caused by a recursive request for a spin lock. It can also occur if something similar to a recursive request for a spin lock has been initiated—for example, when a spin lock has been acquired by a thread, and then that same thread calls a function, which also tries to acquire a spin lock. The second attempt to acquire a spin lock is not blocked in this case because doing so would result in an unrecoverable deadlock. If the calls are made on more than one processor, then one processor will be blocked until the other processor releases the lock.

This error can also occur, without explicit recursion, when all threads and all spin locks are assigned an IRQL. Spin lock IRQLs are always greater than or equal to DPC level, but this is not true for threads. However, a thread that is holding a spin lock must maintain an IRQL greater than or equal to that of the spin lock. Decreasing the thread IRQL below the IRQL level of the spin lock that it is holding allows another thread to be scheduled on the processor. This new thread could then attempt to acquire the same spin lock.

Resolving the Problem

Ensure that you are not recursively acquiring the lock. And, for threads that hold a spin lock, ensure that you are not decreasing the thread IRQL to a level below the IRQL of the spin lock that it is holding.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10: SPIN_LOCK_NOT_OWNED

The SPIN_LOCK_NOT_OWNED bug check has a value of 0x00000010.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x11: THREAD_NOT_MUTEX_OWNER

The THREAD_NOT_MUTEX_OWNER bug check has a value of 0x00000011.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x12: TRAP_CAUSE_UNKNOWN

The TRAP_CAUSE_UNKNOWN bug check has a value of 0x00000012. This indicates that an unknown exception has occurred.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The unexpected interrupt
2	The unknown floating-point exception
3	The enabled and asserted status bits. See the processor definition for details.
4	Reserved

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x13: EMPTY_THREAD_REAPER_LIST

The EMPTY_THREAD_REAPER_LIST bug check has a value of 0x00000013.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x14: CREATE_DELETE_LOCK_NOT_LOCKED

The CREATE_DELETE_LOCK_NOT_LOCKED bug check has a value of 0x00000014.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x15: LAST_CHANCE_CALLED_FROM_KMODE

The LAST_CHANCE_CALLED_FROM_KMODE bug check has a value of 0x00000015.

This bug check appears very infrequently.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x16: CID_HANDLE_CREATION

The CID_HANDLE_CREATION bug check has a value of 0x00000016.

This bug check appears very infrequently.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x17: CID_HANDLE_DELETION

The CID_HANDLE_DELETION bug check has a value of 0x00000017.

This bug check appears very infrequently.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x18: REFERENCE_BY_POINTER

The REFERENCE_BY_POINTER bug check has a value of 0x00000018. This indicates that the reference count of an object is illegal for the current state of the object.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Object type of the object whose reference count is being lowered.
2	Object whose reference count is being lowered.
3	Reserved
4	Reserved

Cause

The reference count of an object is illegal for the current state of the object. Each time a driver uses a pointer to an object, the driver calls a kernel routine to increase the reference count of the object by one. When the driver is done with the pointer, the driver calls another kernel routine to decrease the reference count by one.

Drivers must match calls to the routines that increase (*reference*) and decrease (*dereference*) the reference count. This bug check is caused by an inconsistency in the object's reference count. Typically, the inconsistency is caused by a driver that decreases the reference count of an object too many times, making extra calls that dereference the object. This bug check can occur because an object's reference count goes to zero while there are still open handles to the object. It might also occur when the object's reference count drops below zero, whether or not there are open handles to the object.

Resolving the Problem

Make sure that the driver matches calls to the routines that increase and decrease the reference count of the object. Make sure that your driver does not make extra calls to routines that dereference the object (see Parameter 2).

You can use a debugger to help analyze this problem. To find the handle and pointer count on the object, use the **!object** debugger command.

kd> !object *address*

Where *address* is the address of the object given in Parameter 2.

Send feedback on this topic
Debugging Tools for Windows

December 09, 2009

Debugging Tools for Windows

Bug Check 0x19: BAD_POOL_HEADER

The BAD_POOL_HEADER bug check has a value of 0x00000019. This indicates that a pool header is corrupt.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x2	The pool entry being checked	The size of the pool block	0	The special pool pattern check failed. (The owner has likely corrupted the pool block.)
0x3	The pool entry being checked	The read-back **flink** freelist value	The read-back **blink** freelist value	The pool freelist is corrupt. (In a healthy list, the values of Parameters 2, 3, and 4 should be identical.)
0x5	One of the pool entries	Reserved	The other pool entry	A pair of adjacent pool entries have headers that contradict each other. At least one of them is corrupt.
0x6	One incorrectly-calculated entry	Reserved	The bad entry that caused the miscalculation	The pool block header's previous size is too large.
0x7	0	Reserved	The bad pool entry	The pool block header size is corrupt.
0x8	0	Reserved	The bad pool entry	The pool block header size is zero.
0x9	One incorrectly-calculated entry	Reserved	The bad entry that caused the miscalculation	The pool block header size is corrupted (it is too large).
0xA	The pool entry that should have been found	Reserved	The virtual address of the page that should have contained the pool entry	The pool block header size is corrupt.
0x20	The pool entry that should have been found	The next pool entry	Reserved	The pool block header size is corrupt.

Cause

The pool is already corrupted at the time of the current request.

This may or may not be due to the caller.

Resolving the Problem

The internal pool links must be walked to figure out a possible cause of the problem.

Then you can use special pool for the suspect pool tags, or use Driver Verifier on the suspect driver. The **!analyze** extension may be of help in pinpointing the suspect driver, but this is frequently not the case with pool corrupters.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1A: MEMORY_MANAGEMENT

The MEMORY_MANAGEMENT bug check has a value of 0x0000001A. This indicates that a severe memory management error occurred.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 is the only parameter of interest; this identifies the exact violation.

Parameter 1	Cause of Error
0x1	The fork clone block reference count is corrupt. (This only occurs on checked builds of Windows.)
0x777	The caller is unlocking a system cache address that is not currently locked. (This address was either never mapped or is being unlocked twice.)
0x778	The system is using the very last system cache view address, instead of preserving it.
0x780 0x781	The PTEs mapping the argument system cache view have been corrupted.
0x1000	A caller of **MmGetSystemAddressForMdl*** tried to map a fully-cached physical page as non-cached. This action would cause a conflicting hardware translation buffer entry, and so it was refused by the operating system. Since the caller specified "bug check on failure" in the requesting MDL, the system had no choice but to issue a bug check in this instance.
0x1010	The caller is unlocking a pageable section that is not currently locked. (This section was either never locked or is being unlocked twice.)
0x1234	The caller is trying lock a nonexistent pageable section.

0x1235	The caller is trying to protect an MDL with an invalid mapping.
0x3451	The PTEs of a kernel thread stack that has been swapped out are corrupted.
0x8888	Internal memory management structures are corrupted.
0x8889	
0x41283	The working set index encoded in the PTE is corrupted.
0x41284	A PTE or the working set list is corrupted.
0x41286	The caller is trying to free an invalid pool address.
0x41785	The working set list is corrupted.
0x41287	Internal memory management structures are corrupted. To further investigate the cause, a kernel memory dump file is needed.
0x61940	A PDE has been unexpectedly invalidated.
0x03030303	The boot loader is broken. (This value applies only to Intel Itanium machines.)
Other	An unknown memory management error occurred.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1B: PFN_SHARE_COUNT

The PFN_SHARE_COUNT bug check has a value of 0x0000001B.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1C: PFN_REFERENCE_COUNT

The PFN_REFERENCE_COUNT bug check has a value of 0x0000001C.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1D: NO_SPIN_LOCK_AVAILABLE

The NO_SPIN_LOCK_AVAILABLE bug check has a value of 0x0000001D.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1E: KMODE_EXCEPTION_NOT_HANDLED

The KMODE_EXCEPTION_NOT_HANDLED bug check has a value of 0x0000001E. This indicates that a kernel-mode program generated an exception which the error handler did not catch.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The exception code that was not handled
2	The address at which the exception occurred
3	Parameter 0 of the exception

4 Parameter 1 of the exception

Cause

This is a very common bug check. To interpret it, you must identify which exception was generated.

Common exception codes include:

- 0x80000002: STATUS_DATATYPE_MISALIGNMENT

  An unaligned data reference was encountered.

- 0x80000003: STATUS_BREAKPOINT

  A breakpoint or ASSERT was encountered when no kernel debugger was attached to the system.

- 0xC0000005: STATUS_ACCESS_VIOLATION

  A memory access violation occurred. (Parameter 4 of the bug check is the address that the driver attempted to access.)

For a complete list of exception codes, see the *ntstatus.h* file located in the *inc* directory of the Windows Driver Kit.

Resolving the Problem

If you are not equipped to debug this problem, you should use some basic troubleshooting techniques. If a driver is identified in the bug check message, disable the driver or check with the manufacturer for driver updates. Try changing video adapters. Check with your hardware vendor for any BIOS updates. Disable BIOS memory options such as caching or shadowing.

If you plan to debug this problem, you may find it difficult to obtain a stack trace. Parameter 2 (the exception address) should pinpoint the driver or function that caused this problem.

If exception code 0x80000003 occurs, this indicates that a hard-coded breakpoint or assertion was hit, but the system was started with the **/NODEBUG** switch. This problem should rarely occur. If it occurs repeatedly, make sure a kernel debugger is connected and the system is started with the **/DEBUG** switch.

If exception code 0x80000002 occurs, the trap frame will supply additional information.

If the specific cause of the exception is unknown, the following should be considered:

Hardware incompatibility. First, make sure that any new hardware installed is listed in the Microsoft Windows Marketplace Tested Products List.

Faulty device driver or system service. In addition, a faulty device driver or system service might be responsible for this error. Hardware issues, such as BIOS incompatibilities, memory conflicts, and IRQ conflicts can also generate this error.

If a driver is listed by name within the bug check message, disable or remove that driver. Disable or remove any drivers or services that were recently added. If the error occurs during the startup sequence and the system partition is formatted with NTFS file system, you might be able to use Safe Mode to rename or delete the faulty driver. If the driver is used as part of the system startup process in Safe Mode, you need to start the computer by using the Recovery Console to access the file.

If the problem is associated with *Win32k.sys*, the source of the error might be a third-party remote control program. If such software is installed, the service can be removed by starting the system using the Recovery Console and deleting the offending system service file.

Check the System Log in **Event Viewer** for additional error messages that might help pinpoint the device or driver that is causing bug check 0x1E. Disabling memory caching of the BIOS might also resolve the error. You should also run hardware diagnostics, especially the memory scanner, supplied by the system manufacturer. For details on these procedures, see the owner's manual for your computer.

The error that generates this message can occur after the first restart during Windows Setup, or after Setup is finished. A possible cause of the error is a system BIOS incompatibility. BIOS problems can be resolved by upgrading the system BIOS version.

To get a stack trace if the normal stack tracing procedures fail

1. Use the **kb (Display Stack Backtrace)** command to display parameters in the stack trace. Look for the call to **NT!PspUnhandledExceptionInSystemThread**. (If this function is not listed, see the note below.)
2. The first parameter to **NT!PspUnhandledExceptionInSystemThread** is a pointer to a structure, which contains pointers to an **except** statement:

```
typedef struct _EXCEPTION_POINTERS {
    PEXCEPTION_RECORD ExceptionRecord;
    PCONTEXT ContextRecord;
    } EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;

ULONG PspUnhandledExceptionInSystemThread(
    IN PEXCEPTION_POINTERS ExceptionPointers
    )
```

   Use the **dd (Display Memory)** command on that address to display the necessary data.

3. The first retrieved value is an exception record and the second is a context record. Use the **.exr (Display Exception Record)** command and the **.cxr (Display Context Record)** command with these two values as their arguments, respectively.
4. After the **.cxr** command executes, use the **kb** command to display a stack trace that is based on the context record information. This stack trace indicates the calling stack where the unhandled exception occurred.

Note This procedure assumes that you can locate **NT!PspUnhandledExceptionInSystemThread**. However, in some cases (such as an access violation crash) you will not be able to do this. In that case, look for **ntoskrnl!KiDispatchException**. The third parameter passed to this function is a trap frame address. Use the **.trap (Display Trap Frame)** command with this address to set the Register Context to the proper value. You can then perform stack traces and issue other commands.

Here is an example of bug check 0x1E on an x86 processor:

```
kd> .bugcheck                 get the bug check data
Bugcheck code 0000001e
Arguments c0000005 8013cd0a 00000000 0362cffff

kd> kb                        start with a stack trace
FramePtr  RetAddr   Param1   Param2   Param3   Function Name
8013ed5c  801263ba  00000000 00000000 fe40cb00 NT!_DbgBreakPoint
8013eecc  8013313c  0000001e c0000005 8013cd0a NT!_KeBugCheckEx+0x194
fe40cad0  8013318e  fe40caf8 801359ff fe40cb00 NT!PspUnhandledExceptionInSystemThread+0x18
fe40cad8  801359ff  fe40cb00 00000000 fe40cb00 NT!PspSystemThreadStartup+0x4a
fe40cf7c  8013cb8e  fe43a44c ff6ce388 00000000 NT!_except_handler3+0x47
00000000  00000000  00000000 00000000 00000000 NT!KiThreadStartup+0xe

kd> dd fe40caf8 L2            dump EXCEPTION_POINTERS structure
0xFE40CAF8  fe40cd88 fe40cbc4                   ..@...@.

kd> .exr fe40cd88             first DWORD is the exception record
Exception Record @ FE40CD88:
   ExceptionCode: c0000005
  ExceptionFlags: 00000000
  Chained Record: 00000000
ExceptionAddress: 8013cd0a
NumberParameters: 00000002
   Parameter[0]: 00000000
   Parameter[1]: 0362cfff

kd> .cxr fe40cbc4             second DWORD is the context record
CtxFlags: 00010017
eax=00087000 ebx=00000000 ecx=03ff0000 edx=ff63d000 esi=0362cfff edi=036b3fff
eip=8013cd0a esp=fe40ce50 ebp=fe40cef8 iopl=0         nv dn ei pl nz ac po cy
vip=0    vif=0
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010617
0x8013cd0a  f3a4              rep movsb

kd> kb                        kb gives stack for context record
ChildEBP RetAddr  Args to Child
fe40ce54 80402e09 ff6c4000 ff63d000 03ff0000 NT!_RtlMoveMemory@12+0x3e
fe40ce68 80403c18 ffbc0c28 ff6ce008 ff6c4000 HAL!_HalpCopyBufferMap@20+0x49
fe40ce9c fe43b1e4 ff6cef90 ffbc0c28 ff6ce009 HAL!_IoFlushAdapterBuffers@24+0x148
fe40ceb8 fe4385b4 ff6ce388 6cd00800 ffbc0c28 QIC117!_kdi_FlushDMABuffers@20+0x28
fe40cef8 fe439894 ff6cd008 ffb6c820 fe40cf4c QIC117!_cqd_CmdReadWrite@8+0x26e
fe40cf18 fe437d92 ff6cd008 ffb6c820 ff6e4e50 QIC117!_cqd_DispatchFRB@8+0x210
fe40cf30 fe43a4f5 ff6cd008 ffb6c820 00000000 QIC117!_cqd_ProcessFRB@8+0x134
fe40cf4c 80133184 ff6ce388 00000000 00000000 QIC117!_kdi_ThreadRun@4+0xa9
fe40cf7c 8013cb8e fe43a44c ff6ce388 00000000 NT!_PspSystemThreadStartup@8+0x40
```


Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1F: SHARED_RESOURCE_CONV_ERROR

The SHARED_RESOURCE_CONV_ERROR bug check has a value of 0x0000001F.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x20: KERNEL_APC_PENDING_DURING_EXIT

The KERNEL_APC_PENDING_DURING_EXIT bug check has a value of 0x00000020. This indicates that an asynchronous procedure call (APC) was still pending when a thread exited.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description

1	The address of the APC found pending during exit
2	The thread's APC disable count
3	The current IRQL
4	Reserved

Cause

The key data item is the APC disable count (Parameter 2) for the thread. If the count is nonzero, it will indicate the source of the problem.

The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex.

The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**.

Because these calls should always be in pairs, the APC disable count should be zero when a thread exits. A negative value indicates that a driver has disabled APC calls without re-enabling them. A positive value indicates that the reverse is true.

If you ever see this error, be very suspicious of all drivers installed on the machine — especially unusual or non-standard drivers.

This current IRQL (Parameter 3) should be zero. If it is not, the driver's cancellation routine may have caused this bug check by returning at an elevated IRQL. In this case, carefully note what was running (and what was closing) at the time of the crash, and note all of the installed drivers at the time of the crash. The cause in this case is usually a severe bug in a driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x21: QUOTA_UNDERFLOW

The QUOTA_UNDERFLOW bug check has a value of 0x00000021. This indicates that quota charges have been mishandled by returning more quota to a particular block than was previously charged.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The process that was initially charged, if available.
2	The quota type. For the list of all possible quota type values, see the header file *Ps.h* in the Windows Driver Kit (WDK).
3	The initial charged amount of quota to return.
4	The remaining amount of quota that was not returned.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x22: FILE_SYSTEM

The FILE_SYSTEM bug check has a value of 0x00000022.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x23: FAT_FILE_SYSTEM

The FAT_FILE_SYSTEM bug check has a value of 0x00000023. This indicates that a problem occurred in the FAT file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.

2	If **FatExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **FatExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved

Cause

One possible cause of this bug check is disk corruption. Corruption in the file system or bad blocks (sectors) on the disk can induce this error. Corrupted SCSI and IDE drivers can also adversely affect the system's ability to read and write to the disk, thus causing the error.

Another possible cause is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a disk corruption problem: Check Event Viewer for error messages from SCSI and FASTFAT (System Log) or Autochk (Application Log) that might help pinpoint the device or driver that is causing the error. Try disabling any virus scanners, backup programs, or disk defragmenter tools that continually monitor the system. You should also run hardware diagnostics supplied by the system manufacturer. For details on these procedures, see the owner's manual for your computer. Run **Chkdsk /f /r** to detect and resolve any file system structural corruption. You must restart the system before the disk scan begins on a system partition.

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x24: NTFS_FILE_SYSTEM

The NTFS_FILE_SYSTEM bug check has a value of 0x00000024. This indicates a problem occurred in *ntfs.sys*, the driver file that allows the system to read and write to NTFS drives.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	If **NtfsExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **NtfsExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved

Cause

One possible cause of this bug check is disk corruption. Corruption in the NTFS file system or bad blocks (sectors) on the hard disk can induce this error. Corrupted SCSI and IDE drivers can also adversely affect the system's ability to read and write to disk, thus causing the error.

Another possible cause is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a disk corruption problem: Check Event Viewer for error messages from SCSI and FASTFAT (System Log) or Autochk (Application Log) that might help pinpoint the device or driver that is causing the error. Try disabling any virus scanners, backup programs, or disk defragmenter tools that continually monitor the system. You should also run hardware diagnostics supplied by the system manufacturer. For details on these procedures, see the owner's manual for your computer. Run **Chkdsk /f /r** to detect and resolve any file system structural corruption. You must restart the system before the disk scan begins on a system partition.

To resolve a nonpaged pool memory depletion problem: Either add new physical memory to the computer (thus increasing the quantity of nonpaged pool memory available to the kernel), or reduce the number of files on the Services for Macintosh (SFM) volume.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x25: NPFS_FILE_SYSTEM

The NPFS_FILE_SYSTEM bug check has a value of 0x00000025. This indicates that a problem occurred in the NPFS file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	Reserved
3	Reserved
4	Reserved

Cause

One possible cause of this bug check is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x26: CDFS_FILE_SYSTEM

The CDFS_FILE_SYSTEM bug check has a value of 0x00000026. This indicates that a problem occurred in the CD file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	If **CdExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **CdExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved

Cause

One possible cause of this bug check is disk corruption. Corruption in the file system or bad blocks (sectors) on the disk can induce this error. Corrupted SCSI and IDE drivers can also adversely affect the system's ability to read and write to the disk, thus causing the error.

Another possible cause is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a disk corruption problem: Check Event Viewer for error messages from SCSI and FASTFAT (System Log) or Autochk (Application Log) that might help pinpoint the device or driver that is causing the error. Try disabling any virus scanners, backup programs, or disk defragmenter tools that continually monitor the system. You should also run hardware diagnostics supplied by the system manufacturer. For details on these procedures, see the owner's manual for your computer. Run **Chkdsk /f /r** to detect and resolve any file system structural corruption. You must restart the system before the disk scan begins on a system partition.

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x27: RDR_FILE_SYSTEM

The RDR_FILE_SYSTEM bug check has a value of 0x00000027. This indicates that a problem occurred in the SMB redirector file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The high 16 bits (the first four hexadecimal digits after the "0x") identify the type of problem. Possible values include: 0xCA550000 RDBSS_BUG_CHECK_CACHESUP 0xC1EE0000 RDBSS_BUG_CHECK_CLEANUP 0xC10E0000 RDBSS_BUG_CHECK_CLOSE 0xBAAD0000 RDBSS_BUG_CHECK_NTEXCEPT
2	If **RxExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **RxExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved

Cause

One possible cause of this bug check is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x28: CORRUPT_ACCESS_TOKEN

The CORRUPT_ACCESS_TOKEN bug check has a value of 0x00000028.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x29: SECURITY_SYSTEM

The SECURITY_SYSTEM bug check has a value of 0x00000029.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2A: INCONSISTENT_IRP

The INCONSISTENT_IRP bug check has a value of 0x0000002A. This indicates that an IRP was found to contain inconsistent information.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the IRP that was found to be inconsistent
2	Reserved

3	Reserved
4	Reserved

Cause

An IRP was discovered to be in an inconsistent state. Usually this means some field of the IRP was inconsistent with the remaining state of the IRP. An example would be an IRP that was being completed, but was still marked as being queued to a driver's device queue.

Comments

This bug check code is not currently being used in the system, but exists for debugging purposes.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2B: PANIC_STACK_SWITCH

The PANIC_STACK_SWITCH bug check has a value of 0x0000002B. This indicates that the kernel mode stack was overrun.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The trap frame
2	Reserved
3	Reserved
4	Reserved

Cause

This error normally appears when a kernel-mode driver uses too much stack space. It can also appear when serious data corruption occurs in the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2C: PORT_DRIVER_INTERNAL

The PORT_DRIVER_INTERNAL bug check has a value of 0x0000002C.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2D: SCSI_DISK_DRIVER_INTERNAL

The SCSI_DISK_DRIVER_INTERNAL bug check has a value of 0x0000002D.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2E: DATA_BUS_ERROR

The DATA_BUS_ERROR bug check has a value of 0x0000002E. This typically indicates that a parity error in system memory has been detected.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address that caused the fault
2	Physical address that caused the fault
3	Processor status register (PSR)
4	Faulting instruction register (FIR)

Cause

This error is almost always caused by a hardware problem — a configuration issue, defective hardware, or incompatible hardware.

The most common hardware problems that can cause this error are defective RAM, Level 2 (L2) RAM cache errors, or video RAM errors. Hard disk corruption can also cause this error.

This bug check can also be caused when a device driver attempts to access an address in the 0x8*xxxxxxx* range that does not exist (in other words, that does not have a physical address mapping).

Resolving the Problem

Resolving a hardware problem: If hardware has recently been added to the system, remove it to see if the error recurs.

If existing hardware has failed, remove or replace the faulty component. You should run hardware diagnostics supplied by the system manufacturer to determine which hardware component has failed. For details on these procedures, see the owner's manual for your computer. Check that all adapter cards in the computer are properly seated. Use an ink eraser or an electrical contact treatment, available at electronics supply stores, to ensure that adapter card contacts are clean.

If the problem occurs on a newly installed system, check the availability of updates for the BIOS, the SCSI controller or network cards. Updates of this kind are typically available on the Web site or the bulletin board system (BBS) of the hardware manufacturer.

If the error occurs after installing a new or updated device driver, the driver should be removed or replaced. If, under this circumstance, the error occurs during startup and the system partition is formatted with NTFS, you might be able to use Safe Mode to rename or delete the faulty driver.

If the driver is used as part of the system startup process in Safe Mode, you need to start the computer using the Recovery Console in order to access the file.

For additional error messages that might help pinpoint the device or driver that is causing the error, check the System Log in Event Viewer. Disabling memory caching or shadowing in the BIOS might also resolve this error. In addition, check the system for viruses, using any up-to-date commercial virus scanning software that examines the Master Boot Record of the hard disk. All Windows file systems can be infected by viruses.

Resolving a hard disk corruption problem: Run **Chkdsk /f /r** on the system partition. You must restart the system before the disk scan begins. If you cannot start the system due to the error, use the Recovery Console and run **Chkdsk /r**.

Warning If your system partition is formatted with the file allocation table (FAT) file system, the long filenames used by Windows can be damaged if Scandisk or another Microsoft MS-DOS-based hard disk tool is used to verify the integrity of your hard disk from MS-DOS. Always use the version of Chkdsk that matches your Windows version.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x2F: INSTRUCTION_BUS_ERROR

The INSTRUCTION_BUS_ERROR bug check has a value of 0x0000002F.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x30: SET_OF_INVALID_CONTEXT

The SET_OF_INVALID_CONTEXT bug check has a value of 0x00000030. This indicates that the stack pointer in a trap frame had an invalid value.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The new stack pointer

2	The old stack pointer
3	The trap frame address
4	0

Cause

This bug check occurs when some routine attempts to set the stack pointer in the trap frame to a lower value than the current stack pointer value.

If this error were not caught, it would cause the kernel to run with a stack pointer pointing to stack which is no longer valid.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x31: PHASE0_INITIALIZATION_FAILED

The PHASE0_INITIALIZATION_FAILED bug check has a value of 0x00000031. This indicates that system initialization failed.

Parameters

None

Cause

System initialization failed at a very early stage.

Resolving the Problem

A debugger is required to analyze this.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x32: PHASE1_INITIALIZATION_FAILED

The PHASE1_INITIALIZATION_FAILED bug check has a value of 0x00000032. This indicates that system initialization failed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The NT status code that describes why the system initialization failed
2	Reserved
3	Reserved
4	Reserved

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x33: UNEXPECTED_INITIALIZATION_CALL

The UNEXPECTED_INITIALIZATION_CALL bug check has a value of 0x00000033.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x34: CACHE_MANAGER

The CACHE_MANAGER bug check has a value of 0x00000034. This indicates that a problem occurred in the file system's cache manager.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	Reserved
3	Reserved
4	Reserved

Cause

One possible cause of this bug check is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x35: NO_MORE_IRP_STACK_LOCATIONS

The NO_MORE_IRP_STACK_LOCATIONS bug check has a value of 0x00000035. This bug check occurs when the **IoCallDriver** packet has no more stack locations remaining.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Address of the IRP
2	Reserved
3	Reserved
4	Reserved

Cause

A higher-level driver has attempted to call a lower-level driver through the **IoCallDriver** interface, but there are no more stack locations in the packet. This will prevent the lower-level driver from accessing its parameters.

This is a disastrous situation, since the higher level driver is proceeding as if it has filled in the parameters for the lower level driver (as required). But since there is no stack location for the latter driver, the former has actually written off the end of the packet. This means that some other memory has been corrupted as well.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x36: DEVICE_REFERENCE_COUNT_NOT_ZERO

The DEVICE_REFERENCE_COUNT_NOT_ZERO bug check has a value of 0x00000036. This indicates that a driver attempted to delete a device object that still had a positive reference count.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the device object
2	Reserved

3	Reserved
4	Reserved

Cause

A device driver has attempted to delete one of its device objects from the system, but the reference count for that object was non-zero.

This means there are still outstanding references to the device. (The reference count indicates the number of reasons why this device object cannot be deleted.)

This is a bug in the calling device driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x37: FLOPPY_INTERNAL_ERROR

The FLOPPY_INTERNAL_ERROR bug check has a value of 0x00000037.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x38: SERIAL_DRIVER_INTERNAL

The SERIAL_DRIVER_INTERNAL bug check has a value of 0x00000038.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x39: SYSTEM_EXIT_OWNED_MUTEX

The SYSTEM_EXIT_OWNED_MUTEX bug check has a value of 0x00000039. This indicates that the worker routine returned without releasing the mutex object that it owned.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the worker routine that caused the error.
2	The parameter passed to the worker routine.
3	The address of the work item.
4	Reserved.

Cause

The worker routine returned while it still owned a mutex object. The current worker thread will proceed to run other unrelated work items, and the mutex will never be released.

Resolving the Problem

A debugger is required to analyze this problem. To find the driver that caused the error, use the **ln** (List Nearest Symbols) debugger command:

kd> ln address

Where address is the worker routine given in Parameter 1.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3A: SYSTEM_UNWIND_PREVIOUS_USER

The SYSTEM_UNWIND_PREVIOUS_USER bug check has a value of 0x0000003A.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3B: SYSTEM_SERVICE_EXCEPTION

The SYSTEM_SERVICE_EXCEPTION bug check has a value of 0x0000003B. This indicates that an exception happened while executing a routine that transitions from non-privileged code to privileged code.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The exception that caused the bug check
2	The address of the exception record for the exception that caused the bug check
3	The address of the context record for the exception that caused the bug check
4	0

Cause

This error has been linked to excessive paged pool usage and may occur due to user-mode graphics drivers crossing over and passing bad data to the kernel code.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3C: INTERRUPT_UNWIND_ATTEMPTED

The INTERRUPT_UNWIND_ATTEMPTED bug check has a value of 0x0000003C.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3D: INTERRUPT_EXCEPTION_NOT_HANDLED

The INTERRUPT_EXCEPTION_NOT_HANDLED bug check has a value of 0x0000003D.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3E: MULTIPROCESSOR_CONFIGURATION_NOT_SUPPORTED

The MULTIPROCESSOR_CONFIGURATION_NOT_SUPPORTED bug check has a value of 0x0000003E. This indicates that the system has multiple processors, but they are asymmetric in relation to one another.

Parameters

None

Cause

In order to be symmetric, all processors must be of the same type and level. This system contains processors of different types (for example, a Pentium processor and an 80486 processor).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x3F: NO_MORE_SYSTEM_PTES

The NO_MORE_SYSTEM_PTES bug check has a value of 0x0000003F. This is the result of a system which has performed too many I/O actions. This has resulted in fragmented system page table entries (PTE).

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	**0:** system expansion PTE type **1:** nonpaged pool expansion PTE type
2	Size of memory request
3	Total free system PTEs
4	Total system PTEs

Cause

In almost all cases, the system is not actually out of PTEs. Rather, a driver has requested a large block of memory, but there is no contiguous block of sufficient size to satisfy this request.

Often video drivers will allocate large amounts of kernel memory that must succeed. Some backup programs do the same.

Resolving the Problem

A possible work-around: Modify the registry to increase the total number of system PTEs. If this does not help, remove any recently-installed software, especially backup utilities or disk-intensive applications.

Debugging the problem: The following method can be used to debug bug check 0x3F.

First, get a stack trace, and use the **!sysptes 3** extension command.

Then set **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\TrackPtes** equal to DWORD 1, and reboot. This will cause the system to save stack traces.

This allows you to display more detailed information about the PTE owners. For example:

```
0: kd> !sysptes 4

0x2c47 System PTEs allocated to mapping locked pages

VA       MDL     PageCount  Caller/CallersCaller
f0e5db48 eb6ceef0         1 ntkrpamp!MmMapLockedPages+0x15/ntkrpamp!IopfCallDriver+0x35
f0c3fe48 eb634bf0         1 netbt!NbtTdiAssociateConnection+0x1f/netbt!DelayedNbtProcessConnect+0x17c
f0db38e8 eb65b880         1 mrxsmb!SmbMmAllocateSessionEntry+0x89/mrxsmb!SmbCepInitializeExchange+0xda
f8312568 eb6df880         1 rdbss!RxCreateFromNetRoot+0x3d7/rdbss!RxCreateFromNetRoot+0x93
f8363908 eb685880         1 mrxsmb!SmbMmAllocateSessionEntry+0x89/mrxsmb!SmbCepInitializeExchange+0xda
f0c54248 eb640880         1 rdbss!RxCreateFromNetRoot+0x3d7/rdbss!RxCreateFromNetRoot+0x93
f0ddf448 eb5f3160         1 mrxsmb!MrxSmbUnalignedDirEntryCopyTail+0x387/mrxsmb!MRxSmbCoreInformation+0x36
f150bc08 eb6367b0         1 mrxsmb!MrxSmbUnalignedDirEntryCopyTail+0x387/mrxsmb!MRxSmbCoreInformation+0x36
f1392308 eb6fba70         1 netbt!NbtTdiOpenAddress+0x1fb/netbt!DelayedNbtProcessConnect+0x17c
eb1bee64 edac5000       200 VIDEOPRT!pVideoPortGetDeviceBase+0x118/VIDEOPRT!VideoPortMapMemory+0x45
f139b5a8 edd4b000        12 rdbss!FsRtlCopyWrite2+0x34/rdbss!RxDriverEntry+0x149
eb41f400 ede92000        20 VIDEOPRT!pVideoPortGetDeviceBase+0x139/VIDEOPRT!VideoPortGetDeviceBase+0x1b
eb41f198 edf2a000        20 NDIS!NdisReadNetworkAddress+0x3a/NDIS!NdisFreeSharedMemory+0x58
eb41f1e4 eb110000        10 VIDEOPRT!pVideoPortGetDeviceBase+0x139/VIDEOPRT!VideoPortGetDeviceBase+0x1b
......
```

If the system runs out of PTEs again after the **TrackPtes** registry value has been set, bug check 0xD8 (DRIVER_USED_EXCESSIVE_PTES) will be issued instead of 0x3F.

The name of the driver causing this error will be displayed as well.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x40: TARGET_MDL_TOO_SMALL

The TARGET_MDL_TOO_SMALL bug check has a value of 0x00000040. This indicates that a driver has improperly used **IoBuildPartialMdl**.

Parameters

None

Cause

This is a driver bug. A driver has called the **IoBuildPartialMdl** function and passed it an MDL to map part of a source MDL, but the target MDL is not large enough to map the entire range of addresses requested.

Resolving the Problem

The source and target MDLs, as well as the address range length to be mapped, are the first, second, and fourth arguments to the **IoBuildPartialMdl** function. Therefore, doing a stack trace on this particular function might help during the debugging process. Ensure that your code is correctly calculating the necessary size for the target MDL for the address range length that you are passing to this function.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x41: MUST_SUCCEED_POOL_EMPTY

The MUST_SUCCEED_POOL_EMPTY bug check has a value of 0x00000041. This indicates that a kernel-mode thread has requested too much must-succeed pool.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The size of the request that could not be satisfied
2	The number of pages used from nonpaged pool
3	The number of requests from nonpaged pool larger than PAGE_SIZE
4	The number of pages available

Cause

In Microsoft Windows 2000, only a small amount of must-succeed pool is permitted. In Windows XP and later, no driver is permitted to request must-succeed pool.

If a must-succeed request cannot be filled, this bug check is issued.

Resolving the Problem

Replace or rewrite the driver which is making the request. A driver should not request must-succeed pool. Instead, it should ask for normal pool and gracefully handle the scenario where the pool is temporarily empty.

The **kb (Display Stack Backtrace)** command will show the driver that caused the error.

Additionally, it is possible that a second component has depleted the must-succeed pool. To determine if this is the case, first use the **kb** command. Then use **!vm 1** to display total pool usage, **!poolused 2** to display per-tag nonpaged pool usage, and **!poolused 4** to display per-tag paged pool usage. The component associated with the tag using the most pool is probably the source of the problem.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x42: ATDISK_DRIVER_INTERNAL

The ATDISK_DRIVER_INTERNAL bug check has a value of 0x00000042.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x43: NO_SUCH_PARTITION

The NO_SUCH_PARTITION bug check has a value of 0x00000043.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x44: MULTIPLE_IRP_COMPLETE_REQUESTS

The MULTIPLE_IRP_COMPLETE_REQUESTS bug check has a value of 0x00000044. This indicates that a driver has tried to requested an IRP be completed that is already complete.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the IRP
2	Reserved
3	Reserved
4	Reserved

Cause

A driver has called **IoCompleteRequest** to ask that an IRP be completed, but the packet has already been completed.

Resolving the Problem

This is a tough bug to find because the simplest case — a driver that attempted to complete its own packet twice — is usually not the source of the problem. More likely, two separate drivers each believe that they own the packet, and each has attempted to complete it. The first request succeeds, and the second fails, resulting in this bug check.

Tracking down which drivers in the system caused the error is difficult, because the trail of the first driver has been covered by the second. However, the driver stack for the current request can be found by examining the device object fields in each of the stack locations.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x45: INSUFFICIENT_SYSTEM_MAP_REGS

The INSUFFICIENT_SYSTEM_MAP_REGS bug check has a value of 0x00000045.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x46: DEREF_UNKNOWN_LOGON_SESSION

The DEREF_UNKNOWN_LOGON_SESSION bug check has a value of 0x00000046.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x47: REF_UNKNOWN_LOGON_SESSION

The REF_UNKNOWN_LOGON_SESSION bug check has a value of 0x00000047.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x48: CANCEL_STATE_IN_COMPLETED_IRP

The CANCEL_STATE_IN_COMPLETED_IRP bug check has a value of 0x00000048. This indicates that an I/O request packet (IRP) was completed, and then was subsequently canceled.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	A pointer to the IRP
2	The cancel routine set by the driver
3	Reserved
4	Reserved

Cause

An IRP that had a *Cancel* routine set was completed normally, without cancellation. But after it was complete, a driver called the IRP's *Cancel* routine.

This could be caused by a driver that completed the IRP and then attempted to cancel it.

It could also be caused by two drivers each trying to access the same IRP in an improper way.

Resolving the Problem

The cancel routine parameter can be used to determine which driver or stack caused the bug check.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x49: PAGE_FAULT_WITH_INTERRUPTS_OFF

The PAGE_FAULT_WITH_INTERRUPTS_OFF bug check has a value of 0x00000049.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4A: IRQL_GT_ZERO_AT_SYSTEM_SERVICE

The IRQL_GT_ZERO_AT_SYSTEM_SERVICE bug check has a value of 0x0000004A. This indicates that a thread is returning to user mode from a system call when its IRQL is still above PASSIVE_LEVEL.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the system function (system call routine)
2	The current IRQL
3	0
4	0

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4B: STREAMS_INTERNAL_ERROR

The STREAMS_INTERNAL_ERROR bug check has a value of 0x0000004B.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4C: FATAL_UNHANDLED_HARD_ERROR

The FATAL_UNHANDLED_HARD_ERROR bug check has a value of 0x0000004C.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4D: NO_PAGES_AVAILABLE

The NO_PAGES_AVAILABLE bug check has a value of 0x0000004D. This indicates that no free pages are available to continue operations.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The total number of dirty pages
2	The number of dirty pages destined for the page file
3	*Windows XP and Windows 2000:* The size of the nonpaged pool available at the time the bug check occurred *Windows Server 2003 and later:* Reserved
4	*Windows 2000:* The number of transition pages that are currently stranded *Windows XP and later:* The most recent modified write error status.

Cause

To see general memory statistics, use the **!vm 3** extension.

This bug check can occur for any of the following reasons:

- A driver has blocked, deadlocking the modified or mapped page writers. Examples of this include mutex deadlocks or accesses to paged out memory in file system drivers or filter drivers. This indicates a driver bug.

  If Parameter 1 or Parameter 2 is large, then this is a possibility. Use **!vm 3**.

- A storage driver is not processing requests. Examples of this are stranded queues and non-responding drives. This indicates a driver bug.

If Parameter 1 or Parameter 2 is large, then this is a possibility. Use **!vm 8**, followed by **!process 0 7**.

- A high-priority realtime thread has starved the balance set manager from trimming pages from the working set, or starved the modified page writer from writing them out. This indicates a bug in the component that created this thread.

  This situation is difficult to analyze. Try using **!ready**. Try also **!process 0 7** to list all threads and see if any have accumulated excessive kernel time as well as what their current priorities are. Such processes may have blocked out the memory management threads from making pages available.

- *Windows XP and Windows 2000:* Not enough pool is available for the storage stack to write out modified pages. This indicates a driver bug.

  If Parameter 3 is small, then this is a possibility. Use **!vm** and **!poolused 2**.

- *Windows 2000:* All the processes have been trimmed to their minimums and all modified pages written, but still no memory is available. The freed memory must be stuck in transition pages with non-zero reference counts — thus they cannot be put on the freelist.

  A driver is neglecting to unlock the pages preventing the reference counts from going to zero which would free the pages. This may be due to transfers that never finish, causing the driver routines to run endlessly, or to other driver bugs.

  If Parameter 4 is large, then this is a possibility. But it is very hard to find the driver. Try the **!process 0 1** extension and look for any drivers that have a lot of locked pages.

If the problem cannot be found, then try booting with a kernel debugger attached from the beginning, and monitor the situation.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4E: PFN_LIST_CORRUPT

The PFN_LIST_CORRUPT bug check has a value of 0x0000004E. This indicates that the page frame number (PFN) list is corrupted.

Parameters

The following parameters are displayed on the blue screen. **Parameter 1** indicates the type of violation. The meaning of the other parameters depends on the value of **Parameter 1**.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x01	The **ListHead** value that was corrupted	The number of pages available	0	The list head was corrupt.
0x02	The entry in the list that is being removed	The highest physical page number	The reference count of the entry being removed	A list entry was corrupt.
0x07	The page frame number	The current share count	0	A driver has unlocked a certain page more times than it locked it.
0x8F	New page number	Old page number	0	The free or zeroed page listhead is corrupt.
0x99	Page frame number	Current page state	0	A page table entry (PTE) or PFN is corrupt.
0x9A	Page frame number	Current page state	The reference count of the entry that is being removed	A driver attempted to free a page that is still locked for IO.

Cause

This error is typically caused by a driver passing a bad memory descriptor list. For example, the driver might have called **MmUnlockPages** twice with the same list.

If a kernel debugger is available, examine the stack trace.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x4F: NDIS_INTERNAL_ERROR

The NDIS_INTERNAL_ERROR bug check has a value of 0x0000004F.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x50: PAGE_FAULT_IN_NONPAGED_AREA

The PAGE_FAULT_IN_NONPAGED_AREA bug check has a value of 0x00000050. This indicates that invalid system memory has been referenced.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read operation **1:** Write operation
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

Bug check 0x50 usually occurs after the installation of faulty hardware or in the event of failure of installed hardware (usually related to defective RAM, be it main memory, L2 RAM cache, or video RAM).

Another common cause is the installation of a faulty system service.

Antivirus software can also trigger this error, as can a corrupted NTFS volume.

Resolving the Problem

Resolving a faulty hardware problem: If hardware has been added to the system recently, remove it to see if the error recurs. If existing hardware has failed, remove or replace the faulty component. You should run hardware diagnostics supplied by the system manufacturer. For details on these procedures, see the owner's manual for your computer.

Resolving a faulty system service problem: Disable the service and confirm that this resolves the error. If so, contact the manufacturer of the system service about a possible update. If the error occurs during system startup, restart your computer, and press F8 at the character-mode menu that displays the operating system choices. At the resulting Windows **Advanced Options** menu, choose the **Last Known Good Configuration** option. This option is most effective when only one driver or service is added at a time.

Resolving an antivirus software problem: Disable the program and confirm that this resolves the error. If it does, contact the manufacturer of the program about a possible update.

Resolving a corrupted NTFS volume problem: Run **Chkdsk /f /r** to detect and repair disk errors. You must restart the system before the disk scan begins on a system partition. If the hard disk is SCSI, check for problems between the SCSI controller and the disk.

Finally, check the System Log in Event Viewer for additional error messages that might help pinpoint the device or driver that is causing the error. Disabling memory caching of the BIOS might also resolve it.

Comments

Typically, this address is in freed memory or is simply invalid.

This cannot be protected by a **try - except** handler — it can only be protected by a probe.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x51: REGISTRY_ERROR

The REGISTRY_ERROR bug check has a value of 0x00000051. This indicates that a severe registry error has occurred.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Reserved
2	Reserved
3	The pointer to the hive (if available)
4	If the hive is corrupt, the return code of **HvCheckHive** (if available)

Cause

Something has gone wrong with the registry. If a kernel debugger is available, get a stack trace.

This error may indicate that the registry encountered an I/O error while trying to read one of its files. This can be caused by hardware problems or file system corruption.

It may also occur due to a failure in a refresh operation, which is used only in by the security system, and then only when resource limits are encountered.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x52: MAILSLOT_FILE_SYSTEM

The MAILSLOT_FILE_SYSTEM bug check has a value of 0x00000052.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x53: NO_BOOT_DEVICE

The NO_BOOT_DEVICE bug check has a value of 0x00000053.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x54: LM_SERVER_INTERNAL_ERROR

The LM_SERVER_INTERNAL_ERROR bug check has a value of 0x00000054.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x55: DATA_COHERENCY_EXCEPTION

The DATA_COHERENCY_EXCEPTION bug check has a value of 0x00000055.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x56: INSTRUCTION_COHERENCY_EXCEPTION

The INSTRUCTION_COHERENCY_EXCEPTION bug check has a value of 0x00000056.

This bug check appears very infrequently.

Debugging Tools for Windows

December 09, 2009

Debugging Tools for Windows

Bug Check 0x57: XNS_INTERNAL_ERROR

The XNS_INTERNAL_ERROR bug check has a value of 0x00000057.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x58: FTDISK_INTERNAL_ERROR

The FTDISK_INTERNAL_ERROR bug check has a value of 0x00000058. This is issued if the system is booted from the wrong copy of a mirrored partition.

Parameters

None

Cause

The hives are indicating that the mirror is valid, but it is not. The hives should actually be pointing to the shadow partition.

This is almost always caused by the primary partition being revived.

Resolving the Problem

Reboot the system from the shadow partition.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x59: PINBALL_FILE_SYSTEM

The PINBALL_FILE_SYSTEM bug check has a value of 0x00000059. This indicates that a problem occurred in the Pinball file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	Reserved
3	Reserved
4	Reserved

Cause

One possible cause of this bug check is depletion of nonpaged pool memory. If the nonpaged pool memory is completely depleted, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver requiring nonpaged pool memory can also trigger this error.

Resolving the Problem

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x5A: CRITICAL_SERVICE_FAILED

The CRITICAL_SERVICE_FAILED bug check has a value of 0x0000005A.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Bug Check 0x5B: SET_ENV_VAR_FAILED

The SET_ENV_VAR_FAILED bug check has a value of 0x0000005B.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Bug Check 0x5C: HAL_INITIALIZATION_FAILED

The HAL_INITIALIZATION_FAILED bug check has a value of 0x0000005C.

This indicates that the HAL initialization failed.

Debugging Tools for Windows
December 09, 2009

Bug Check 0x5D: UNSUPPORTED_PROCESSOR

The UNSUPPORTED_PROCESSOR bug check has a value of 0x0000005D. This indicates that the computer is attempting to run Windows on an unsupported processor.

Parameters

None

Cause

Windows requires a higher-grade processor than the one you are using.

Debugging Tools for Windows
December 09, 2009

Bug Check 0x5E: OBJECT_INITIALIZATION_FAILED

The OBJECT_INITIALIZATION_FAILED bug check has a value of 0x0000005E.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Bug Check 0x5F: SECURITY_INITIALIZATION_FAILED

The SECURITY_INITIALIZATION_FAILED bug check has a value of 0x0000005F.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x60: PROCESS_INITIALIZATION_FAILED

The PROCESS_INITIALIZATION_FAILED bug check has a value of 0x00000060.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x61: HAL1_INITIALIZATION_FAILED

The HAL1_INITIALIZATION_FAILED bug check has a value of 0x00000061.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x62: OBJECT1_INITIALIZATION_FAILED

The OBJECT1_INITIALIZATION_FAILED bug check has a value of 0x00000062.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x63: SECURITY1_INITIALIZATION_FAILED

The SECURITY1_INITIALIZATION_FAILED bug check has a value of 0x00000063.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x64: SYMBOLIC_INITIALIZATION_FAILED

The SYMBOLIC_INITIALIZATION_FAILED bug check has a value of 0x00000064.

This bug check appears very infrequently.

Debugging Tools for Windows

December 09, 2009

Debugging Tools for Windows

Bug Check 0x65: MEMORY1_INITIALIZATION_FAILED

The MEMORY1_INITIALIZATION_FAILED bug check has a value of 0x00000065.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x66: CACHE_INITIALIZATION_FAILED

The CACHE_INITIALIZATION_FAILED bug check has a value of 0x00000066.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x67: CONFIG_INITIALIZATION_FAILED

The CONFIG_INITIALIZATION_FAILED bug check has a value of 0x00000067. This bug check indicates that the registry configuration failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	Reserved
2	The location selector
3	The NT status code
4	Reserved

Cause

The registry could not allocate the pool that it needed to contain the registry files. This situation should never occur, because the register allocates this pool early enough in system initialization so that plenty of paged pool should be available.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x68: FILE_INITIALIZATION_FAILED

The FILE_INITIALIZATION_FAILED bug check has a value of 0x00000068.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x69: IO1_INITIALIZATION_FAILED

The IO1_INITIALIZATION_FAILED bug check has a value of 0x00000069. This bug check indicates that the initialization of the I/O system failed for some reason.

Parameters

None

Cause

There is very little information available to analyze this error.

Most likely, the setup routine has improperly installed the system, or a user has reconfigured the system.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6A: LPC_INITIALIZATION_FAILED

The LPC_INITIALIZATION_FAILED bug check has a value of 0x0000006A.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6B: PROCESS1_INITIALIZATION_FAILED

The PROCESS1_INITIALIZATION_FAILED bug check has a value of 0x0000006B. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the failure
2	Reserved
3	Reserved
4	Reserved

Cause

Any part of the disk subsystem can cause the PROCESS1_INITIALIZATION_FAILED bug check, including bad disks, bad or incorrect cables, mixing different ATA-type devices on the same chain, or drives that are not available becuase of hardware regeneration.

This bug check can also be caused by a missing file from the boot partition or by a driver file that a user accidentally disabled in the **Drivers** tab.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6C: REFMON_INITIALIZATION_FAILED

The REFMON_INITIALIZATION_FAILED bug check has a value of 0x0000006C.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6D: SESSION1_INITIALIZATION_FAILED

The SESSION1_INITIALIZATION_FAILED bug check has a value of 0x0000006D. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the initialization failure
2	0
3	0
4	0

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6E: SESSION2_INITIALIZATION_FAILED

The SESSION2_INITIALIZATION_FAILED bug check has a value of 0x0000006E. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the Windows operating system to conclude that initialization failed
2	0
3	0
4	0

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x6F: SESSION3_INITIALIZATION_FAILED

The SESSION3_INITIALIZATION_FAILED bug check has a value of 0x0000006F. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the Windows operating system to conclude that initialization failed
2	0
3	0
4	0

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x70: SESSION4_INITIALIZATION_FAILED

The SESSION4_INITIALIZATION_FAILED bug check has a value of 0x00000070. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the Windows operating system to conclude that initialization failed
2	0
3	0
4	0

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x71: SESSION5_INITIALIZATION_FAILED

The SESSION5_INITIALIZATION_FAILED bug check has a value of 0x00000071. This bug check indicates that the initialization of the Microsoft Windows operating system failed.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The NT status code that caused the Windows operating system to conclude that initialization failed
2	0
3	0
4	0

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x72: ASSIGN_DRIVE_LETTERS_FAILED

The ASSIGN_DRIVE_LETTERS_FAILED bug check has a value of 0x00000072.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x73: CONFIG_LIST_FAILED

The CONFIG_LIST_FAILED bug check has a value of 0x00000073. This bug check indicates that one of the top-level registry keys, also known as core system hives, cannot be linked in the registry tree.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	1
2	The NT status code that led the Windows operating system to assume that it failed to load the hive
3	The index of the hive in the hive list
4	A pointer to a UNICODE_STRING structure that contains the file name of the hive

Cause

The registry hive that cannot be linked might be SAM, SECURITY, SOFTWARE, or DEFAULT. The hive is valid, because it was loaded successfully.

Examine Parameter 2 to see why the hive could not be linked in the registry tree. One common cause of this error is that the Windows operating system is out of disk space on the system drive. (In this situation, this parameter is 0xC000017D, STATUS_NO_LOG_SPACE.) Another common problem is that an attempt to allocate pool has failed. (In this situation, Parameter 2 is 0xC000009A, STATUS_INSUFFICIENT_RESOURCES.) You must investigate other status codes.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x74: BAD_SYSTEM_CONFIG_INFO

The BAD_SYSTEM_CONFIG_INFO bug check has a value of 0x00000074. This bug check indicates that there is an error in the registry.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	Reserved
2	Reserved
3	Reserved
4	The NT status code (if it is available)

Cause

The BAD_SYSTEM_CONFIG_INFO bug check occurs if the SYSTEM hive is corrupt. However, this corruption is unlikely, because the boot loader, known as NT Loader (NTLDR) in versions of Windows prior to Vista, checks a hive for corruption when it loads the hive.

This bug check can also occur if some critical registry keys and values are missing. Thee keys and values might be missing if a user manually edited the registry.

Resolving the Problem

Try restarting the computer by selecting "last known good configuration" in the boot options.

If the restart does not fix the problem, the registry damage is too extensive. You must reinstall the OS or use the Emergency Repair Disk (ERD) that you previously created by using the Windows Backup tool.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x75: CANNOT_WRITE_CONFIGURATION

The CANNOT_WRITE_CONFIGURATION bug check has a value of 0x00000075. This bug check indicates that the SYSTEM registry hive file cannot be converted to a mapped file.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	1
2	The NT status code that led the Windows operating system to assume that it had failed to convert the hive
3	Reserved
4	Reserved

Cause

The CANNOT_WRITE_CONFIGURATION bug check typically occurs if the system is out of pool and the Windows operating system cannot reopen the hive.

This bug check should almost never occur, because the conversion of the hive file occurs early enough during system initialization so that enough pool should be available.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x76: PROCESS_HAS_LOCKED_PAGES

The PROCESS_HAS_LOCKED_PAGES bug check has a value of 0x00000076. This bug check indicates that a driver failed to release locked pages after an I/O operation, or that it attempted to unlock pages that were already unlocked.

Parameters

The following parameters appear on the blue screen.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of error
0x00	The pointer to the process object	The number of locked pages	The pointer to driver stacks (if they are enabled). Otherwise, this parameter is zero.	The process being terminated has locked memory pages. The driver must unlock any memory that it might have locked in a process, before the process terminates.
0x01	MDL specified by the driver	Current number of locked memory pages in that process	A pointer to driver stacks for that process (if they are enabled). Otherwise, this parameter is zero.	The driver is attempting to unlock process memory pages that are not locked.

Cause

The driver either failed to unlock pages that it locked (parameter 1 value is 0x0), or the driver is attempting to unlock pages that have not been locked or that have already been unlocked (parameter 1 value is 0x1).

Resolving the Problem

If the parameter 1 value is 0x0

First use the **!search** extension on the current process pointer throughout all of physical memory. This extension might find at least one memory descriptor list (MDL) that points to the current process. Next, use **!search** on each MDL that you find to obtain the I/O request packet (IRP) that points to the current process. From this IRP, you can identify which driver is leaking the pages.

Otherwise, you can detect which driver caused the error by editing the registry:

1. In the **\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management** registry key, create or edit the **TrackLockedPages** value, and then set it equal to DWORD 1.
2. Restart the computer.

The system then saves stack traces, so you can easily identify the driver that caused the problem. If the driver causes the same error again, bug check 0xCB (DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS) is issued, and the name of the driver that causes this error is displayed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

If the parameter 1 value is 0x1

Examine the driver source code that locks and unlocks memory, and try to locate an instance where memory is unlocked without first being locked.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x77: KERNEL_STACK_INPAGE_ERROR

The KERNEL_STACK_INPAGE_ERROR bug check has a value of 0x00000077. This bug check indicates that the requested page of kernel data from the paging file could not be read into memory.

Parameters

The four parameters that listed in the message have two possible meanings.

If the first parameter is 0, 1, or 2, the parameters have the following meaning.

Parameter	Description
1	**0:** The page of kernel data was retrieved from page cache. **1:** The page was retrieved from a disk. **2:** The page was retrieved from a disk, the storage stack returned SUCCESS, but **Status.Information** is not equal to PAGE_SIZE.
2	The value that appears in the stack where the signature should be.
3	0
4	The address of the signature on the kernel stack

If the first parameter is any value other than 0, 1, or 2, the parameters have the following meaning.

Parameter	Description
1	The status code
2	The I/O status code
3	The page file number
4	The offset into page file

Cause

If the first parameter is 0 or 1, the stack signature in the kernel stack was not found. This error is probably caused by defective hardware, such as a RAM error.

If the first parameter is 2, the driver stack returned an inconsistent status for the read of the page. For example, the driver stack returned a success status even though it did not read the whole page.

If the first parameter is any value other than 0, 1, or 2, the value of the first parameter is an NTSTATUS error code that the driver stack returns after it tries to retrieve the page of kernel data. You can determine the exact cause of this error from the I/O status code (the second parameter). Some common status codes include the following:

- 0xC000009A, or STATUS_INSUFFICIENT_RESOURCES, indicates a lack of nonpaged pool resources. This status code indicates a driver error in the storage stack. (The storage stack should always be able to retrieve this data, regardless of software resource availability.)
- 0xC000009C, or STATUS_DEVICE_DATA_ERROR, indicates bad blocks (sectors) on the hard disk.
- 0xC000009D, or STATUS_DEVICE_NOT_CONNECTED, indicates defective or loose cabling, termination, or that the controller does not see the hard disk drive.
- 0xC000016A, or STATUS_DISK_OPERATION_FAILED, indicates bad blocks (sectors) on the hard disk.
- 0xC0000185, or STATUS_IO_DEVICE_ERROR, indicates improper termination or defective cabling on SCSI devices or that two devices are trying to use the same IRQ.

These status codes are the most common ones that have specific causes. For more information about other possible status codes that might be returned, see the *Ntstatus.h* file in the Microsoft Windows Driver Kit (WDK).

A virus infection can also cause this bug check.

Resolving the Problem

Resolving a bad block problem: If you can restart the computer after the error, Autochk runs automatically and attempts to map the bad sector to prevent it from being used anymore.

If Autochk does not scan the hard disk for errors, you can manually start the disk scanner. Run **Chkdsk /f /r** on the system partition. You must restart the computer before the disk scan begins. If you cannot start the system because the error, use the Recovery Console and run **Chkdsk /r**.

Warning If your system partition is formatted with the FAT file system, the long file names that the Windows operating system uses might be damaged if you use Scandisk or another MS-DOS-based hard disk tool to verify the integrity of your hard disk drive from MS-DOS. Always use the version of Chkdsk that matches your version of the Windows operating system.

Resolving a defective hardware problem: If the I/O status is 0xC0000185 and the paging file is on an SCSI disk, check the disk cabling and SCSI termination for problems.

Resolving a failing RAM problem: Run the hardware diagnostics that the system manufacturer supplies, especially the memory scanner. For more information about these procedures, see the owner's manual for your computer.

Check that all the adapter cards in the computer are properly seated. Use an ink eraser or an electrical contact treatment, available at electronics supply stores, to ensure adapter card contacts are clean.

Check the System Log in Event Viewer for additional error messages that might help identify the device that is causing the error. You can also disable memory caching of the BIOS to try to resolve this error.

Make sure that the latest Windows Service Pack is installed.

If the preceding steps fail to resolve the error, take the system motherboard to a repair facility for diagnostic testing. A crack, a scratched trace, or a defective component on the motherboard can cause this error.

Resolving a virus infection: Check your computer for viruses by using any up-to-date, commercial virus scanning software that examines the Master Boot Record of the hard disk. All Windows file systems can be infected by viruses.

See Also

Bug Check 0x7A (KERNEL_DATA_INPAGE_ERROR)

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x78: PHASE0_EXCEPTION

The PHASE0_EXCEPTION bug check has a value of 0x00000078.

This bug check occurs when an unexpected break is encountered during HAL initialization. This break can occur if you have set the **/break** parameter in your boot settings but have not enabled kernel debugging.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x79: MISMATCHED_HAL

The MISMATCHED_HAL bug check has a value of 0x00000079. This bug check indicates that the Hardware Abstraction Layer (HAL) revision level or configuration does not match that of the kernel or the computer.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of mismatch.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause.
0x1	The major processor control block (PRCB) level of *Ntoskrnl.exe*.	The major PRCB level of *Hal.dll*.	Reserved	The PRCB release levels are mismatched. (Something is out of date.)
0x2	The build type of *Ntoskrnl.exe*.	The build type of *Hal.dll*.	Reserved	The build types are mismatched.
0x3	The size of the loader parameter extension.	The major version of the loader parameter extension.	The minor version of the loader parameter extension.	The loader (*ntldr*) and HAL versions are mismatched.

When Parameter 1 equals 0x2, the following build type codes are used:

- 0: Multiprocessor-enabled free build
- 1: Multiprocessor-enabled checked build
- 2: Single-processor free build
- 3: Single-processor checked build

Cause

The MISMATCHED_HAL bug check often occurs when a user manually updates *Ntoskrnl.exe* or *Hal.dll*.

The error can also indicate that one of those two files is out of date. For example, the HAL might be designed for Microsoft Windows 2000 and the kernel is designed for Windows XP. Or the computer might erroneously have a multiprocessor HAL and a single-processor kernel installed, or vice versa.

The *Ntoskrnl.exe* kernel file is for single-processor systems and *Ntkrnlmp.exe* is for multiprocessor systems. However, these file names correspond to the files on the installation media.After you have installed the Windows operating system, the file is renamed to *Ntoskrnl.exe*, regardless of the source file that is used. The HAL file also uses the name *Hal.dll* after installation, but there are several possible HAL files on the installation media. For more information, see "Installing the Checked Build" in the Windows Driver Kit (WDK).

Resolving the Problem

Restart the computer by using the product CD or the Windows Setup disks. At the Welcome screen, press F10 to start the Recovery Console. Use the **Copy** command to copy the correct HAL or kernel file from the original CD into the appropriate folder on the hard disk. The **Copy** command detects whether the file that you are copying is in the Microsoft compressed file format. If so, it automatically expands the file that is copied on the target drive.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7A: KERNEL_DATA_INPAGE_ERROR

The KERNEL_DATA_INPAGE_ERROR bug check has a value of 0x0000007A. This bug check indicates that the requested page of kernel data from the paging file could not be read into memory.

Parameters

The four parameters that are listed in the message can have three possible meanings. If the first parameter is 1 or 2, or 3 and the third parameter is 0, the parameters have the following definitions.

Parameter	Description
1	The lock type that was held (1, 2, or 3)
2	The error status (usually an I/O status code)
3	**If Lock Type is 1:** the current process **If Lock Type is 2 or 3:** 0
4	The virtual address that could not be paged into memory

If the first parameter is 3 (and the third parameter is nonzero) or 4, the parameters have the following definitions.

Parameter	Description
1	The lock type that was held (3 or 4)
2	The error status (typically an I/O status code)
3	The address of the InPageSupport structure
4	The faulting address

Otherwise, the parameters have the following definitions.

Parameter	Description
1	The address of the page table entry (PTE)
2	The error status (usually an I/O status code)
3	The PTE contents

4	The faulting address

Cause

Frequently, you can determine the cause of the KERNEL_DATA_INPAGE_ERROR bug check from the error status (Parameter 2). Some common status codes include the following:

- 0xC000009A, or STATUS_INSUFFICIENT_RESOURCES, indicates a lack of nonpaged pool resources.
- 0xC000009C, or STATUS_DEVICE_DATA_ERROR, typically indicates bad blocks (sectors) on the hard disk.
- 0xC000009D, or STATUS_DEVICE_NOT_CONNECTED, indicates defective or loose cabling, termination, or that the controller does not see the hard disk.
- 0xC000016A, or STATUS_DISK_OPERATION_FAILED, indicates bad blocks (sectors) on the hard disk.
- 0xC0000185, or STATUS_IO_DEVICE_ERROR, indicates improper termination or defective cabling on SCSI devices or that two devices are trying to use the same IRQ.

These status codes are the most common ones that have specific causes. For more information about other possible status codes that can be returned, see the *Ntstatus.h* file in the Microsoft Windows Driver Kit (WDK).

Another common cause of this error message is defective hardware or failing RAM.

A virus infection can also cause this bug check.

Resolving the Problem

Resolving a bad block problem: An I/O status code of 0xC000009C or 0xC000016A typically indicates that the data could not be read from the disk because of a bad block (sector). If you can restart the computer after the error, Autochk runs automatically and attempts to map the bad sector to prevent it from being used anymore.

If Autochk does not scan the hard disk for errors, you can manually start the disk scanner. Run **Chkdsk /f /r** on the system partition. You must restart the computer before the disk scan begins. If you cannot start the computer because of the error, use the Recovery Console and run **Chkdsk /r**.

Warning If your system partition is formatted with the FAT file system, the long file names that the Windows operating system uses might be damaged if you use Scandisk or another MS-DOS-based hard disk tool to verify the integrity of your hard disk from MS-DOS. Always use the version of Chkdsk that matches your version of Windows.

Resolving a defective hardware problem: If the I/O status is C0000185 and the paging file is on an SCSI disk, check the disk cabling and SCSI termination for problems.

Resolving a failing RAM problem: Run the hardware diagnostics that the system manufacturer supplies, especially the memory scanner. For more information about these procedures, see the owner's manual for your computer.

Check that all the adapter cards in the computer are properly seated. Use an ink eraser or an electrical contact treatment, available at electronics supply stores, to ensure adapter card contacts are clean.

Check the System Log in Event Viewer for additional error messages that might help identify the device that is causing the error. You can also disable memory caching of the BIOS to try to resolve this error.

Make sure that the latest Windows Service Pack is installed.

If the preceding steps do not resolve the error, take the system motherboard to a repair facility for diagnostic testing. A crack, a scratched trace, or a defective component on the motherboard can cause this error.

Resolving a virus infection: Check your computer for viruses by using any up-to-date, commercial virus scanning software that examines the Master Boot Record of the hard disk. All Windows file systems can be infected by viruses.

See Also

Bug Check 0x77 (KERNEL_STACK_INPAGE_ERROR)

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7B: INACCESSIBLE_BOOT_DEVICE

The INACCESSIBLE_BOOT_DEVICE bug check has a value of 0x0000007B. This bug check indicates that the Microsoft Windows operating system has lost access to the system partition during startup.

Parameters

The following parameters appear in the message.

Parameter	Description
1	The address of a UNICODE_STRING structure, or the address of the device object that could not be mounted
2	0
3	0
4	0

To determine the meaning of Parameter 1, look at the data that it points to. If the first word (USHORT) at this address is even, Parameter 1 is the beginning of a Unicode string. If the first word (USHORT) at this address is 0x3, Parameter 1 is the first field (Type) of a device object.

- If this parameter points to a device object, the file system that was supposed to read the boot device failed to initialize or simply did not recognize the data on the boot device as a file system structure. In this situation, the specified device object is the object that could not be mounted.
- If this parameter points to a Unicode string, you must read the first 8 bytes at this address. These bytes form the UNICODE_STRING structure, which is defined as follows:

```
USHORT Length;
USHORT MaximumLength;
PWSTR Buffer;
```

The **Length** field gives the actual length of the string. The **Buffer** field points to the beginning of the string (**Buffer** is always be at least 0x80000000.)

The actual string contains the Advanced RISC Computing (ARC) specification name of the device that the boot was being attempted from. ARC names are a generic way to identify devices in the ARC environment.

Cause

The INACCESSIBLE_BOOT_DEVICE bug check frequently occurs because of a boot device failure. During I/O system initialization, the boot device driver might have failed to initialize the boot device (typically a hard disk). File system initialization might have failed because it did not recognize the data on the boot device. Also, repartitioning the system partition or installing a new SCSI adapter or disk controller might induce this error.

This error can also occur because of incompatible disk hardware. If the error occurred at the initial setup of the system, the system might have been installed on an unsupported disk or SCSI controller. Some controllers are supported only by drivers that are in the Windows Driver Library (WDL). (These drivers require the user to do a custom installation.)

Resolving the Problem

This error always occurs while the system is starting. This error frequently occurs before the debugger connection is established, so debugging can be difficult or impossible.

Resolving a failed boot device problem: If a boot device is at fault, you must edit the boot options.For more information about changing these options, see Configuring Software on the Target Computer .

Resolving an incompatible disk hardware problem: If Setup autodetects the controller, you might have to skip detection and use a specific manufacturer's disk to load the driver. Also, check the availability of updates for the system BIOS and SCSI controller firmware. Updates of this kind are typically available on the Web site or BBS of the hardware manufacturer.

Remove any recently added hardware, especially hard disk drives or controllers, to see if the error is resolved. If the problematic hardware is a hard disk drive, the disk firmware version might be incompatible with your version of the Windows operating system. Contact the manufacturer for updates. If you removed another piece of hardware and the error is resolved, IRQ or I/O port conflicts likely exist. Reconfigure the new device according to the manufacturer's instructions.

Confirm that all hard disk drivers, hard disk controllers, and SCSI adapters are listed in the Microsoft Windows Marketplace Tested Products List.

If you recently added a driver, restart your computer, and press F8 at the character-based menu that displays the operating system choices. In the **Advanced Options** menu, select the **Last Known Good Configuration** option. This option is most effective when you add only one driver or service at a time.

In addition, check your computer for viruses by using any up-to-date, commercial virus scanning software that examines the Master Boot Record of the hard disk. All Windows file systems can be infected by viruses.

This error can also occur because of hard disk corruption. Run **Chkdsk /f /r** on the system partition. You must restart the computer before the disk scan begins. If you cannot start the computer because of the error, use the Recovery Console and run **Chkdsk /r**.

If you cannot start the system in the last known good configuration, you should try to start off the Windows CD. Then, you can run **Chkdsk** from the Repair Console.

Warning If your system partition is formatted with the FAT file system, the long file names that the Windows operating system uses might be damaged if you use Scandisk or another MS-DOS-based hard disk tool to verify the integrity of your hard disk drive from MS-DOS. Always use the version of Chkdsk that matches your version of Windows.

If your system has SCSI adapters, contact the adapter manufacturer to obtain updated Windows drivers. Try disabling sync negotiation in the SCSI BIOS, checking the cabling and the SCSI IDs of each device, and confirming proper termination. For IDE devices, define the onboard IDE port as Primary only. Also check each IDE device for the proper **master/subordinate/stand alone** setting. Try removing all IDE devices except for hard disks. Finally, check the System Log in Event Viewer for additional error messages that might help identify the device or driver that is causing the error.

To analyze this error: Run an **lm (List Loaded Modules)** command in the debugger. Verify that the following drivers were loaded: *disk*, *classpnp*, *ftdisk*, *partmgr*, and *FAT* or *NTFS*.

```
kd> lm

start    end        module name
80001000 80016000   hal          (deferred)
80016000 80018c40   bootvid      (deferred)
80019000 8001dfc0   pciidex      (deferred)
8001e000 8001ff60   dmload       (deferred)
80086000 80086980   pciide       (deferred)
802c1000 802cc000   pci          (deferred)
802cc000 802d39a0   isapnp       (deferred)
802d4000 802ed000   ftdisk       (deferred)
802ed000 802f3820   mountmgr     (deferred)
802f4000 802fad40   fdc          (deferred)
802fb000 802fdc20   partmgr      (deferred)
802fe000 802fef00   wmilib       (deferred)
8039b000 803b8000   dmio         (deferred)
803b8000 803cb000   atapi        (deferred)
803cb000 803d1560   disk         (deferred)
803d2000 803d8e80   classpnp     (deferred)
```

```
803d9000 803fa000   fastfat      (deferred)
80400000 80540000   nt           (pdb symbols)   \\localsymbols\symbols\exe\ntoskrnl.dbg
80540000 80546f20   ksecdd       (deferred)
80547000 80554620   cnss         (deferred)
80555000 80579000   ndis         (deferred)
```

You probably have *pci* or *isapnp* loaded. Also make sure your controller drivers are loaded. That is, make sure *Atapi.sys* is loaded with the channel drivers (*pciide* and *pciidex* or *intelid*) or *scsiport.sys* is loaded with the appropriate miniport driver.

It is helpful to know as much as possible about the boot device that Windows is installed on. For example, you can investigate the following items:

- Find out what type of controller the boot device is connected to (SCSI, IDE, 1394, etc). Find the manufacturer of non-IDE controllers (Adaptec, Symbios, and so on).
- Note the SCSI ID of the boot device if you are using SCSI.
- Indicate if other devices are attached to the same controller that the boot device is on (CD-ROM drives, zip drives, and so on).
- Note the file system that is used on the drive.

The **!devnode** extension gives you more information, if you know what your boot devices are.

Typically Plug and Play cannot assign resources to the boot device. You can verify this restriction by finding an entry for the service. If the status flags include DNF_INSUFFICIENT_RESOURCES or do not include DNF_STARTED or DNF_ENUMERATED, you have found the problem. Try **!devnode 0 1 scsi** or **!devnode 0 1 atapi** to save some time instead of dumping the whole device tree.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7C: BUGCODE_NDIS_DRIVER

The BUGCODE_NDIS_DRIVER bug check has a value of 0x0000007C. This bug check indicates that a problem occurred with an NDIS driver.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x01	The address of the miniport block	The number of bytes that are requested	The current IRQL	A driver called **NdisMAllocateSharedMemory** at a raised IRQL.
0x02	The address of the miniport block	The shared memory page that was corrupted	The address of NDIS_WRAPPER_CONTEXT that keeps track of the driver's shared memory allocations	During a call to **NdisMAllocateSharedMemory**, NDIS detected that a previously-allocated shared memory page had been corrupted.
0x03	The address of the miniport block	The page that contains the shared memory	The virtual address of the shared memory	A driver called **NdisMFreeSharedMemory**[**Async**] with a shared memory pointer that had already been freed.
0x04	The address of NDIS_M_DRIVER_BLOCK	The address of DRIVER_OBJECT	0	*AddDevice* was called with a driver that is not on the list of drivers that are registered with NDIS. *(Enabled only on special instrumented NDIS.)*
0x05 0x06	The address of the miniport block	The address of the packet descriptor that the driver uses	The address of the packet array that contained this packet descriptor	An Ethernet driver indicated that it received a packet by using a packet descriptor that the protocol stack is currently using.
0x07	The address of the miniport block	The address of the packet descriptor that the driver uses	The address of the packet array that contained this packet descriptor	An FDDI driver indicated that it received a packet by using a packet descriptor that the protocol stack is currently using.
0x08	The address of the miniport block	The address of NDIS_MINIPORT_INTERRUPT	0	A miniport driver did not deregister its interrupt during the halt process.
0x09	The address of the miniport block	The address of the miniport driver's timer queue (NDIS_MINIPORT_TIMER)	0	A miniport driver stopped without successfully canceling all its timers.
0x0A	The address of NDIS_M_DRIVER_BLOCK	The address of DRIVER_OBJECT	The reference count for the miniport driver	A miniport driver is getting unloaded prematurely.
0x0B	The address of the miniport block	The address of NDIS_MINIPORT_INTERRUPT	0	A miniport driver failed its initialization without deregistering its interrupt.
0x0C	The address of the miniport block	The address of the miniport driver's timer queue (NDIS_MINIPORT_TIMER)	0	A miniport driver failed its initialization without successfully canceling all its timers.
0x0D	The address of the miniport block	The address of NDIS_MINIPORT_INTERRUPT	0	A miniport driver did not deregister its interrupt during the halt process. (The halt was called from the initialize routine after the miniport driver returned success from its initialize handler.)
0x0E	The address of the miniport block	The address of the miniport driver's timer queue (NDIS_MINIPORT_TIMER)	0	A miniport driver stopped without successfully canceling all its timers. (The halt was called from the initialize routine after the miniport driver returned success from its initialize handler.)

0x0F	The address of the miniport block	The reset status	AddressingReset (BOOLEAN)	A miniport driver called **NdisMResetComplete** without any pending reset request.
0x10	The address of the miniport block	The address of NDIS_MINIPORT_INTERRUPT	0	After resuming from a low-power state, a miniport driver failed its initialization without deregistering its interrupt.
0x11	The address of the miniport block	The address of the miniport driver's timer queue (NDIS_MINIPORT_TIMER)	0	After resuming from a low-power state, a miniport driver failed its initialization without successfully canceling all its timers.
0x12	The address of the miniport block	The address of the packet descriptor that the driver uses	The address of the packet array that contained this packet descriptor	A miniport driver indicated that it received a packet by using a packet descriptor that the protocol stack is currently using.
0x13	The address of the miniport block	The address of the packet descriptor that the driver uses	The address of the packet array that contained this packet descriptor	A Token-Ring miniport driver indicated that it received a packet by using a packet descriptor that the protocol stack currently uses.
0x14	The current IRQL value	0	0	An NDIS driver called NdisWaitEvent at IRQL > PASSIVE_LEVEL. The function must be called at IRQL = PASSIVE_LEVEL.
0x15	The address of the miniport block	0	0	An NDIS 6 miniport driver was calling an NDIS 5 API. An NDIS 6 miniport driver cannot call NdisMQueryInformationComplete or NdisMSetInformationComplete.
0x16	The address of the protocol block	The address of the context area that is allocated by the protocol driver	The address of the open block	NDIS encountered an invalid handle in a binding operation. A protocol driver's **ProtocolBindAdapterEx** function returned NDIS_STATUS_SUCCESS, either directly or asynchronously through **NdisCompleteBindAdapterEx**. However, the binding context information contains an invalid handle to a block that indicates the open state of the miniport adapter. In this case, the open handle is not NULL, but it cannot be referenced.
0x17	The address of the interface provider block	0	0	The NDIS driver was attempting to deregister as a network interface provider while an interface was still registered.

Cause

Parameter 1 indicates the specific cause of the BUGCODE_NDIS_DRIVER bug check.

If one of the bug check parameters specifies the address of the miniport block, you can obtain more information by using **!ndiskd.miniport** together with this address.

If one of the bug check parameters specifies the address of the packet descriptor that the driver uses, you can obtain more information by using **!ndiskd.pkt** together with this address.

Comments

This bug check code occurs only on Microsoft Windows Server 2003 and later versions of Windows. In Windows 2000 and Windows XP, the corresponding code is bug check 0xD2 (BUGCODE_ID_DRIVER).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7D: INSTALL_MORE_MEMORY

The INSTALL_MORE_MEMORY bug check has a value of 0x0000007D. This bug check indicates that there is not enough memory to start up the Microsoft Windows operating system.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The number of physical pages that are found
2	The lowest physical page
3	The highest physical page
4	0

Cause

The Windows operating system does not have sufficient memory to complete the startup process.

Resolving the Problem

Install more memory.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7E: SYSTEM_THREAD_EXCEPTION_NOT_HANDLED

The SYSTEM_THREAD_EXCEPTION_NOT_HANDLED bug check has a value of 0x0000007E. This bug check indicates that a system thread generated an exception that the error handler did not catch.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The exception code that was not handled
2	The address where the exception occurred
3	The address of the exception record
4	The address of the context record

Cause

The SYSTEM_THREAD_EXCEPTION_NOT_HANDLED bug check is a very common bug check. To interpret it, you must identify which exception was generated.

Common exception codes include the follwoing:

- 0x80000002: STATUS_DATATYPE_MISALIGNMENT indicates an unaligned data reference was encountered.
- 0x80000003: STATUS_BREAKPOINT indicates a breakpoint or ASSERT was encountered when no kernel debugger was attached to the system.
- 0xC0000005: STATUS_ACCESS_VIOLATION indicates a memory access violation occurred.

For a complete list of exception codes, see the *Ntstatus.h* file that is located in the *inc* directory of the Microsoft Windows Driver Kit (WDK).

Resolving the Problem

If you are not equipped to debug this problem, you should use some basic troubleshooting techniques.

- Make sure you have enough disk space.
- If a driver is identified in the bug check message, disable the driver or check with the manufacturer for driver updates.
- Try changing video adapters.
- Check with your hardware vendor for any BIOS updates.
- Disable BIOS memory options such as caching or shadowing.

If you plan to debug this problem, you might find it difficult to obtain a stack trace. Parameter 2 (the exception address) should identify the driver or function that caused this problem.

If exception code 0x80000003 occurs, a hard-coded breakpoint or assertion was hit, but the system was started with the **/NODEBUG** switch. This problem should rarely occur. If it occurs repeatedly, make sure that a kernel debugger is connected and the system is started with the **/DEBUG** switch.

If exception code 0x80000002 occurs, the trap frame supplies additional information.

If you do not know the specific cause of the exception, consider the following issues:

- Hardware incompatibility. Make sure that any new hardware that is installed is listed in the Microsoft Windows Marketplace Tested Products List.
- Faulty device driver or system service. A faulty device driver or system service might be responsible for this error. Hardware issues, such as BIOS incompatibilities, memory conflicts, and IRQ conflicts can also generate this error.

If a driver is listed by name within the bug check message, disable or remove that driver. Disable or remove any drivers or services that were recently added. If the error occurs during the startup sequence and the system partition is formatted with NTFS file system, you might be able to use Safe Mode to rename or delete the faulty driver. If the driver is used as part of the system startup process in Safe Mode, you must start the computer by using the Recovery Console to access the file.

If the problem is associated with *Win32k.sys*, the source of the error might be a third-party remote control program. If such software is installed, you can remove the service by starting the computer by using the Recovery Console and then deleting the offending system service file.

Check the System Log in **Event Viewer** for additional error messages that might help identify the device or driver that is causing bug check 0x7E.

You can also disable memory caching of the BIOS might to try to resolve the error. You should also run hardware diagnostics, especially the memory scanner, that the system manufacturer supplies. For more information about these procedures, see the owner's manual for your computer.

The error that generates this message can occur after the first restart during Windows Setup, or after Setup is finished. A possible cause of the error is lack of disk space for installation and system BIOS incompatibilities. For problems during Windows installation that are associated with lack of disk space, reduce the number of files on the target hard disk drive. Check for and delete any temporary files that you do not have to have, Internet cache files, application backup files, and *.chk* files that contain saved file fragments from disk scans. You can also use another hard disk drive with more free space for the installation. You can resolve BIOS problems by upgrading the system BIOS version.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x7F: UNEXPECTED_KERNEL_MODE_TRAP

The UNEXPECTED_KERNEL_MODE_TRAP bug check has a value of 0x0000007F. This bug check indicates that the Intel CPU generated a trap and the kernel failed to catch this trap.

This trap could be a *bound trap* (a trap the kernel is not permitted to catch) or a *double fault* (a fault that occurred while processing an earlier fault, which always results in a system failure).

Parameters

The first parameter that appears on the blue screen specifies the trap number.

The most common trap codes include the following:

- 0x00000000, or Divide by Zero Error, indicates that a DIV instruction is executed and the divisor is zero. Memory corruption, other hardware problems, or software failures can cause this error.
- 0x00000004, or Overflow, occurs when the processor executes a call to an interrupt handler when the overflow (OF) flag is set.
- 0x00000005, or Bounds Check Fault, indicates that the processor, while executing a BOUND instruction, finds that the operand exceeds the specified limits. A BOUND instruction ensures that a signed array index is within a certain range.
- 0x00000006, or Invalid Opcode, indicates that the processor tries to execute an invalid instruction. This error typically occurs when the instruction pointer has become corrupted and is pointing to the wrong location. The most common cause of this error is hardware memory corruption.
- 0x00000008, or Double Fault, indicates that an exception occurs during a call to the handler for a prior exception. Typically, the two exceptions are handled serially. However, there are several exceptions that cannot be handled serially, and in this situation the processor signals a double fault. There are two common causes of a double fault:
  - A kernel stack overflow. This overflow occurs when a guard page is hit, and the kernel tries to push a trap frame. Because there is no stack left, a stack overflow results, causing the double fault. If you think this overview has occurred, use **!thread** to determine the stack limits, and then use **kb (Display Stack Backtrace)** with a large parameter (for example, **kb 100**) to display the full stack.
  - A hardware problem.

The less-common trap codes include the following:

- 0x00000001 — A system-debugger call
- 0x00000003 — A debugger breakpoint
- 0x00000007 — A hardware coprocessor instruction with no coprocessor present
- 0x0000000A — A corrupted Task State Segment
- 0x0000000B — An access to a memory segment that was not present
- 0x0000000C — An access to memory beyond the limits of a stack
- 0x0000000D — An exception not covered by some other exception; a protection fault that pertains to access violations for applications

For other trap numbers, see an Intel architecture manual.

Cause

Bug check 0x7F typically occurs after you install a faulty or mismatched hardware (especially memory) or if installed hardware fails.

A double fault can occur when the kernel stack overflows. This overflow occurs if multiple drivers are attached to the same stack. For example, if two file system filter drivers are attached to the same stack and then the file system recurses back in, the stack overflows.

Resolving the Problem

Debugging: Always begin with the **!analyze** extension.

If this extension is not sufficient, use the **kv (Display Stack Backtrace)** debugger command.

- If **kv** shows a **taskGate**, use the **.tss (Display Task State Segment)** command on the part before the colon.
- If **kv** shows a trap frame, use the **.trap (Display Trap Frame)** command to format the frame.
- Otherwise, use the **.trap (Display Trap Frame)** command on the appropriate frame. (On x86-based platforms, this frame is associated with the procedure **NT!KiTrap**.)

After using one of these commands, use **kv** again to display the new stack.

Troubleshooting: If you recently added hardware to the computer, remove it to see if the error recurs. If existing hardware has failed, remove or replace the faulty component. Run hardware diagnostics that the system manufacturer supplies to determine which hardware component failed.

The memory scanner is especially important. Faulty or mismatched memory can cause this bug check. For more informaiton about these procedures, see the owner's manual for your computer. Check that all adapter cards in the computer are properly seated. Use an ink eraser or an electrical contact treatment, available at electronics supply stores, to ensure adapter card contacts are clean.

If the error appears on a newly installed system, check the availability of updates for the BIOS, the SCSI controller, or network cards. These kind of updates are typically available on the Web site or BBS of the hardware manufacturer.

Confirm that all hard disk drives, hard disk controllers, and SCSI adapters are listed in the Microsoft Windows Marketplace Tested Products List.

If the error occurred after the installation of a new or updated device driver, you should remove or replace the driver. If, under this circumstance, the error occurs during the startup sequence and the system partition is formatted with NTFS, you might be able to use Safe Mode to rename or delete the faulty driver. If the driver is used as part of the system startup process in Safe Mode, you have to start the computer by using the Recovery Console in order to access the file.

Also restart your computer, and then press F8 at the character-based menu that displays the operating system choices. At the **Advanced Options** menu, select the **Last Known Good Configuration** option. This option is most effective when you add only one driver or service at a time.

Overclocking (setting the CPU to run at speeds above the rated specification) can cause this error. If you have overclocked the computer that is experiencing the error, return the CPU to the default clock speed setting.

Check the System Log in Event Viewer for additional error messages that might help identify the device or driver that is causing the error. You can also disable memory caching of the BIOS to try to resolve the problem.

If you encountered this error while upgrading to a new version of the Windows operating system, the error might be caused by a device driver, a system service, a virus scanner, or a backup tool that is incompatible with the new version. If possible, remove all third-party device drivers and system services and disable any virus scanners before you upgrade. Contact the software manufacturer to obtain updates of these tools. Also make sure that you have installed the latest Windows Service Pack.

Finally, if all the above steps do not resolve the error, take the system motherboard to a repair facility for diagnostic testing. A crack, a scratched trace, or a defective component on the motherboard can also cause this error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x80: NMI_HARDWARE_FAILURE

The NMI_HARDWARE_FAILURE bug check has a value of 0x00000080. This bug check indicates that a hardware malfunction has occurred.

Parameters

None

Cause

A variety of hardware malfunctions can cause the NMI_HARDWARE_FAILURE bug check. The exact cause is difficult to determine.

Resolving the Problem

Remove any hardware or drivers that have been recently installed. Make sure that all memory modules are of the same type.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x81: SPIN_LOCK_INIT_FAILURE

The SPIN_LOCK_INIT_FAILURE bug check has a value of 0x00000081.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x82: DFS_FILE_SYSTEM

The DFS_FILE_SYSTEM bug check has a value of 0x00000082.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x85: SETUP_FAILURE

The SETUP_FAILURE bug check has a value of 0x00000085. This bug check indicates that a fatal error occurred during setup.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation. Parameter 4 is not used. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Cause
0x0	0	0	The OEM HAL font is not a valid *.fon* format file, so setup cannot display text. This cause indicates that *Vgaxxx.fon* on the boot floppy or CD is damaged.
0x1	The precise video initialization failure: **0: NtCreateFile** of \device\video0 **1:** IOCTL_VIDEO_QUERY_NUM_AVAIL_MODES **2:** IOCTL_VIDEO_QUERY_AVAIL_MODES **3:** The desired video mode is not supported. This value indicates an internal setup error. **4:** IOCTL_VIDEO_SET_CURRENT_MODE (unable to set video mode) **5:** IOCTL_VIDEO_MAP_VIDEO_MEMORY **6:** IOCTL_VIDEO_LOAD_AND_SET_FONT	The status code from the NT API call, if appropriate	Video initialization failed. This failure might indicate that the disk that contains *Vga.sys* (or another video driver that is appropriate to the computer) is damaged or that the computer has video hardware that the Microsoft Windows operating system cannot communicate with.
0x2	0	0	Out of memory.
0x3	The precise keyboard initialization failure: **0: NtCreateFile** of \device\KeyboardClass0 failed. (Setup did not find a keyboard connected to the computer.) **1:** Unable to load keyboard layout DLL. (Setup could not load the keyboard layout file. This failure indicates that the CD or floppy disk is missing a file, such as *Kbdus.dll* for the U.S. release or another layout DLL for localized releases.)	0	Keyboard initialization failed. This failure might indicate that the disk that contains the keyboard driver (*I8042prt.sys* or *Kbdclass.sys*) is damaged or that the computer has keyboard hardware that Windows cannot communicate with. This failure might also mean that the keyboard layout DLL could not be loaded.
0x4	0	0	Setup could not resolve the ARC device path name of the device that setup was started from. This error is an internal setup error.
0x5	Reserved	Reserved	Partitioning sanity check failed. This error indicates a bug in a disk driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x8B: MBR_CHECKSUM_MISMATCH

The MBR_CHECKSUM_MISMATCH bug check has a value of 0x0000008B. This bug check indicates that a mismatch has occurred in the MBR checksum.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The disk signature from MBR
2	The MBR checksum that the OS Loader calculates
3	The MBR checksum that the system calculates
4	Reserved

Cause

The MBR_CHECKSUM_MISMATCH bug check occurs during the boot process when the MBR checksum that the Microsoft Windows operating system calculates does not

match the checksum that the loader passes in.

This error typically indicates a virus.

Resolving the Problem

There are many forms of viruses and not all can be detected. Typically, the newer viruses usually can be detected only by a virus scanner that has recently been upgraded. You should boot with a write-protected disk that contains a virus scanner and try to clean out the infection.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x8E: KERNEL_MODE_EXCEPTION_NOT_HANDLED

The KERNEL_MODE_EXCEPTION_NOT_HANDLED bug check has a value of 0x0000008E. This bug check indicates that a kernel-mode application generated an exception that the error handler did not catch.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The exception code that was not handled
2	The address where the exception occurred
3	The trap frame
4	Reserved

Cause

The KERNEL_MODE_EXCEPTION_NOT_HANDLED bug check is a very common bug check. To interpret it, you must identify which exception was generated.

Common exception codes include the following:

- 0x80000002: STATUS_DATATYPE_MISALIGNMENT indicates that an unaligned data reference was encountered.
- 0x80000003: STATUS_BREAKPOINT indicates that a breakpoint or ASSERT was encountered when no kernel debugger was attached to the system.
- 0xC0000005: STATUS_ACCESS_VIOLATION indicates that a memory access violation occurred.

For a complete list of exception codes, see the *Ntstatus.h* file that is located in the *inc* directory of the Microsoft Windows Driver Kit (WDK).

Resolving the Problem

If you are not equipped to debug this problem, you should use some basic troubleshooting techniques:

- Make sure you have enough disk space.
- If a driver is identified in the bug check message, disable the driver or check with the manufacturer for driver updates.
- Try changing video adapters.
- Check with your hardware vendor for any BIOS updates.
- Disable BIOS memory options such as caching or shadowing.

If you plan to debug this problem, you might find it difficult to obtain a stack trace. Parameter 2 (the exception address) should identify the driver or function that caused this problem.

If exception code 0x80000003 occurs, a hard-coded breakpoint or assertion was hit, but the computer was started with the **/NODEBUG** switch. This problem should rarely occur. If it occurs repeatedly, make sure that a kernel debugger is connected and that the computer is started with the **/DEBUG** switch.

If exception code 0x80000002 occurs, the trap frame supplies additional information.

If you do not know the specific cause of the exception, consider the following items:

- Hardware incompatibility. Make sure that any new hardware installed is listed in the Microsoft Windows Marketplace Tested Products List.
- Faulty device driver or system service. A faulty device driver or system service might be responsible for this error. Hardware issues, such as BIOS incompatibilities, memory conflicts, and IRQ conflicts can also generate this error.

If the bug check message lists a driver by name , disable or remove that driver. Also, disable or remove any drivers or services that were recently added. If the error occurs during the startup sequence and the system partition is formatted with NTFS file system, you might be able to use Safe Mode to rename or delete the faulty driver. If the driver is used as part of the system startup process in Safe Mode, you have to start the computer by using the Recovery Console to access the file.

If the problem is associated with *Win32k.sys*, the source of the error might be a third-party remote control program. If such software is installed, you can remove the service by starting the system by using the Recovery Console and then deleting the offending system service file.

Check the System Log in **Event Viewer** for additional error messages that might help identify the device or driver that is causing bug check 0x1E. You can disable memory caching of the BIOS to try to resolve the error. You should also run hardware diagnostics, especially the memory scanner, that the system manufacturer supplies. For more information about these procedures, see the owner's manual for your computer.

The error that generates this message can occur after the first restart during Windows Setup, or after Setup is finished. A possible cause of the error is lack of disk space for installation and system BIOS incompatibilities. For problems during Windows installation that are associated with lack of disk space, reduce the number of files on the target

hard disk drive. Check for and delete any temporary files that you do not have to have, Internet cache files, application backup files, and *.chk* files that contain saved file fragments from disk scans. You can also use another hard disk drive with more free space for the installation.

You can resolve BIOS problems by upgrading the system BIOS version.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x8F: PP0_INITIALIZATION_FAILED

The PP0_INITIALIZATION_FAILED bug check has a value of 0x0000008F. This bug check indicates that the Plug and Play (PnP) manager could not be initialized.

Parameters

None

Cause

An error occurred during Phase 0 initialization of the kernel-mode PnP manager.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x90: PP1_INITIALIZATION_FAILED

The PP1_INITIALIZATION_FAILED bug check has a value of 0x00000090. This bug check indicates that the Plug and Play (PnP) manager could not be initialized.

Parameters

None

Cause

An error occurred during Phase 1 initialization of the kernel-mode PnP manager.

Phase 1 is where most of the initialization is done, including setting up the registry files and other environment settings for drivers to call during the subsequent I/O initialization.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x92: UP_DRIVER_ON_MP_SYSTEM

The UP_DRIVER_ON_MP_SYSTEM bug check has a value of 0x00000092. This bug check indicates that a uniprocessor-only driver has been loaded on a multiprocessor system.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The base address of the driver
2	Reserved
3	Reserved
4	Reserved

Cause

A driver that is compiled to work only on uniprocessor machines has been loaded, but the Microsoft Windows operating system is running on a multiprocessor system with more than one active processor.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x93: INVALID_KERNEL_HANDLE

The INVALID_KERNEL_HANDLE bug check has a value of 0x00000093. This bug check indicates that an invalid or protected handle was passed to **NtClose**.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The handle that is passed to **NtClose**
2	**0:** The caller tried to close a protected handle **1:** The caller tried to close an invalid handle
3	Reserved
4	Reserved

Cause

The INVALID_KERNEL_HANDLE bug check indicates that some kernel code (for example, a server, redirector, or another driver) tried to close an invalid handle or a protected handle.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x94: KERNEL_STACK_LOCKED_AT_EXIT

The KERNEL_STACK_LOCKED_AT_EXIT bug check has a value of 0x00000094. This bug check indicates that a thread exited while its kernel stack was marked as not swappable

Parameters

None

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x96: INVALID_WORK_QUEUE_ITEM

The INVALID_WORK_QUEUE_ITEM bug check has a value of 0x00000096. This bug check indicates that a queue entry was removed that contained a null pointer.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The address of the queue entry whose **flink** or **blink** field is NULL.
2	The address of the queue that is being referenced. Typically, this queue is an **ExWorkerQueue**.
3	The base address of the **ExWorkerQueue** array. (This address helps you determine if the queue in question is indeed an **ExWorkerQueue**. If the queue is an **ExWorkerQueue**, the offset from this parameter will isolate the queue.)
4	Assuming the queue is an **ExWorkerQueue**, this value is the address of the worker routine that would have been called if the work item had been valid. (You can use this address to isolate the driver that is misusing the work queue.)

Cause

The INVALID_WORK_QUEUE_ITEM bug check occurs when **KeRemoveQueue** removes a queue entry whose **flink** or **blink** field is NULL.

Any queue misuse can cause this error. But typically this error occurs because worker thread work items are misused.

An entry on a queue can be inserted on the list only one time. When an item is removed from a queue, its **flink** field is set to NULL. Then, when this item is removed the second time, this bug check occurs.

In most situations, the queue that is being referenced is an **ExWorkerQueue** (executive worker queue). To help identify the driver that caused the error, Parameter 4 displays the address of the worker routine that would have been called if this work item had been valid. However, if the queue that is being referenced is *not* an **ExWorkerQueue**, this parameter is not useful.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x97: BOUND_IMAGE_UNSUPPORTED

The BOUND_IMAGE_UNSUPPORTED bug check has a value of 0x00000097.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x98: END_OF_NT_EVALUATION_PERIOD

The END_OF_NT_EVALUATION_PERIOD bug check has a value of 0x00000098. This bug check indicates that the trial period for the Microsoft Windows operating system has ended.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The low-order 32 bits of the product expiration date
2	The high-order 32 bits of the product expiration date
3	Reserved
4	Reserved

Cause

Your installation of the Windows operating system is an evaluation unit with an expiration date. The trial period is over.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x99: INVALID_REGION_OR_SEGMENT

The INVALID_REGION_OR_SEGMENT bug check has a value of 0x00000099. This bug check indicates that **ExInitializeRegion** or **ExInterlockedExtendRegion** was called with an invalid set of parameters.

Parameters

None

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9A: SYSTEM_LICENSE_VIOLATION

The SYSTEM_LICENSE_VIOLATION bug check has a value of 0x0000009A. This bug check indicates that the software license agreement has been violated.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x00	**0:** The product should be WinNT **1:** The product should be LanmanNT or ServerNT	A partial serial number	The first two characters of the product type from the product options	Offline product type changes have been attempted.
0x01	The registered evaluation time from source 1	A partial serial number	The registered evaluation time from an alternate source	Offline changes to the Microsoft Windows evaluation unit time period have been attempted.
0x02	The status code that is associated with the open failure	0	0	The setup key could not be opened.
0x03	The status code that is associated with the key lookup failure	0	0	The SetupType or SetupInProgress value from the setup key is missing, so setup mode could not be detected.
0x04	The status code that is associated with the key lookup failure	0	0	The **SystemPrefix** value from the setup key is missing.
0x05	(See the setup code)	An invalid value was found in licensed processors	The officially licensed number of processors	Offline changes to the number of licensed processors have been attempted.
0x06	The status code that is associated with the open failure	0	0	The **ProductOptions** key could not be opened.
0x07	The status code that is associated with the read failure	0	0	The **ProductType** value could not be read.
0x08	The status code that is associated with the Change Notify failure	0	0	Change Notify on **ProductOptions** failed.
0x09	The status code that is associated with the Change Notify failure	0	0	Change Notify on **SystemPrefix** failed.
0x0A	0	0	0	An NTW system was converted to an NTS system.
0x0B	The status code that is associated with the change failure	0	0	The reference of the setup key failed.
0x0C	The status code that is associated with the change failure	0	0	The reference of the product options key failed.
0x0D	The status code that is associated with the failure	0	0	The attempt to open **ProductOptions** in the worker thread failed.
0x0F	The status code that is associated with the failure	0	0	The attempt to open the setup key failed.
0x10	The status code that is associated with the failure	**0:** set value failed **1:** Change Notify failed	0	A failure occurred in the setup key worker thread.
0x11	The status code that is associated with the failure	**0:** set value failed **1:** Change Notify failed	0	A failure occurred in the product options key worker thread.
0x12	The status code that is associated with the failure	0	0	Unable to open the **LicenseInfoSuites** key for the suite.
0x13	The status code that is associated with the failure	0	0	Unable to query the **LicenseInfoSuites** key for the suite.
0x14	The size of the memory allocation	0	0	Unable to allocate memory.
0x15	The status code that is associated with the failure	Reserved	0	Unable to reset the **ConcurrentLimit** value for the suite key.
0x16	The status code that is associated with the failure	0	0	Unable to open the license key for a suite product.
0x17	The status code that is associated with the failure	0	0	Unable to reset the **ConcurrentLimit** value for a suite product.
0x18	The status code that is associated with the open failure	Reserved	0	Unable to start the Change Notify for the **LicenseInfoSuites**.
0x19	0	0	0	A suite is running on a system that must be PDC.
0x1A	The status code that is associated with the failure	0	0	A failure occurred when enumerating the suites.
0x1B	0	0	0	Changes to the policy cache were attempted.

Cause

The Microsoft Windows operating system detects a violation of the software license agreement.

A user might have tried to change the product type of an offline system or change the trial period of an evaluation unit of Windows. For more information about the specific violation, see the parameter list.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9B: UDFS_FILE_SYSTEM

The UDFS_FILE_SYSTEM bug check has a value of 0x0000009B. This bug check indicates that a problem occurred in the UDF file system.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	If **UdfExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **UdfExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved.

Cause

The UDFS_FILE_SYSTEM bug check might be caused disk corruption. Corruption in the file system or bad blocks (sectors) on the disk can induce this error. Corrupted SCSI and IDE drivers can also adversely affect the system's ability to read and write to the disk and cause the error.

This bug check might also occur if nonpaged pool memory is full. If the nonpaged pool memory is full, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver that requires nonpaged pool memory can also trigger this error.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a disk corruption problem: Check Event Viewer for error messages from SCSI and FASTFAT (System Log) or Autochk (Application Log) that might help identify the device or driver that is causing the error. Disable any virus scanners, backup application, or disk defragmenter tools that continually monitor the system. You should also run hardware diagnostics that the system manufacturer supplies. For more information about these procedures, see the owner's manual for your computer. Run **Chkdsk /f /r** to detect and resolve any file system structural corruption. You must restart the system before the disk scan begins on a system partition.

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This memory increases the quantity of nonpaged pool memory that is available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9C: MACHINE_CHECK_EXCEPTION

The MACHINE_CHECK_EXCEPTION bug check has a value of 0x0000009C. This bug check indicates that a fatal machine check exception has occurred.

Parameters

The four parameters that are listed in the message have different meanings, depending on the processor type.

If the processor is based on an older x86-based architecture and has the Machine Check Exception (MCE) feature but not the Machine Check Architecture (MCA) feature (for example, the Intel Pentium processor), the parameters have the following meaning.

Parameter	Description
1	The low 32 bits of P5_MC_TYPE Machine Service Report (MSR)
2	The address of the MCA_EXCEPTION structure
3	The high 32 bits of P5_MC_ADDR MSR
4	The low 32 bits of P5_MC_ADDR MSR

If the processor is based on a newer x86-based architecture and has the MCA feature and the MCE feature (for example, any Intel Processor of family 6 or higher, such as Pentium Pro, Pentium IV, or Xeon), or if the processor is an x64-based processor, the parameters have the following meaning.

Parameter	Description
1	The bank number
2	The address of the MCA_EXCEPTION structure
3	The high 32 bits of MCi_STATUS MSR for the MCA bank that had the error
4	The low 32 bits of MCi_STATUS MSR for the MCA bank that had the error

On an Itanium-based processor, the parameters have the following meaning.

Note Parameter 1 indicates the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x1	The address of the log	The size of the log	0	
0x2	The address of the log	The size of the log	The error code	The system abstraction layer (SAL) returned an error for SAL_GET_STATEINFO while processing MCA.
0x3	The address of the	The size of the	The error	SAL returned an error for SAL_CLEAR_STATEINFO while it processed MCA.

	log	log	code	
0x4	The address of the log	The size of the log	0	Firmware (FW) reported a fatal MCA.
0x5	The address of the log	The size of the log	0	There are two possible causes: • SAL reported a recoverable MCA, but this recovery is not currently supported. • SAL generated an MCA but could not produce an error record.
0xB	The address of the log	The size of the log	0	
0xC	The address of the log	The size of the log	The error code	SAL returned an error for SAL_GET_STATEINFO while processing an INIT event.
0xD	The address of the log	The size of the log	The error code	SAL returned an error for SAL_CLEAR_STATEINFO while it processed an INIT event.
0xE	The address of the log	The size of the log	0	

Comments

For more information about Machine Check Architecture (MCA), see the Intel or AMD Web sites.

Note Starting with Windows Vista, this bug check is no longer supported, and has been replaced with bug Check 0x124: WHEA_UNCORRECTABLE_ERROR.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9E: USER_MODE_HEALTH_MONITOR

The USER_MODE_HEALTH_MONITOR bug check has a value of 0x0000009E. This bug check indicates that one or more critical user-mode components failed to satisfy a health check.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The process that failed to satisfy a health check in the configured time-out
2	The health monitoring time-out, in seconds
3	Reserved
4	Reserved

Cause

Hardware mechanisms, such as watchdog timers, can detect that basic kernel services are not executing. However, resource starvation issues (including memory leaks, lock contention, and scheduling priority misconfiguration) can block critical user-mode components without blocking deferred procedure calls (DPCs) or draining the non-paged pool.

Kernel components can extend watchdog timer functionality to user mode by periodically monitoring critical applications. This bug check indicates that a user-mode health check failed in a way that prevents graceful shutdown. This bug check restores critical services by restarting or enabling application failover to other servers.

On the Microsoft Windows Server 2003, Enterprise Edition, Windows Server 2003, Datacenter Edition, and Windows 2000 with Service Pack 4 (SP4) operating systems, a user-mode hang can also cause this bug check. The bug check occurs in this situation only if the user has set **HangRecoveryAction** to a value of 3.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x9F: DRIVER_POWER_STATE_FAILURE

The DRIVER_POWER_STATE_FAILURE bug check has a value of 0x0000009F. This bug check indicates that the driver is in an inconsistent or invalid power state.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x1	The device object	Reserved	Reserved	The device object that is being freed still has an outstanding power request that it has not completed.
0x2	The target device's device object, if it is	The device object	The driver object, if it is available	The device object completed the I/O request packet (IRP) for the system power state request, but it failed to call

	available			**PoStartNextPowerIrp**.
0x3 (*Windows 2000 only*)	A pointer to the target device object	A pointer to the device object	The IRP	The device driver did not properly set the IRP as "pending" or complete the IRP.
0x3 (*Windows XP and later*)	The physical device object (PDO) of the stack	The functional device object (FDO) of the stack	The blocked IRP	A device object has been blocking an IRP for too long a time.
0x100 (*Windows 2000 only*)	A pointer to the nonpaged device object	A pointer to the target device object	A pointer to the device object to notify	The device objects in the devnode inconsistently used DO_POWER_PAGABLE.
0x101 (*Windows 2000 only*)	The child device object (FDO)	The child device object (PDO)	The parent device object	A parent device object has detected that a child device has not set the DO_POWER_PAGABLE bit.
0x500 (*Windows XP and Windows Server 2003 only*)	Reserved	The target device's device object, if available	Device object	The device object completed the IRP for the system power state request, but it failed to call **PoStartNextPowerIrp**.

Cause

For a description of the possible causes, see the description of each code in the Parameters section.

The errors that cause Parameter 1 to be 0x3, 0x100, or 0x101 only exist in Microsoft Windows 2000. In Windows XP and later versions of Windows, these errors are superseded by Driver Verifier tests. For more information about Driver Verifier, see the Driver Verifier section of the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA0: INTERNAL_POWER_ERROR

The INTERNAL_POWER_ERROR bug check has a value of 0x000000A0. This bug check indicates that the power policy manager experienced a fatal error.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x1	**1:** A device has overrun its maximum number of reference counts. **2, 3, or 4:** (*Windows Server 2003, Windows XP, and Windows 2000 only*) Too many inrush power IRPs have been queued. **5:** (*Windows Server 2003, Windows XP, and Windows 2000 only*) The power IRP has been sent to a passive level device object.	Except when Parameter 2 is equal to 5, this parameter indicates the maximum number of pending IRPs that are allowed. If Parameter 2 is equal to 5, this parameter is reserved.	Reserved	An error occurred during the handling of the power I/O request packet (IRP).
0x2	Reserved	Reserved	Reserved	An internal failure has occurred while attempting to process a power event.
0x3	The expected checksum	The actual checksum	The line number of the failure	The checksum for a hibernation context page does not match its expected checksum.
0x4	The expected checksum	The actual checksum	The line number of the failure	The checksum for a page about to be written to the hibernation file does not match its expected checksum.
0x5	Reserved	Reserved	Reserved	An unknown shutdown code has been sent to the system shutdown handler.
0x7	Reserved	Reserved	Reserved	An unhandled exception has occurred.
0x8				A fatal error occurred while processing a system power event. (*See the following table for more details.*)

When Parameter 1 is0x8, a fatal error occurred while processing a system power event. In this situation, Parameter 2 indicates the cause of the error. The meaning of the other parameters depends on the value of Parameter 2.

Parameter 2	Parameter 3	Parameter 4	Cause
0x100	The device object	The address of the POWER_CHANNEL_SUMMARY structure (a record of device object states)	An unknown device type is being processed.
0x101	Exception pointer	Reserved	An unhandled exception occurred.
0x102	The address of the DUMP_INITIALIZATION_CONTEXT structure, which contains the information that is passed from the	The address of the POP_HIBER_CONTEXT structure, which contains information about the state of the computer	The hibernation working buffer size is not page-aligned.

	system to the disk dump driver during the driver's initialization	before hibernation	
0x103	The address of the POP_HIBER_CONTEXT structure	Reserved	Some working pages were not accounted for during the hibernation process.
0x104	The address of the POP_HIBER_CONTEXT structure	Reserved	An attempt was made to map internal hibernation memory while the internal memory structures were locked.
0x105	The address of the POP_HIBER_CONTEXT structure	Reserved	An attempt was made to map internal hibernation memory with an unsupported memory type flag.
0x106	The MDL	Reserved	A memory descriptor list (MDL) was created during the hibernation process that describes memory that is not paged-aligned.
0x107	The address of the POP_HIBER_CONTEXT structure	The address of the PO_MEMORY_RANGE_ARRAY structure	A data mismatch occurred in the internal hibernation data structures.
0x108	The address of the POP_HIBER_CONTEXT structure	Reserved	The disk subsystem failed to properly write part of the hibernation file.
0x109	The expected checksum	The actual checksum	The checksum for the processor state data does not match its expected checksum.
0x10A	The address of the POP_HIBER_CONTEXT structure	The NTSTATUS failure code	The disk subsystem failed to properly write part of the hibernation file.
0x200	The device object	The address of the DEVICE_OBJECT_POWER_EXTENSION notification structure	An unknown device type is being checked for an idle state.
0x300	The device object	The IRP	An unknown status was returned from a battery power IRP.
0x301	The device object	The IRP	The battery has entered an unknown state.
0x400	The IRP stack location	The device object	A device has overrun its maximum number of reference counts.
0x401, 0x402, or 0x403	The pending IRP list	The device object	(*Windows Server 2003, Windows XP, and Windows 2000 only)* Too many inrush power IRPs have been queued.
0x404	The IRP stack location	The device object	(*Windows Server 2003, Windows XP, and Windows 2000 only)* A power IRP has been sent to a passive level device object.
0x500	The IRP	The device object	An unknown status was returned from a thermal power IRP.

Cause

For more information about the exact cause for the INTERNAL_POWER_ERROR bug check, see the tables in the Parameters section above.

Resolving the Problem

The following procedures will help you debug certain instances of this bug check.

Debugging bug check 0xA0 when Parameter 1 equals 0x2

1. Examine the stack. Look for the **ntoskrnl!PopExceptionFilter** function. This function contains the following code as its first argument.

   ```
   (error_code << 16) | _LINE_
   ```

   If the caller is **PopExceptionFilter**, the first argument to this function is of type PEXCEPTION_POINTERS. Note the value of this argument.

2. Use the **dt (Display Type)** command and specify the value that you found in the previous step as *argument*.

   dt nt!_EXCEPTION_POINTERS *argument*

   . This command displays the structure. Note the address of the context record.

3. Use the **.cxr (Display Context Record)** command and specify the context record that you found in the previous step as *record*.

   .cxr *record*

   . This command sets the register context to the proper value.

4. Use a variety of commands to analyze the source of the error. Start with **kb (Display Stack Backtrace)** .

Debugging bug check 0xA0 when Parameter 1 equals 0x7

1. Examine the stack. Look for the **ntoskrnl!PopExceptionFilter** function. The first argument to this function is of type PEXCEPTION_POINTERS. Note the value of this argument.
2. Use the **dt (Display Type)** command and specify the value that you found in the previous step as *argument*.

   ```
   dt nt!_EXCEPTION_POINTERS argument
   ```

   This command displays the structure. Note the address of the context record.

3. Use the **.cxr (Display Context Record)** command and specify the context record that you found in the previous step as *record*.

   ```
   .cxr record
   ```

   This command sets the register context to the proper value.

4. Use a variety of commands to analyze the source of the error. Start with **kb (Display Stack Backtrace)** .

Debugging bug check 0xA0 when Parameter 1 equals 0x8 and Parameter 2 equals 0x101

1. Use the **dt (Display Type)** command and specify the value of Parameter 3 as *argument*.

   ```
   dt nt!_EXCEPTION_POINTERS argument
   ```

   This command displays the structure. Note the address of the context record.

2. Use the **.cxr (Display Context Record)** command and specify the context record that you found the previous step as *record*.

   ```
   .cxr record
   ```

   This command sets the register context to the proper value.

3. Use a variety of commands to analyze the source of the error. Start with **kb (Display Stack Backtrace)** .

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA1: PCI_BUS_DRIVER_INTERNAL

The PCI_BUS_DRIVER_INTERNAL bug check has a value of 0x000000A1. This bug check indicates that the PCI Bus driver detected inconsistency problems in its internal structures and could not continue.

Parameters

None

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA2: MEMORY_IMAGE_CORRUPT

The MEMORY_IMAGE_CORRUPT bug check has a value of 0x000000A2. This bug check indicates that corruption has been detected in the image of an executable file in memory.

Parameters

The following parameters appear on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x02	*If Parameter 3 is zero:* The page number in the table page that failed *If Parameter 3 is nonzero:* The page number with the failing page run index	Zero, or the index that failed to match the run	0	A table page check failure occurred.
0x03	The starting physical page number of the range	The length (in pages) of the range	The page number of the table page that contains this run	The checksum for the range of memory listed is incorrect.

Cause

A cyclic redundancy check (CRC) check on the memory range has failed.

On a system wake operation, various regions of memory might be checked to guard against memory failures.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA3: ACPI_DRIVER_INTERNAL

The ACPI_DRIVER_INTERNAL bug check has a value of 0x000000A3. This bug check indicates that the ACPI driver detected an internal inconsistency.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	Reserved
2	Reserved
3	Reserved
4	Reserved

Cause

An inconsistency in the ACPI driver is so severe that continuing to run would cause serious problems.

One possible source of this problem is a BIOS error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA4: CNSS_FILE_SYSTEM_FILTER

The CNSS_FILE_SYSTEM_FILTER bug check has a value of 0x000000A4. This bug check indicates that a problem occurred in the CNSS file system filter.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") identify the source file by its identifier number. The low 16 bits identify the source line in the file where the bug check occurred.
2	Reserved
3	Reserved
4	Reserved

Cause

The CNSS_FILE_SYSTEM_FILTER bug check might occur because nonpaged pool memory is full. If the nonpaged pool memory is completely full, this error can stop the system. However, during the indexing process, if the amount of available nonpaged pool memory is very low, another kernel-mode driver that requires nonpaged pool memory can also trigger this error.

Resolving the Problem

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This memory sincrease the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA5: ACPI_BIOS_ERROR

The ACPI_BIOS_ERROR bug check has a value of 0x000000A5. This bug check indicates that the Advanced Configuration and Power Interface (ACPI) BIOS of the computer is not fully compliant with the ACPI specification.

Parameters

Four bug check parameters appear on the blue screen. Parameter 1 indicates the kind of the incompatibility. The meaning of the other parameters depends on the value of Parameter 1.

If the BIOS incompatibility is related to Plug and Play (PnP) or power management, the following parameters are used.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x01	ACPI's **deviceExtension**	ACPI's **ResourceList**	**0:** No resource list is found **1:** No IRQ resource is found in list	ACPI cannot find the System Control Interrupt (SCI) vector in the resources that are handed to it when ACPI is started.
0x02				*(See the table later on this page)*
0x03	The ACPI object that was being run	The return value from the interpreter	The name of the control method (in ULONG format)	ACPI tried to run a control method while creating device extensions to represent the ACPI namespace, but this control method failed.
0x04	The ACPI extension that _PRW belongs to	A pointer to the method	The **DataType** returned (see *Amli.h*)	ACPI evaluated a _PRW and expected to find an integer as a package element.
0x05	The ACPI extension that _PRW belongs to	Aointer to the _PRW	The number of elements in the _PRW	ACPI evaluated a _PRW, and the package that came back failed to contain at least two elements. The ACPI specification requires that two elements always be present in a _PRW.
0x06	The ACPI extension that _PRx belongs to	A pointer to the _PRx	A pointer to the name of the object to look for	ACPI tried to find a named object, but it could not find the object.
0x07	The ACPI extension that the method belongs to	A pointer to the method	The **DataType** returned (see *Amli.h*)	ACPI evaluated a method and expected to receive a buffer in return. However, the method returned some other data type.
0x08	The ACPI extension that the method belongs to	A pointer to the method	The **DataType** returned (see *Amli.h*)	ACPI evaluated a method and expected to receive an integer in return. However, the method returned some other data type.
0x09	The ACPI extension that the method belongs to	A pointer to the method	The **DataType** returned (see *Amli.h*)	ACPI evaluated a method and expected to receive a package in return. However, the method returned some other data type.
0x0A	The ACPI extension that the method belongs to	A pointer to the method	The **DataType** returned (see *Amli.h*)	ACPI evaluated a method and expected to receive a string in return. However, the method returned some other data type.
0x0B	The ACPI extension that _EJD belongs to	The status that the interpreter returns	The name of the object that ACPI is trying to find	ACPI cannot find the object that an _EJD string references.
0x0C	The ACPI extension that ACPI found a dock device for	A pointer to the _EJD method	**0:** BIOS does not claim system is dockage **1:** Duplicate device extensions for dock device	ACPI provides faulty or insufficient information for dock support.
0x0D	The ACPI extension that ACPI needs the object for	The (ULONG) name of the method that ACPI looked for	**0:** Base case **1:** Conflict	ACPI could not find a required method or object in the namespace This bug check code is used if there is no _HID or _ADR present.
0x0E	The NS **PowerResource** that ACPI needs the object for	The (ULONG) name of the method that ACPI looked for	0: Base case	ACPI could not find a required method or object in the namespace for a power resource (or entity other than a "device"). This bug check code is used if there is no _ON, _OFF, or _STA present for a power resource.
0x0F	The current buffer that ACPI was parsing	The buffer's tag	The specified length of the buffer	ACPI could not parse the resource descriptor.
0x10				*(See the table later on this page)*
0x11				*(See the table later on this page)*
0x14	The current buffer that ACPI was parsing	The buffer's tag	A pointer to a variable that contains the ULONGLONG length of the buffer	ACPI could not parse the resource descriptor. The length exceeds MAXULONG.
0x15	The ACPI Machine Language (AML) context	**1:** Failed to load table **2:** The Parameter Path String Object was not found **3:** Failed to insert Parameter Data into the ParameterPath String Object **4:** Out of system memory	The NT status code	ACPI had a fatal error when attempting to load a table.
0x16	A pointer to the parent NSOBJ	A pointer to the illegal child ACPI namespace object	Reserved	ACPI had a fatal error when processing an xSDT. An object was declared as a child of a parent that cannot have children.

If an interrupt routing failure or incompatibility has occurred, the following parameters are used.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x2001	**InterruptModel** (integer)	The return value from the interpreter	A pointer to the PIC control method	ACPI tried to evaluate the PIC control method but failed.
0x10001	A pointer to the device	A pointer to the parent of the device object	A pointer to the _PRT object	ACPI tried to do interrupt routing, but failed.

	object		(See the following Comments section)	
0x10002	A pointer to the device object	A pointer to the string name that ACPI was looking for but could not find	A pointer to the _PRT object (See the following Comments section)	ACPI could not find the link node referenced in a _PRT.
0x10003	A pointer to the device object	The device ID or function number. This DWORD is encoded as follows: bits 5:0 are the PCI device number, and bits 8:6 are the PCI function number	A pointer to the _PRT object (See the following Comments section)	ACPI could not find a mapping in the _PRT package for a device.
0x10005	A pointer to the _PRT object (See the following Comments section)	A pointer to the current _PRT element. (This pointer is an index into the _PRT.)	The device ID or function number. This DWORD is encoded as follows: bits 15:0 are the PCI function number, and bits 31:16 are the PCI device number	ACPI found an entry in the _PRT that the function ID is not all F's for. (The generic format for a _PRT entry is that the device number is specified, but the function number is not.)
0x10006	A pointer to the link node. (This device is missing the _DIS method.)	0	0	ACPI found a link node, but it cannot disable the node. (Link nodes must be disabled to allow for reprogramming.)
0x10007	The vector that could not be found	0	0	The _PRT contained a reference to a vector that is not described in the I/O APIC entry's MAPIC table.
0x10008	The invalid interrupt level.	0	0	The ACPI SCI interrupt level is invalid.
0x10009	0	0	0	The Fixed ACPI Description Table (FADT) could not be located.
0x1000A	0	0	0	The Root System Description Pointer (RSDP) or Extended System Description Table (XSDT) could not be located
0x1000B	The ACPI table signature	A pointer to the ACPI table	0	The length of the ACPI table is not consistent with the table revision.
0x20000	The I/O port in the Fixed Table	0	0	The PM_TMR_BLK entry in the Fixed ACPI Description Table doesn't point to a working ACPI timer block.

If a miscellaneous failure or incompatibility has occurred, the following parameters are used.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause
0x20000	The I/O port in the Fixed Table	0	0	The PM_TMR_BLK entry in the Fixed ACPI Description Table does not point to a working ACPI timer block.

If Parameter 1 equals **0x02**, the ACPI BIOS could not process the resource list for the PCI root buses. In this case, Parameter 3 specifies the exact problem, and the remaining parameters have the following definitions.

Parameter 2	Parameter 3	Parameter 4	Cause
The ACPI extension for the PCI bus	0x0	A pointer to the QUERY_RESOURCES IRP	ACPI cannot convert the BIOS' resource list into the proper format. This probably represents an error in the BIOS' list encoding procedure.
The ACPI extension for the PCI bus	0x1	A pointer to the QUERY_RESOURCE_REQUIREMENTS IRP	ACPI cannot convert the BIOS' resource list into the proper format. This probably represents an error in the BIOS' list encoding procedure.
The ACPI extension for the PCI bus	0x2	0	ACPI found an empty resource list.
The ACPI extension for the PCI bus	0x3	A pointer to the PNP CRS descriptor	ACPI could not find the current bus number in the CRS.
The ACPI extension for the PCI bus	A pointer to the resource list for PCI	A pointer to the E820 memory table	The list of resources that PCI claims to decode overlaps with the list of memory regions that the E820 BIOS interface reports. (This kind of conflict is never permitted.)

If Parameter 1 equals **0x10**, the ACPI BIOS could not determine the system-to-device-state mapping correctly. In this situation, Parameter 3 specifies the exact problem, and the remaining parameters have the following definitions.

Parameter 2	Parameter 3	Parameter 4	Cause
The ACPI extension whose mapping is needed	0x0	The DEVICE_POWER_STATE (this is "x+1")	_PRx was mapped back to a non-supported S-state.
The ACPI extension whose mapping is needed	0x1	The SYSTEM_POWER_STATE that cannot be mapped	ACPI cannot find a D-state to associate with the S-state.

The ACPI extension whose mapping is needed	0x2	The SYSTEM_POWER_STATE that cannot be mapped	The device claims to be able to wake the system when the system is in this S-state, but the system does not actually support this S-state.

If Parameter 1 equals **0x11**, the system could not enter ACPI mode. In this situation, Parameter 2 specifies the exact problem, and the remaining parameters have the following definitions.

Parameter 2	Parameter 3	Parameter 4	Cause
0x0	0	0	The system could not initialize the AML interpreter.
0x1	0	0	The system could not find RSDT.
0x2	0	0	The system could not allocate critical driver structures.
0x3	0	0	The system could not load RSDT.
0x4	0	0	The system could not load DDBs.
0x5	0	0	The system cannot connect the Interrupt vector.
0x6	0	0	SCI_EN never becomes set in PM1 Control Register.
0x7	A pointer to the table that had a bad checksum	Creator revision	The table checksum is incorrect.
0x8	A pointer to the table that ACPI failed to load	Creator revision	ACPI failed to load DDB.
0x9	FADT version	0	Unsupported firmware version.
0xA	0	0	The system could not find MADT.
0xB	0	0	The system could not find any valid Local SAPIC structures in the MADT.

Cause

The value of Parameter 1 indicates the error.

Resolving the Problem

If you are debugging this error, use the **!analyze -v** extension. This extension displays all the relevant data (device extensions, nsobjects, or whatever is appropriate to the specific error).

If you are not performing debugging, this error indicates that you have to obtain a new BIOS. Contact your vendor or visit the internet to get a new BIOS.

If you cannot obtain an updated BIOS, or the latest BIOS is still not ACPI compliant, you can turn off ACPI mode during text-mode setup. To turn off ACPI mode, press the F7 key when you are prompted to install storage drivers. The system does not notify you that the F7 key was pressed, but it silently disables ACPI and enables you to continue your installation.

Comments

A PCI routing table (_PRT) is the ACPI BIOS object that specifies how all the PCI devices are connected to the interrupt controllers. A computer with multiple PCI buses might have multiple _PRTs.

You can display a _PRT in the debugger by using the **!acpikd.nsobj** extension together with the address of the _PRT object as its argument.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xA7: BAD_EXHANDLE

The BAD_EXHANDLE bug check has a value of 0x000000A7. This bug check indicates that the kernel-mode handle table detected an inconsistent handle table entry state.

Parameters

None

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xAB: SESSION_HAS_VALID_POOL_ON_EXIT

The SESSION_HAS_VALID_POOL_ON_EXIT bug check has a value of 0x000000AB. This bug check indicates that a session unload occurred while a session driver still held memory.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The session ID.

2	The number of paged pool bytes that are leaking.
3	The number of nonpaged pool bytes that are leaking.
4	The total number of paged and nonpaged allocations that are leaking. (The number of nonpaged allocations are in the upper half of this word, and paged allocations are in the lower half of this word.)

Cause

The SESSION_HAS_VALID_POOL_ON_EXIT bug check occurs because a session driver does not free its pool allocations before a session unload. This bug check indicates a bug in *Win32k.sys*, *Atmfd.dll*, *Rdpdd.dll*, or a video driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xAC: HAL_MEMORY_ALLOCATION

The HAL_MEMORY_ALLOCATION bug check has a value of 0x000000AC. This bug check indicates that the hardware abstraction layer (HAL) could not obtain sufficient memory.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	The allocation size
2	0
3	A pointer to a string that contains the file name
4	Reserved

Cause

The HAL could not obtain non-paged memory pool for a system critical requirement.

These critical memory allocations are made early in system initialization, and the HAL_MEMORY_ALLOCATION bug check is not expected. This bug check probably indicates some other critical error such as pool corruption or massive consumption.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xAD: VIDEO_DRIVER_DEBUG_REPORT_REQUEST

The VIDEO_DRIVER_DEBUG_REPORT_REQUEST bug check has a value of 0x000000AD. This bug check indicates that the video port created a non-fatal minidump on behalf of the video driver during run time.

Parameters

The following parameters appear on the blue screen.

Parameter	Description
1	Driver-specific
2	Driver-specific
3	Driver-specific
4	The number of all reports that have been requested since boot time

Comments

The video port created a non-fatal minidump on behalf of the video driver during run time because the video driver requested a debug report.

The VIDEO_DRIVER_DEBUG_REPORT_REQUEST bug check can be caused only by minidump creation, not by the creation of a full dump or kernel dump.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xB4: VIDEO_DRIVER_INIT_FAILURE

The VIDEO_DRIVER_INIT_FAILURE bug check has a value of 0x000000B4. This indicates that Windows was unable to enter graphics mode.

Parameters

None

Cause

The system was not able to go into graphics mode because no display drivers were able to start.

This usually occurs when no video miniport drivers are able to load successfully.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xB8: ATTEMPTED_SWITCH_FROM_DPC

The ATTEMPTED_SWITCH_FROM_DPC bug check has a value of 0x000000B8. This indicates that an illegal operation was attempted by a delayed procedure call (DPC) routine.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The original thread causing the failure
2	The new thread
3	The stack address of the original thread
4	Reserved

Cause

A wait operation, attach process, or yield was attempted from a DPC routine. This is an illegal operation.

Resolving the Problem

The stack trace will lead to the code in the original DPC routine that caused the error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xB9: CHIPSET_DETECTED_ERROR

The CHIPSET_DETECTED_ERROR bug check has a value of 0x000000B9.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xBA: SESSION_HAS_VALID_VIEWS_ON_EXIT

The SESSION_HAS_VALID_VIEWS_ON_EXIT bug check has a value of 0x000000BA. This indicates that a session driver still had mapped views when the session unloaded.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description

1	The session ID
2	The number of mapped views that are leaking
3	The address of this session's mapped views table
4	The size of this session's mapped views table

Cause

This error is caused by a session driver not unmapping its mapped views prior to a session unload. This indicates a bug in *win32k.sys*, *atmfd.dll*, *rdpdd.dll*, or a video driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xBB: NETWORK_BOOT_INITIALIZATION_FAILED

The NETWORK_BOOT_INITIALIZATION_FAILED bug check has a value of 0x000000BB. This indicates that Windows failed to successfully boot off a network.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The part of network initialization that failed. Possible values are: **1:** Failure while updating the registry. **2:** Failure while starting the network stack. Windows sends IOCTLs to the redirector and datagram receiver, then waits for the redirector to be ready. If it is not ready within a certain period of time, this error is issued. **3:** Failure while sending the DHCP IOCTL to TCP. This is how Windows informs the transport of its IP address.
2	The failure status
3	Reserved
4	Reserved

Cause

This error is caused when Windows is booting off a network, and a critical function fails during I/O initialization.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xBC: NETWORK_BOOT_DUPLICATE_ADDRESS

The NETWORK_BOOT_DUPLICATE_ADDRESS bug check has a value of 0x000000BC. This indicates that a duplicate IP address was assigned to this machine while booting off a network.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The IP address, shown as a DWORD. An address of the form *aa.bb.cc.dd* will appear as 0xDDCCBBAA.
2	The hardware address of the other machine. (For an Ethernet connection, see the following note.)
3	The hardware address of the other machine. (For an Ethernet connection, see the following note.)
4	The hardware address of the other machine. (For an Ethernet connection, this will be zero.)

Note When Parameter 4 equals zero, this indicates an Ethernet connection. In that case, the MAC address will be stored in Parameter 2 and Parameter 3. An Ethernet MAC address of the form *aa-bb-cc-dd-ee-ff* will cause Parameter 2 to equal 0xAABBCCDD, and Parameter 3 to equal 0xEEFF0000.

Cause

This error indicates that when TCP/IP sent out an ARP for its IP address, it got a response from another machine indicating a duplicate IP address.

When Windows is booting off a network, this is a fatal error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xBE: ATTEMPTED_WRITE_TO_READONLY_MEMORY

The ATTEMPTED_WRITE_TO_READONLY_MEMORY bug check has a value of 0x000000BE. This is issued if a driver attempts to write to a read-only memory segment.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address of attempted write
2	PTE contents
3	Reserved
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xBF: MUTEX_ALREADY_OWNED

The MUTEX_ALREADY_OWNED bug check has a value of 0x000000BF. This indicates that a thread attempted to acquire ownership of a mutex it already owned.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the mutex
2	The thread that caused the error
3	0
4	Reserved

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC1: SPECIAL_POOL_DETECTED_MEMORY_CORRUPTION

The SPECIAL_POOL_DETECTED_MEMORY_CORRUPTION bug check has a value of 0x000000C1. This indicates that the driver wrote to an invalid section of the special pool.

Parameters

The following parameters are displayed on the blue screen. Parameter 4 indicates the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
Address that the driver tried to free	Reserved	0	0x20	A driver attempted to free pool which was not allocated.
Address that the driver tried to free	Bytes requested	Bytes calculated (actually given to the caller)	0x21, 0x22	A driver attempted to free a bad address.
Address that the driver tried to free	Address where bits are corrupted	Reserved	0x23	A driver freed an address, but nearby bytes within the same page have been corrupted.
Address that the driver tried to free	Address where bits are corrupted	Reserved	0x24	A driver freed an address, but bytes occurring after the end of the allocation have been overwritten.
Current IRQL	Pool type	Number of bytes	0x30	A driver attempted to allocate pool at an incorrect IRQL.
Current IRQL	Pool type	Address that the driver tried to free	0x31	A driver attempted to free pool at an incorrect IRQL.
Address that the driver	Address where one bit is	Reserved	0x32	A driver freed an address, but nearby bytes within the same page have

tried to free corrupted a single bit error.

The _POOL_TYPE codes are enumerated in *ntddk.h*. In particular, zero indicates nonpaged pool and one indicates paged pool.

Cause

A driver has written to an invalid section of the special pool.

Resolving the Problem

Obtain a backtrace of the current thread. This backtrace will usually reveal the source of the error.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC2: BAD_POOL_CALLER

The BAD_POOL_CALLER bug check has a value of 0x000000C2. This indicates that the current thread is making a bad pool request.

Parameters

The following parameters are displayed on the blue screen. **Parameter 1** indicates the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x00	0	Pool type	Pool tag	The current thread requested a zero-byte pool allocation.
0x01, 0x02, or 0x04	Pointer to pool header	First part of pool header contents	0	The pool header has been corrupted.
0x06	Reserved	Pointer to pool header	Pool header contents	The current thread attempted to free the pool, which was already freed.
0x07	Reserved	Pool header contents	Address of the block of pool being freed	The current thread attempted to free the pool, which was already freed.
0x08	Current IRQL	Pool type	Size of allocation, in bytes	The current thread attempted to allocate the pool at an invalid IRQL.
0x09	Current IRQL	Pool type	Address of pool	The current thread attempted to free the pool at an invalid IRQL.
0x0A	Address of pool	Allocator's tag	Tag being used in the attempted free	The current thread attempted to free pool memory by using the wrong tag. (The memory might belong to another component.)
0x0B, 0x0C, or 0x0D	Address of pool	Pool allocation's tag	Bad quota process pointer	The current thread attempted to release a quota on a corrupted pool allocation.
0x40	Starting address	Start of system address space	0	The current thread attempted to free the kernel pool at a user-mode address.
0x41	Starting address	Physical page frame	Highest physical page frame	The current thread attempted to free a non-allocated nonpaged pool address.
0x42 or 0x43	Address being freed	0	0	The current thread attempted to free a virtual address that was never in any pool.
0x44	Starting address	Reserved	0	The current thread attempted to free a non-allocated nonpaged pool address.
0x46	Starting address	0	0	The current thread attempted to free an invalid pool address.
0x47	Starting address	Physical page frame	Highest physical page frame	The current thread attempted to free a non-allocated nonpaged pool address.
0x48	Starting address	Reserved	Reserved	The current thread attempted to free a non-allocated paged pool address.
0x50	Starting address	Start offset, in pages, from beginning of paged pool	Size of paged pool, in bytes	The current thread attempted to free a non-allocated paged pool address.
0x60	Starting address	0	0	The current thread attempted to free an invalid contiguous memory address. (The caller of **MmFreeContiguousMemory** is passing a bad pointer.)
0x99	Address that is being freed	0	0	The current thread attempted to free pool with an invalid address. (This code can also indicate corruption in the pool header.)
0x9A	Pool type	Number of bytes requested	Pool tag	The current thread marked an allocation request MUST_SUCCEED. (This pool type is no longer supported.)
0x9B	Pool type	Number of bytes requested	Caller's address	The current thread attempted to allocate a pool with a tag of 0 (This would be untrackable, and possibly corrupt the existing tag tables.)

0x9C	Pool type	Number of bytes requested	Caller's address	The current thread attempted to allocate a pool with a tag of "BIG". (This would be untrackable and could possibly corrupt the existing tag tables.)
0x9D	Incorrect pool tag used	Pool type	Caller's address	The current thread attempted to allocate a pool with a tag that does not contain any letters or digits. Using such tags makes tracking pool issues difficult.
0x41286	Reserved	Reserved	Start offset from the beginning of the paged pool, in pages	The current thread attempted to free a paged pool address in the middle of an allocation.

The _POOL_TYPE codes are enumerated in *Ntddk.h*. In particular, 0 indicates nonpaged pool and 1 indicates paged pool.

Cause

An invalid pool request has been made by the current thread.

Resolving the Problem

Activate Driver Verifier to obtain more information about these errors. For details, see the Driver Verifier section of the Windows Driver Kit (WDK).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC4: DRIVER_VERIFIER_DETECTED_VIOLATION

The DRIVER_VERIFIER_DETECTED_VIOLATION bug check has a value of 0x000000C4. This is the general bug check code for fatal errors found by Driver Verifier.

Parameters

Four bug check parameters are displayed on the blue screen. Parameter 1 identifies the type of violation. The meaning of the remaining parameters varies with the value of Parameter 1. The parameter values are described in the following table.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x00	Current IRQL	Pool type	0	The driver requested a zero-byte pool allocation.
0x01	Current IRQL	Pool type	Size of allocation, in bytes	The driver attempted to allocate paged memory with IRQL > APC_LEVEL.
0x02	Current IRQL	Pool type	Size of allocation, in bytes	The driver attempted to allocate nonpaged memory with IRQL > DISPATCH_LEVEL.
0x03 (*Windows Vista and later operating systems only)*	Reserved	Reserved	Reserved	The driver attempted to allocate multiple pages of must succeed pool, but at most one page can be allocated using this routine.
0x10	Bad Address	0	0	The driver attempted to free an address that was not returned from an allocate call.
0x11	Current IRQL	Pool type	Address of pool	The driver attempted to free paged pool with IRQL > APC_LEVEL.
0x12	Current IRQL	Pool type	Address of pool	The driver attempted to free nonpaged pool with IRQL > DISPATCH_LEVEL.
0x13 or 0x14	Reserved	Pointer to pool header	Pool header contents	The driver attempted to free memory pool which was already freed.
0x15	Timer entry	Pool type (-1 for special pool)	Pool address being freed	The driver attempted to free pool which contains an active timer.
0x16	Reserved	Pool address	0	The driver attempted to free pool at a bad address, or the driver passed invalid parameters to a memory routine.
0x17	Resource entry	Pool type (-1 for special pool)	Pool address being freed	The driver attempted to free pool which contains an active ERESOURCE.
0x30	Current IRQL	Requested IRQL	0	The driver passed an invalid parameter to **KeRaiseIrql**. (The parameter was either a value lower than the current IRQL, or a value higher than HIGH_LEVEL. This may be the result of using an uninitialized parameter.)
0x31	Current IRQL	Requested IRQL	**0:** New IRQL is bad **1:** New IRQL is invalid inside a DPC routine	The driver passed an invalid parameter to **KeLowerIrql**. (The parameter was either a value higher than the current IRQL, or a value higher than HIGH_LEVEL. This may be the result of using an uninitialized parameter.)
0x32	Current IRQL	Spin lock address	0	The driver called **KeReleaseSpinLock** at an IRQL other than DISPATCH_LEVEL. (This may be due to a double-release of a spin lock.)
0x33	Current IRQL	Fast mutex address	0	The driver attempted to acquire fast mutex with IRQL > APC_LEVEL.

0x34	Current IRQL	Fast mutex address	0	The driver attempted to release fast mutex at an IRQL other than APC_LEVEL.
0x35	Current IRQL	Spin lock address	Old IRQL	The kernel released a spin lock with IRQL not equal to DISPATCH_LEVEL.
0x36	Current IRQL	Spin lock number	Old IRQL	The kernel released a queued spin lock with IRQL not equal to DISPATCH_LEVEL.
0x37	Current IRQL	Thread APC disable count	Resource	The driver tried to acquire a resource, but APCs are not disabled.
0x38	Current IRQL	Thread APC disable count	Resource	The driver tried to release a resource, but APCs are not disabled.
0x39	Current IRQL	Thread APC disable count	Mutex	The driver tried to acquire a mutex "unsafe" with IRQL not equal to APC_LEVEL on entry.
0x3A	Current IRQL	Thread APC disable count	Mutex	The driver tried to release a mutex "unsafe" with IRQL not equal to APC_LEVEL on entry.
0x3B	Current IRQL	Object to wait for	Time-out parameter	The driver called **KeWaitXxx** with IRQL >= DISPATCH_LEVEL. (This is permitted only if the driver already owns the DISPATCHER lock and it passes a time-out value of zero to the routine.)
0x3C	Handle passed to routine	Object type	0	The driver called **ObReferenceObjectByHandle** with a bad handle.
0x3D	0	0	Address of the bad resource	The driver passed a bad (unaligned) resource to **ExAcquireResourceExclusive**.
0x3E	0	0	0	The driver called **KeLeaveCriticalRegion** for a thread that is not currently in a critical region.
0x3F	Object address	New object reference count. **-1:** dereference case **1:** reference case	0	The driver applied **ObReferenceObject** to an object that has a reference count of zero, or the driver applied **ObDereferenceObject** to an object that has a reference count of zero.
0x40	Current IRQL	Spin lock address	0	The driver called **KeAcquireSpinLockAtDpcLevel** with IRQL < DISPATCH_LEVEL.
0x41	Current IRQL	Spin lock address	0	The driver called **KeReleaseSpinLockFromDpcLevel** with IRQL < DISPATCH_LEVEL.
0x42	Current IRQL	Spin lock address	0	The driver called **KeAcquireSpinLock** with IRQL > DISPATCH_LEVEL.
0x51	Base address of allocation	Address of the reference beyond the allocation	Number of charged bytes	The driver attempted to free memory after having written past the end of the allocation. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x52	Base address of allocation	Reserved	Number of charged bytes	The driver attempted to free memory after having written past the end of the allocation. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x53, 0x54, or 0x59	Base address of allocation	Reserved	Reserved	The driver attempted to free memory after having written past the end of the allocation. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x60	Bytes allocated from paged pool	Bytes allocated from nonpaged pool	Total number of allocations that were not freed	The driver is unloading without first freeing its pool allocations. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x61	Bytes allocated from paged pool	Bytes allocated from nonpaged pool	Total number of allocations that were not freed	A driver thread is attempting to allocate pool memory while the driver is unloading. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x62	Name of the driver	Reserved	Total number of allocations that were not freed, including both paged and nonpaged pool	The driver is unloading without first freeing its pool allocations. A bug check with this parameter occurs only when the **Pool Tracking** option of Driver Verifier is active.
0x6F	MDL address	Physical page being locked	Highest physical page in the system	The driver passed a page to **MmProbeAndLockPages** that was not in the PFN database. (This often results from a driver that attempts to lock its own private dualport RAM. Such behavior can corrupt memory on machines with noncontiguous physical RAM.)
0x70	Current IRQL	MDL address	Access mode	The driver called **MmProbeAndLockPages** with IRQL > DISPATCH_LEVEL.
0x71	Current IRQL	MDL address	Process address	The driver called **MmProbeAndLockProcessPages** with IRQL > DISPATCH_LEVEL.
0x72	Current IRQL	MDL address	Process address	The driver called **MmProbeAndLockSelectedPages** with IRQL > DISPATCH_LEVEL.
0x73	Current IRQL	*In 32-bit Windows:* Low 32 bits of the physical address *In 64-bit Windows:* the 64-	Number of bytes	The driver called **MmMapIoSpace** with IRQL > DISPATCH_LEVEL.

		bit physical address		
0x74	Current IRQL	MDL address	Access mode	The driver called **MmMapLockedPages** in kernel mode with IRQL > DISPATCH_LEVEL.
0x75	Current IRQL	MDL address	Access mode	The driver called **MmMapLockedPages** in user mode with IRQL > APC_LEVEL.
0x76	Current IRQL	MDL address	Access mode	The driver called **MmMapLockedPagesSpecifyCache** in kernel mode with IRQL > DISPATCH_LEVEL.
0x77	Current IRQL	MDL address	Access mode	The driver called **MmMapLockedPagesSpecifyCache** in user mode with IRQL > APC_LEVEL.
0x78	Current IRQL	MDL address	0	The driver called **MmUnlockPages** with IRQL > DISPATCH_LEVEL.
0x79	Current IRQL	Virtual address being unmapped	MDL address	The driver called **MmUnmapLockedPages** in kernel mode with IRQL > DISPATCH_LEVEL.
0x7A	Current IRQL	Virtual address being unmapped	MDL address	The driver called **MmUnmapLockedPages** in user mode with IRQL > APC_LEVEL.
0x7B	Current IRQL	Virtual address being unmapped	Number of bytes	The driver called **MmUnmapIoSpace** with IRQL > APC_LEVEL.
0x7C	MDL address	MDL flags	0	The driver called **MmUnlockPages**, and passed an MDL whose pages were never successfully locked.
0x7D	MDL address	MDL flags	0	The driver called **MmUnlockPages**, and passed an MDL whose pages are from nonpaged pool. (These should never be unlocked.)
0x80	Current IRQL	Event address	0	The driver called **KeSetEvent** with IRQL > DISPATCH_LEVEL.
0x81	MDL address	MDL flags	0	The driver called **MmMapLockedPages**. (You should use **MmMapLockedPagesSpecifyCache** instead, with the *BugCheckOnFailure* parameter set to FALSE.)
0x82	MDL address	MDL flags	0	The driver called **MmMapLockedPagesSpecifyCache** with the *BugCheckOnFailure* parameter equal to TRUE. (This parameter should be set to FALSE.)
0x83	Start of physical address range to map	Number of bytes to map	First page frame number that isn't locked down	The driver called **MmMapIoSpace** without having locked down the MDL pages. The physical pages represented by the physical address range being mapped must have been locked down prior to making this call.
0x84	Start of physical address range to map	Number of bytes to map	First page frame number that is on the free list	The driver called **MmMapIoSpace** without having locked down the MDL pages (or after freeing the MDL pages).
0x85	MDL address	Number of pages to map	First page frame number that isn't locked down	The driver called **MmMapLockedPages** without having locked down the MDL pages.
0x86	MDL address	Number of pages to map	First page frame number that is on the free list	The driver called **MmMapLockedPages** without having locked down the MDL pages (or after freeing the MDL pages).
0x87	Base physical page of the existing mapping (Shift left for physical address)	Number of pages already mapped in the existing mapping	MEMORY_CACHING_TYPE of the existing mapping	The driver called **MmMapIoSpace**, but the caller's cache type conflicts with an existing mapping.
0x88	Base physical page of the requested mapping (Shift left for physical address)	Number of pages in the requested mapping	MEMORY_CACHING_TYPE of the requested mapping	The driver called **MmMapIoSpace** to map a physical range as non-cached or write-combined, but the caller's physical range already has an existing cached mapping.
0x89	MDL address	Pointer to the non-memory page in the MDL	The non-memory page number in the MDL	An MDL is not marked as "I/O", but it contains non-memory page addresses.
0x8A	MDL address	Base physical page of the requested mapping (Shift left for physical address)	MEMORY_CACHING_TYPE of the requested mapping	The driver called **MmMapLockedPages*Xxx*** to map a physical range as non-cached or write-combined, but the caller's physical range already has an existing cached mapping.
0x90 (*Windows 2000, Windows XP, and Windows Server 2003 only*)	Reserved	Reserved	Reserved	The driver switched stacks, and the current stack is neither a thread stack nor a DPC stack. (Typically, the driver doing this should be on the stack obtained by using the **kb (Display Stack Backtrace)** command.)
0x91	Reserved	Reserved	Reserved	The driver switched stacks using a method that is not supported by the operating system. The only supported way to extend a kernel mode stack is by using **KeExpandKernelStackAndCallout**.

0xA0 (*Windows Server 2003 and later operating systems only)*	Pointer to the IRP making the read or write request	Device object of the lower device	Number of the sector in which the error was detected	A cyclic redundancy check (CRC) error was detected on a hard disk. A bug check with this parameter occurs only when the **Disk Integrity Checking** option of Driver Verifier is active.
0xA1 (*Windows Server 2003 and later operating systems only)*	Copy of the IRP making the read or write request. (The actual IRP has been completed.)	Device object of the lower device	Number of the sector in which the error was detected	A CRC error was detected on a sector (asynchronously). A bug check with this parameter occurs only when the **Disk Integrity Checking** option of Driver Verifier is active.
0xA2 (*Windows Server 2003 and later operating systems only)*	IRP making the read or write request, or a copy of this IRP	Device object of the lower device	Number of the sector in which the error was detected	The CRCDISK checksum copies don't match. This could be a paging error. A bug check with this parameter occurs only when the **Disk Integrity Checking** option of Driver Verifier is active.
0xB0 *(Windows Vista and later operating systems only)*	MDL address	MDL flags	Incorrect MDL flags	The driver called **MmProbeAndLockPages** for an MDL with incorrect flags. For example, the driver passed an MDL created by **MmBuildMdlForNonPagedPool** to **MmProbeAndLockPages.**
0xB1 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Incorrect MDL flags	The driver called **MmProbeAndLockProcessPages** for an MDL with incorrect flags. For example, the driver passed an MDL created by **MmBuildMdlForNonPagedPool** to **MmProbeAndLockProcessPages**.
0xB2 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Incorrect MDL flags	The driver called **MmMapLockedPages** for an MDL with incorrect flags. For example, the driver passed an MDL that is already mapped to a system address or that was not locked to **MmMapLockedPages**.
0xB3 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Missing MDL flags (at least one was expected)	The driver called **MmMapLockedPages** for an MDL with incorrect flags. For example, the driver passed an MDL that is not locked to **MmMapLockedPages**.
0xB4 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Unexpected partial MDL flag	The driver called **MmUnlockPages** for a partial MDL. A partial MDL is one that was created by **IoBuildPartialMdl**.
0xB5 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Unexpected partial MDL flag	The driver called **MmUnmapLockedPages** for a partial MDL. A partial MDL is one that was created by **IoBuildPartialMdl**.
0xB6 (*Windows Vista and later operating systems only)*	MDL address	MDL flags	Missing MDL flag	The driver called **MmUnmapLockedPages** for an MDL that is not mapped to a system address.
0xB7 (*Windows Vista and later operating systems only)*	Number of corrupted physical pages	Address of first corrupted physical page	Address of last corrupted physical page	The system BIOS has corrupted low physical memory during a sleep transition.
0xC0 (*Windows Vista and later operating systems only)*	Address of the IRP	Reserved	Reserved	The driver called **IoCallDriver** with interrupts disabled.
0xC1 (*Windows Vista and later operating systems only)*	Address of the driver dispatch routine	Reserved	Reserved	A driver dispatch routine was returned with interrupts disabled.
0xC2 (*Windows Vista and later operating systems only)*	Reserved	Reserved	Reserved	The driver called a Fast I/O dispatch routine after interrupts were disabled.
0xC3 (*Windows Vista and later operating systems only)*	Address of the driver Fast I/O dispatch routine	Reserved	Reserved	A driver Fast I/O dispatch routine was returned with interrupts disabled.
0xC5 (*Windows Vista and later operating systems only)*	Address of the driver dispatch routine	The current thread's APC disable count	The thread's APC disable count prior to calling the driver dispatch routine	A driver dispatch routine has changed the thread's APC disable count. The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex. The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**. Because these calls should always be in pairs, the APC disable count should be zero whenever a thread is exited. A negative value indicates that a driver has disabled APC calls without re-enabling them. A positive value indicates that the reverse is true.
0xC6 (*Windows Vista and later operating systems only)*	Address of the driver Fast I/O dispatch routine	Current thread's APC disable count	The thread's APC disable count prior to calling the Fast I/O driver dispatch routine	A driver Fast I/O dispatch routine has changed the thread's APC disable count. The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex. The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**. Because these calls should always be in pairs, the APC disable count should be zero whenever a thread is exited. A

				negative value indicates that a driver has disabled APC calls without re-enabling them. A positive value indicates that the reverse is true.
0xCA (*Windows Vista and later operating systems only)*	Address of the lookaside list	Reserved	Reserved	The driver has attempted to re-initialize a lookaside list.
0xCB (*Windows Vista and later operating systems only)*	Address of the lookaside list	Reserved	Reserved	The driver has attempted to delete an uninitialized lookaside list.
0xCC (*Windows Vista and later operating systems only)*	Address of the lookaside list	Starting address of the pool allocation	Size of the pool allocation	The driver has attempted to free a pool allocation that contains an active lookaside list.
0xCD (*Windows Vista and later operating systems only)*	Address of the lookaside list	Block size specified by the caller	Minimum supported block size	The driver has attempted to create a lookaside list with an allocation block size that is too small.
0xD0 (*Windows Vista and later operating systems only)*	Address of the ERESOURCE structure	Reserved	Reserved	The driver has attempted to re-initialize an ERESOURCE structure.
0xD1 (*Windows Vista and later operating systems only)*	Address of the ERESOURCE structure	Reserved	Reserved	The driver has attempted to delete an uninitialized ERESOURCE structure.
0xD2 (*Windows Vista and later operating systems only)*	Address of the ERESOURCE structure	Starting address of the pool allocation	Size of the pool allocation	The driver has attempted to free a pool allocation that contains an active ERESOURCE structure.
0xD5 (*Windows Vista and later operating systems only)*	Address of the IO_REMOVE_LOCK structure created by the checked build version of the driver	Current **IoReleaseRemoveLock** tag	Reserved	The current **IoReleaseRemoveLock** tag does not match the previous **IoAcquireRemoveLock** tag. If the driver calling **IoReleaseRemoveLock** is not in a checked build, Parameter 2 is the address of the shadow IO_REMOVE_LOCK structure created by Driver Verifier on behalf of the driver. In this case, the address of the IO_REMOVE_LOCK structure used by the driver is not used at all, because Driver Verifier is replacing the lock address for all the remove lock APIs. A bug check with this parameter occurs only when the **I/O Verification** option of Driver Verifier is active.
0xD6 (*Windows Vista and later operating systems only)*	Address of the IO_REMOVE_LOCK structure created by the checked build version of the driver	Tag that does not match previous **IoAcquireRemoveLock** tag	Previous **IoAcquireRemoveLock** tag	The current **IoReleaseRemoveLockAndWait** tag does not match the previous **IoAcquireRemoveLock** tag. If the driver calling **IoReleaseRemoveLock** is not a checked build, Parameter 2 is the address of the shadow IO_REMOVE_LOCK structure created by Driver Verifier on behalf of the driver. In this case, the address of the IO_REMOVE_LOCK structure used by the driver is not used at all, because Driver Verifier is replacing the lock address for all the remove lock APIs. A bug check with this parameter occurs only when the **I/O Verification** option of Driver Verifier is active.
0xD7 *(*Windows 7*operating systems and later only)*	Address of the checked build Remove Lock structure that is used internally by Driver Verifier	Address of the Remove Lock structure that is specified by the driver	Reserved	A Remove Lock cannot be re-initialized, even after it calls **IoReleaseRemoveLockAndWait**, because other threads might still be using that lock (by calling **IoAcquireRemoveLock**). The driver should allocate the Remove Lock inside its device extension, and initialize it a single time. The lock will be deleted together with the device extension.
0xDA (*Windows Vista and later operating systems only)*	Starting address of the driver	WMI callback address inside the driver	Reserved	An attempt was made to unload a driver that has not deregistered its WMI callback function.
0xDB (*Windows Vista and later operating systems only)*	Address of the device object	Reserved	Reserved	An attempt was made to delete a device object that was not deregistered from WMI.
0xDC (*Windows Vista and later operating systems only)*	Reserved	Reserved	Reserved	An invalid RegHandle value was specified as a parameter of the function **EtwUnregister.**
0xDD (*Windows Vista and later operating systems only)*	Address of the call to **EtwRegister**	Starting address of the unloading driver	Reserved	An attempt was made to unload a driver without calling **EtwUnregister.**
0xDF *(*Windows 7*operating systems and later only)*	Synchronization object address			The synchronization object is in session address space. Synchronization objects are not allowed in session address space because they can be manipulated from another session or from system threads that have no session virtual address space.
0xE0 (*Windows Vista and later operating systems only)*	User-mode address that is used as a parameter	Size ,in bytes, of the address range that is used as a parameter	Reserved	A call was made to an operating system kernel function that specified a user-mode address as a parameter.
0xE1 (*Windows Vista and later operating systems only)*	Address of the synchronization object	Reserved	Reserved	A synchronization object was found to have an address that was either invalid or pageable.
0xE2 (*Windows Vista and later operating*	Address of the IRP	User-mode address present in the IRP	Reserved	An IRP with **Irp->RequestorMode** set to **KernelMode** was found to have a user-mode address as one of its

systems only)				members.
0xE3 (*Windows Vista and later operating systems only)*	Address of the call to the API	User-mode address used as a parameter in the API	Reserved	A driver has made a call to a kernel-mode **Zw*Xxx*** routine with a user-mode address as a parameter.
0xE4 (*Windows Vista and later operating systems only)*	Address of the call to the API	Address of the malformed UNICODE_STRING structure	Reserved	A driver has made a call to a kernel-mode **Zw*Xxx*** routine with a malformed UNICODE_STRING structure as a parameter.
0xE5 (*Windows Vista and later operating systems only)*	Current IRQL	Reserved	Reserved	A call was made to a Kernel API at the incorrect IRQL.
0xEA (*Windows Vista and later operating systems only)*	Current IRQL	The thread's APC disable count	Address of the pushlock	A driver has attempted to acquire a pushlock while APCs are enabled.
0xEB (*Windows Vista and later operating systems only)*	Current IRQL	The thread's APC disable count	Address of the pushlock	A driver has attempted to release a pushlock while APCs are enabled.
0xF0 (*Windows Vista and later operating systems only)*	Address of the destination buffer	Address of the source buffer	Number of bytes to copy	A driver called the **memcpy** function with overlapping source and destination buffers.
0xF5 (*Windows Vista and later operating systems only)*	Address of the NULL handle	Object type	Reserved	A driver passed a NULL handle to **ObReferenceObjectByHandle**.
0xF6 *(*Windows 7*operating systems and later)*	Handle value being referenced	Address of the current process	Address inside the driver that performs the incorrect reference	A driver references a user-mode handle as kernel mode.
0xFA *(*Windows 7*operating systems and later)*	Completion routine address.	IRQL value before it calls the completion routine	Current IRQL value, after it calls the completion routine	The IRP completion routine returned at an IRQL that was different from the IRQL the routine was called at.
0xFB *(*Windows 7*operating systems and later)*	Completion routine address	Current thread's APC disable count	The thread's APC disable count before it calls the IRP completion routine	The thread's APC disable count was changed by the driver's IRP completion routine. The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex. The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**. Because these calls should always be in pairs, the APC disable count should be zero whenever a thread is exited. A negative value indicates that a driver has disabled APC calls without re-enabling them. A positive value indicates that the reverse is true.
0x105 *(*Windows 7*operating systems and later)*	Address of the IRP			The driver uses ExFreePool instead of IoFreeIrp to release the IRP.
0x10A *(*Windows 7*operating systems and later)*				The driver attempts to charge pool quota to the Idle process.
0x10B *(*Windows 7*operating systems and later)*				The driver attempts to charge pool quota from a DPC routine. This is incorrect because the current process context is undefined.
0x110 *(*Windows 7*operating systems and later)*	Address of the Interrupt Service Routine	Address of the extended context that was saved before it executed the ISR	Address of the extended context was saved after it executed the ISR	The interrupt service routine (ISR) for the driver has corrupted the extended thread context.
0x115 *(*Windows 7*operating systems and later)*	The address of the thread that is responsible for the shutdown, which might be deadlocked			Driver Verifier detected that the system has taken longer than 20 minutes and shutdown is not complete.
0x11A *(*Windows 7*operating systems and later)*	Current IRQL			The driver calls KeEnterCriticalRegion at IRQL > APC_LEVEL.
0x11B *(*Windows 7*operating systems and later)*	Current IRQL			The driver calls KeLeaveCriticalRegion at IRQL > APC_LEVEL.

0x120 *(Windows 7operating systems and later)*	Address of the IRQL value	Address of the Object to wait on	Address of Timeout value	The thread waits at IRQL > DISPATCH_LEVEL. Callers of KeWaitForSingleObject or KeWaitForMultipleObjects must run at IRQL <= DISPATCH_LEVEL.
0x121 *(Windows 7operating systems and later)*	Address of the IRQL value	Address of the Object to wait on	Address of Timeout value	The thread waits at IRQL equals DISPATCH_LEVEL and the Timeout is NULL. Callers of KeWaitForSingleObject or KeWaitForMultipleObjects can run at IRQL <= DISPATCH_LEVEL. If a NULL pointer is supplied for Timeout, the calling thread remains in a wait state until the Object is signaled.
0x122 *(Windows 7operating systems and later)*	Address of the IRQL value	Address of the Object to wait on	Address of the Timeout value	The thread waits at DISPATCH_LEVEL and Timeout value is not equal to zero (0). If the Timeout != 0, the callers of KeWaitForSingleObject or KeWaitForMultipleObjects must run at IRQL <= APC_LEVEL.
0x123 *(Windows 7operating systems and later)*	Address of the Object to wait on			The caller of KeWaitForSingleObject or KeWaitForMultipleObjects specified the wait as **UserMode**, but the Object is on the kernel stack.
0x130 *(Windows 7operating systems and later)*	Address of work item			The work item is in session address space. Work items are not allowed in session address space because they can be manipulated from another session or from system threads that have no session virtual address space.
0x131 *(Windows 7operating systems and later)*	Address of work item			The work item is in pageable memory. Work items have to be in nonpageable memory because the kernel uses them at DISPATCH_LEVEL.
0x135	Address of IRP	Number of milliseconds allowed between the **IoCancelIrp** call and the completion for this IRP		The canceled IRP did not completed in the expected time The driver took longer than expected to complete the canceled IRP.
0x13A	Address of the pool block being freed	Incorrect value	Address of the incorrect value	The driver has called **ExFreePool** and Driver Verifier detects an error in one of the internal values that is used to track pool usage.
0x13B	Address of the pool block being freed	Address of the incorrect value	Address of a pointer to the incorrect memory page	The driver has called **ExFreePool** and Driver Verifier detects an error in one of the internal values that is used to track pool usage.
0x13C	Address of the pool block being freed	Incorrect value	Address of the incorrect value	The driver has called **ExFreePool** and Driver Verifier detects an error in one of the internal values that is used to track pool usage.
0x13D	Address of the pool block being freed	Address of the incorrect value	Correct value that was expected	The driver has called **ExFreePool** and Driver Verifier detects an error in one of the internal values that is used to track pool usage.
0x13E	Pool block address specified by the caller	Pool block address tracked by Driver Verifier	Pointer to the pool block address that is tracked by Driver Verifier	The pool block address specified by the caller of **ExFreePool** is different from the address tracked by Driver Verifier.
0x13F	Address of the pool block being freed	Number of bytes being freed	Pointer to the number of bytes tracked by Driver Verifier	The number of bytes of memory being freed in the call to **ExFreePool** is different from the number of bytes tracked by Driver Verifier.
0x1000 (*Windows XP and later operating systems only)*	Address of the resource	Reserved	Reserved	**Self-deadlock:** The current thread has tried to recursively acquire a resource. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1001 (*Windows XP and later operating systems only)*	Address of the resource that was the final cause of the deadlock	Reserved	Reserved	**Deadlock:** A lock hierarchy violation has been found. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active. (Use the **!deadlock** extension for further information.)
0x1002 (*Windows XP and later operating systems only)*	Address of the resource	Reserved	Reserved	**Uninitialized resource:** A resource has been acquired without having been initialized first. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1003 (*Windows XP and later operating systems only)*	Address of the resource that is being released deadlocked	Address of the resource that should have been released first	Reserved	**Unexpected release:** A resource has been released in an incorrect order. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1004 (*Windows XP and later operating systems only)*	Address of the resource	Address of the thread that acquired the resource	Address of the current thread	**Unexpected thread:** The wrong thread releases a resource. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1005 (*Windows XP and later operating*	Address of the resource	Reserved	Reserved	**Multiple initialization:** A resource is initialized more than one time. A bug check with this parameter occurs only

systems only)				when the **Deadlock Detection** option of Driver Verifier is active.
0x1006 (*Windows XP and later operating systems only)*	Address of the thread being deleted	Address of the resource owned by the thread	Reserved	**Thread holds resources:** A thread is deleted before the thread can release its resources. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1007 (*Windows XP and later operating systems only)*	Address of the resource	Reserved	Reserved	**Unacquired resource:** A resource is released before it has been acquired. A bug check with this parameter occurs only when the **Deadlock Detection** option of Driver Verifier is active.
0x1008 (operating systems and later)	Lock address	Driver Verifier internal data	Driver Verifier internal data	The driver tried to acquire a lock by using an API that is mismatched for this lock type.
0x1009 (operating systems and later)	Lock address	Driver Verifier internal data	Driver Verifier internal data	The driver tried to release a lock by using an API that is mismatched for this lock type.
0x100A (operating systems and later)	Owner thread address	Driver Verifier internal data		The terminated thread owns the lock.
0x100B (operating systems and later)	Lock address	Owner thread address	Driver Verifier internal address	The deleted lock is still owned by a thread.

Cause

See the description of each code in the Parameters section for a description of the cause. Further information can be obtained by using the **!analyze -v** extension.

Resolving the Problem

This bug check can only occur when Driver Verifier has been instructed to monitor one or more drivers. If you did not intend to use Driver Verifier, you should deactivate it. You might consider removing the driver which caused this problem as well.

If you are the driver writer, use the information obtained through this bug check to fix the bugs in your code.

For full details on Driver Verifier, see the Driver Verifier section of the Windows Driver Kit (WDK).

Comments

The _POOL_TYPE codes are enumerated in *Ntddk.h*. In particular, **0** (zero) indicates nonpaged pool and **1** (one) indicates paged pool.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC5: DRIVER_CORRUPTED_EXPOOL

The DRIVER_CORRUPTED_EXPOOL bug check has a value of 0x000000C5. This indicates that the system attempted to access invalid memory at a process IRQL that was too high.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write
4	Address that referenced memory

Cause

The kernel attempted to access pageable memory (or perhaps completely invalid memory) when the IRQL was too high. The ultimate cause of this problem is almost certainly a driver that has corrupted the system pool.

In most cases, this bug check results if a driver corrupts a small allocation (less than PAGE_SIZE). Larger allocations result in bug check 0xD0

(DRIVER_CORRUPTED_MMPOOL).

Resolving the Problem

If you have recently installed any new software, check to see if it is properly installed. Check for updated drivers on the manufacturer's website.

To debug this error, use the special pool option of Driver Verifier. If this fails to reveal the driver that caused the error, use the Global Flags utility to enable the special pool by pool tag.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC6: DRIVER_CAUGHT_MODIFYING_FREED_POOL

The DRIVER_CAUGHT_MODIFYING_FREED_POOL bug check has a value of 0x000000C6. This indicates that the driver attempted to access a freed memory pool.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	**0:** Read **1:** Write
3	**0:** Kernel mode **1:** User mode
4	Reserved

Comments

The faulty component will be displayed in the current kernel stack. This driver should be either replaced or debugged.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC7: TIMER_OR_DPC_INVALID

The TIMER_OR_DPC_INVALID bug check has a value of 0x000000C7. This is issued if a kernel timer or delayed procedure call (DPC) is found somewhere in memory where it is not permitted.

Parameters

The following parameters are displayed on the blue screen.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of error
0x0	Address of the timer object	Start of memory range being checked	End of memory range being checked	The timer object was found in a block of memory where a timer object is not permitted. .
0x1	Address of the DPC object	Start of memory range being checked	End of memory range being checked	The DPC object was found in a block of memory where a DPC object is not permitted.
0x2	Address of the DPC routine	Start of memory range being checked	End of memory range being checked	The DPC routine was found in a block of memory where a DPC object is not permitted.
0x3	Address of the DPC object	Processor number	Number of processors in the system	The processor number for the DPC object is not correct.
0x4	Address of the DPC routine	The thread's APC disable count before the kernel calls the DPC routine	The thread's APC disable count after the DPC routine is called	The thread's APC disable count was changed during DPC routine execution. The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex. The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**.

0x5	Address of the DPC routine	The thread's APC disable count before the kernel calls the DPC routine	The thread's APC disable count after the DPC routine is called	The thread's APC disable count was changed during the execution of timer DPC routine. The APC disable count is decremented each time a driver calls **KeEnterCriticalRegion**, **FsRtlEnterFileSystem**, or acquires a mutex. The APC disable count is incremented each time a driver calls **KeLeaveCriticalRegion**, **KeReleaseMutex**, or **FsRtlExitFileSystem**.

Cause

This condition is usually caused by a driver failing to cancel a timer or DPC before freeing the memory where it resides.

Resolving the Problem

If you are the driver writer, use the information obtained through this bug check to fix the bugs in your code.

If you are a system administrator, you should unload the driver if the problem persists.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC8: IRQL_UNEXPECTED_VALUE

The IRQL_UNEXPECTED_VALUE bug check has a value of 0x000000C8. This indicates that the processor's IRQL is not what it should be at this time.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The value of the following bit computation: (Current IRQL << 16) | (Expected IRQL << 8) | UniqueValue
2	Zero, or **APC->KernelRoutine**
3	Zero, or **APC**
4	Zero, or **APC->NormalRoutine**

You can determine "UniqueValue" by computing (Parameter 1 AND 0xFF). If "UniqueValue" is either zero or one, Parameter 2, Parameter 3, and Parameter 4 will equal the indicated APC pointers. Otherwise, these parameters will equal zero.

Cause

This error is usually caused by a device driver or another lower-level program that changed the IRQL for some period and did not restore the original IRQL at the end of that period. For example, the routine may have acquired a spin lock and failed to release it.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC9: DRIVER_VERIFIER_IOMANAGER_VIOLATION

The DRIVER_VERIFIER_IOMANAGER_VIOLATION bug check has a value of 0x000000C9. This is the bug check code for all Driver Verifier **I/O Verification** violations.

Parameters

When Driver Verifier is active and **I/O Verification** is selected, various I/O violations will cause this bug check to be issued. The following parameters will be displayed on the blue screen. Parameter 1 identifies the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error

0x01	Address of IRP being freed	0	0	The driver attempted to free an object whose type is not IO_TYPE_IRP.
0x02	Address of IRP being freed	0	0	The driver attempted to free an IRP that is still associated with a thread.
0x03	Address of IRP being sent	0	0	The driver passed **IoCallDriver** an IRP Type not equal to IRP_TYPE.
0x04	Address of device object	0	0	The driver passed **IoCallDriver** an invalid device object.
0x05	Address of device object associated with offending driver	IRQL before **IoCallDriver**	IRQL after **IoCallDriver**	The IRQL changed during a call to the driver dispatch routine.
0x06	IRP status	Address of IRP being completed	0	The driver called **IoCompleteRequest** with a status marked as pending (or equal to -1).
0x07	Address of cancel routine	Address of IRP being completed	0	The driver called **IoCompleteRequest** while its cancel routine was still set.
0x08	Address of device object	IRP major function code	Exception status code	The driver passed **IoBuildAsynchronousFsdRequest** an invalid buffer.
0x09	Address of device object	I/O control code	Exception status code	The driver passed **IoBuildDeviceIoControlRequest** an invalid buffer.
0x0A	Address of device object	0	0	The driver passed **IoInitializeTimer** a device object with an already-initialized timer.
0x0C	Address of I/O status block	0	0	The driver passed an I/O status block to an IRP, but this block is allocated on a stack which has already unwound past that point.
0x0D	Address of user event object	0	0	The driver passed a user event to an IRP, but this event is allocated on a stack which has already unwound past that point.
0x0E	Current IRQL	Address of IRP	0	The driver called **IoCompleteRequest** with IRQL > DISPATCH_LEVEL.
0x0F	Address of the device object to which the IRP is being sent	Pointer to the IRP	Pointer to file object	The driver sent a create request with a file object that has been closed, or that had its open canceled.

In addition to the errors mentioned in the previous table, there are a number of **I/O Verification** errors that will cause Driver Verifier to halt the system, but which are not actually bug checks.

These errors cause messages to be displayed on the blue screen, in a crash dump file, and in a kernel debugger. These messages will appear differently in each of these locations. When these errors occur, the hexadecimal bug check code 0xC9 and the bug check string DRIVER_VERIFIER_IOMANAGER_VIOLATION do *not* appear on the blue screen or in the debugger, although they will appear in a crash dump file.

On the blue screen, the following data will be displayed:

- The message **IO SYSTEM VERIFICATION ERROR**.
- The message **WDM DRIVER ERROR *XXX***, where *XXX* is a hexadecimal code representing the specific error. (See the table below for a list of the I/O error codes and their meanings.)
- The name of the driver which caused the error.
- The address in the driver's code where the error was detected (Parameter 2).
- A pointer to the IRP (Parameter 3).
- A pointer to the device object (Parameter 4).

If a kernel-mode crash dump has been enabled, the following information will appear in the crash dump file:

- The message **BugCheck 0xC9 (DRIVER_VERIFIER_IOMANAGER_VIOLATION)**.
- The hexadecimal I/O error code. (See the table below for a list of the I/O error codes and their meanings.)
- The address in the driver's code where the error was detected.
- A pointer to the IRP.
- A pointer to the device object.

If a kernel debugger is attached to the system which has caused this violation, the following information will be sent to the debugger:

- The message **WDM DRIVER ERROR**, along with an assessment of the severity of the error.
- The name of the driver which caused the error.
- A descriptive string which explains the cause of this error. Often additional information is passed along, such as a pointer to the IRP. (See the table below for a list of these descriptive strings and what additional information is specified.)
- A query for further action. Possible responses are **b** (break), **i** (ignore), **z** (zap), **r** (remove), or **d** (disable). Instructing the operating system to continue allows you to see what would happen "down the line" if this error had not occurred. Of course, this often will lead to additional bug checks. The "zap" option will actually remove the breakpoint that caused this error to be discovered.

Note No other bug checks can be ignored in this manner. Only this kind of **I/O Verification** errors can be ignored, and even these errors can only be ignored if a kernel debugger is attached.

The following table lists those **I/O Verification** errors that can appear. In Windows 2000, these errors will only be displayed if **I/O Verification** is set to **Level 2**.

I/O Error Code	Severity	Cause of Error
0x200	Unknown	*This code covers all unknown **I/O Verification** errors.*
0x201	Fatal error	A device is deleting itself while there is another device beneath it in the driver stack. This may be because the caller has forgotten to call **IoDetachDevice** first, or the lower driver may have incorrectly deleted itself.
0x202	Fatal error	A driver has attempted to detach from a device object that is not attached to anything. This may occur if detach was called twice on the same device object. (*Device object specified.*)
0x203	Fatal error	A driver has called **IoCallDriver** without setting the cancel routine in the IRP to NULL. (*IRP specified.*)
0x204	Fatal error	The caller has passed in NULL as a device object. This is fatal. (*IRP specified.*)
0x205	Fatal error	The caller is forwarding an IRP that is currently queued beneath it. The code handling IRPs returning STATUS_PENDING in this driver appears to be broken. (*IRP specified.*)
0x206	Fatal error	The caller has incorrectly forwarded an IRP (control field not zeroed). The driver should use **IoCopyCurrentIrpStackLocationToNext** or **IoSkipCurrentIrpStackLocation**. (*IRP specified.*)
0x207	Fatal error	The caller has manually copied the stack and has inadvertently copied the upper layer's completion routine. The driver should use **IoCopyCurrentIrpStackLocationToNext**. (*IRP specified.*)

0x208	Fatal error	This IRP is about to run out of stack locations. Someone may have forwarded this IRP from another stack. (*IRP specified.*)
0x209	Fatal error	The caller is completing an IRP that is currently queued beneath it. The code handling IRPs returning STATUS_PENDING in this driver appears to be broken. (*IRP specified.*)
0x20A	Fatal error	The caller of **IoFreeIrp** is freeing an IRP that is still in use. (*Original IRP and IRP in use specified.*)
0x20B	Fatal error	The caller of **IoFreeIrp** is freeing an IRP that is still in use. (*IRP specified.*)
0x20C	Fatal error	The caller of **IoFreeIrp** is freeing an IRP that is still queued against a thread. (*IRP specified.*)
0x20D	Fatal error	The caller of **IoInitializeIrp** has passed an IRP that was allocated with **IoAllocateIrp**. This is illegal and unnecessary, and has caused a quota leak. Check the documentation for **IoReuseIrp** if this IRP is being recycled.
0x20E	Non-fatal error	A PNP IRP has an invalid status. (Any PNP IRP must have its status initialized to STATUS_NOT_SUPPORTED.) (*IRP specified.*)
0x20F	Non-fatal error	A Power IRP has an invalid status. (Any Power IRP must have its status initialized to STATUS_NOT_SUPPORTED.) (*IRP specified.*)
0x210	Non-fatal error	A WMI IRP has an invalid status. (Any WMI IRP must have its status initialized to STATUS_NOT_SUPPORTED.) (*IRP specified.*)
0x211	Non-fatal error	The caller has forwarded an IRP while skipping a device object in the stack. The caller is probably sending IRPs to the PDO instead of to the device returned by **IoAttachDeviceToDeviceStack**. (*IRP specified.*)
0x212	Non-fatal error	The caller has trashed or has not properly copied the IRP's stack. (*IRP specified.*)
0x213	Non-fatal error	The caller has changed the status field of an IRP it does not understand. (*IRP specified.*)
0x214	Non-fatal error	The caller has changed the information field of an IRP it does not understand. (*IRP specified.*)
0x215	Non-fatal error	A non-successful non-STATUS_NOT_SUPPORTED IRP status for IRP_MJ_PNP is being passed down stack. (*IRP specified.*) Failed PNP IRPs must be completed.
0x216	Non-fatal error	The previously-set IRP_MJ_PNP status has been converted to STATUS_NOT_SUPPORTED. (*IRP specified.*)
0x217	Non-fatal error	The driver has not handled a required IRP. The driver must update the status of the IRP to indicate whether or not it has been handled. (*IRP specified.*)
0x218	Non-fatal error	The driver has responded to an IRP that is reserved for other device objects elsewhere in the stack. (*IRP specified.*)
0x219	Non-fatal error	A non-successful non-STATUS_NOT_SUPPORTED IRP status for IRP_MJ_POWER is being passed down stack. (*IRP specified.*) Failed POWER IRPs must be completed.
0x21A	Non-fatal error	The previously-set IRP_MJ_POWER status has been converted to STATUS_NOT_SUPPORTED. (*IRP specified.*)
0x21B	Non-fatal error	A driver has returned a suspicious status. This is probably due to an uninitialized variable bug in the driver. (*IRP specified.*)
0x21C	Warning	The caller has copied the IRP stack but not set a completion routine. This is inefficient — use **IoSkipCurrentIrpStackLocation** instead. (*IRP specified.*)
0x21D	Fatal error	An IRP dispatch handler has not properly detached from the stack below it upon receiving a remove IRP. (*Device object, dispatch routine, and IRP specified.*)
0x21E	Fatal error	An IRP dispatch handler has not properly deleted its device object upon receiving a remove IRP. (*Device object, dispatch routine, and IRP specified.*)
0x21F	Non-fatal error	A driver has not filled out a dispatch routine for a required IRP major function. (*IRP specified.*)
0x220	Non-fatal error	IRP_MJ_SYSTEM_CONTROL has been completed by someone other than the ProviderId. This IRP should either have been completed earlier or should have been passed down. (*IRP specified, along with the device object where it was targeted.*)
0x221	Fatal error	An IRP dispatch handler for a PDO has deleted its device object, but the hardware has not been reported as missing in a bus relations query. (*Device object, dispatch routine, and IRP specified.*)
0x222	Fatal error	A Bus Filter's IRP dispatch handler has detached upon receiving a remove IRP when the PDO is still alive. Bus Filters must clean up in **FastIoDetach** callbacks. (*Device object, dispatch routine, and IRP specified.*)
0x223	Fatal error	An IRP dispatch handler for a bus filter has deleted its device object, but the PDO is still present. Bus filters must clean up in **FastIoDetach** callbacks. (*Device object, dispatch routine, and IRP specified.*)
0x224	Fatal error	An IRP dispatch handler has returned a status that is inconsistent with the IRP's **IoStatus.Status** field. (*Dispatch handler routine, IRP, IRP's IoStatus.Status, and returned Status specified*.)
0x225	Non-fatal error	An IRP dispatch handler has returned a status that is illegal (0xFFFFFFFF). This is probably due to an uninitialized stack variable. To debug this error, use the **ln (List Nearest Symbols)** command with the specified address.
0x226	Fatal error	An IRP dispatch handler has returned without passing down or completing this IRP, or someone forgot to return STATUS_PENDING. (*IRP specified.*)
0x227	Fatal error	An IRP completion routine is in pageable code. (This is never permitted.) (*Routine and IRP specified.*)
0x228	Non-fatal error	A driver's completion routine has not marked the IRP pending if the **PendingReturned** field was set in the IRP passed to it. This may cause Windows to hang, especially if an error is returned by the stack. (*Routine and IRP specified.*)
0x229	Fatal error	A cancel routine has been set for an IRP that is currently being processed by drivers lower in the stack, possibly stomping their cancel routine. (*Routine and IRP specified.*)
0x22A	Non-fatal error	The physical device object (PDO) has not responded to a required IRP. (*IRP specified.*)
0x22B	Non-fatal error	The physical device object (PDO) has forgotten to fill out the device relation list with the PDO for the **TargetDeviceRelation** query. (*IRP specified.*)
0x22C	Fatal error	The code implementing the **TargetDeviceRelation** query has not called **ObReferenceObject** on the PDO. (*IRP specified.*)
0x22D	Non-fatal error	The caller has completed a IRP_MJ_PNP it didn't understand instead of passing it down. (*IRP specified.*)
0x22E	Non-fatal error	The caller has completed a successful IRP_MJ_PNP instead of passing it down. (*IRP specified.*)
0x22F	Non-fatal error	The caller has completed an untouched IRP_MJ_PNP (instead of passing the IRP down), or non-PDO has failed the IRP using illegal value of STATUS_NOT_SUPPORTED. (*IRP specified.*)
0x230	Non-fatal error	The caller has completed an IRP_MJ_POWER it didn't understand instead of passing it down. (*IRP specified.*)

0x231	Fatal error	The caller has completed a successful IRP_MJ_POWER instead of passing it down. (*IRP specified.*)
0x232	Non-fatal error	The caller has completed an untouched IRP_MJ_POWER (instead of passing the IRP down), or non-PDO has failed the IRP using illegal value of STATUS_NOT_SUPPORTED. (*IRP specified.*)
0x233	Non-fatal error	The version field of the query capabilities structure in a query capabilities IRP was not properly initialized. (*IRP specified.*)
0x234	Non-fatal error	The size field of the query capabilities structure in a query capabilities IRP was not properly initialized. (*IRP specified.*)
0x235	Non-fatal error	The address field of the query capabilities structure in a query capabilities IRP was not properly initialized to -1. (*IRP specified.*)
0x236	Non-fatal error	The UI Number field of the query capabilities structure in a query capabilities IRP was not properly initialized to -1. (*IRP specified.*)
0x237	Fatal error	A driver has sent an IRP that is restricted for system use only. (*IRP specified.*)
0x238	Warning	The caller of **IoInitializeIrp** has passed an IRP that was allocated with **IoAllocateIrp**. This is illegal, unnecessary, and negatively impacts performance in normal use. If this IRP is being recycled, see **IoReuseIrp** in the Windows Driver Kit.
0x239	Warning	The caller of **IoCompleteRequest** is completing an IRP that has never been forwarded via a call to **IoCallDriver** or **PoCallDriver**. This may be a bug. (*IRP specified.*)
0x23A	Fatal error	A driver has forwarded an IRP at an IRQL that is illegal for this major code. (*IRP specified.*)
0x23B	Non-fatal error	The caller has changed the status field of an IRP it does not understand. (*IRP specified.*)

The following table lists additional **I/O Verification** errors that can appear in Windows XP and later. Some of these errors will only be revealed if **Enhanced I/O Verification** is activated.

I/O Error Code	Severity	Cause of Error
0x23C	Fatal error	A driver has completed an IRP without setting the cancel routine in the IRP to NULL. (*IRP specified.*)
0x23D	Non-fatal error	A driver has returned STATUS_PENDING but did not mark the IRP pending via a call to **IoMarkIrpPending**. (*IRP specified.*)
0x23E	Non-fatal error	A driver has marked an IRP pending but didn't return STATUS_PENDING. (*IRP specified.*)
0x23F	Fatal error	A driver has not inherited the DO_POWER_PAGABLE bit from the stack it has attached to. (*Device object specified.*)
0x240	Fatal error	A driver is attempting to delete a device object that has already been deleted via a prior call to **IoDeleteDevice**.
0x241	Fatal error	A driver has detached its device object during a surprise remove IRP. (*IRP and device object specified.*)
0x242	Fatal error	A driver has deleted its device object during a surprise remove IRP. (*IRP and device object specified.*)
0x243	Fatal error	A driver has failed to clear the DO_DEVICE_INITIALIZING flag at the end of **AddDevice**. (*Device object specified.*)
0x244	Fatal error	A driver has not copied either the DO_BUFFERED_IO or the DO_DIRECT_IO flag from the device object it is attaching to. (*Device object specified.*)
0x245	Fatal error	A driver has set both the DO_BUFFERED_IO and the DO_DIRECT_IO flags. These flags are mutually exclusive. (*Device object specified.*)
0x246	Fatal error	A driver has failed to copy the **DeviceType** field from the device object it is attaching to. (*Device object specified.*)
0x247	Fatal error	A driver has failed an IRP that cannot legally be failed. (*IRP specified.*)
0x248	Fatal error	A driver has added a device object that is not a PDO to a device relations query. (*IRP and device object specified.*)
0x249	Non-fatal error	A driver has enumerated two child PDOs that returned identical Device IDs. (*Both device objects specified.*)
0x24A	Fatal error	A driver has mistakenly called a file I/O function with IRQL not equal to PASSIVE_LEVEL.
0x24B	Fatal error	A driver has completed an IRP_MN_QUERY_DEVICE_RELATIONS request of type **TargetDeviceRelation** as successful, but did not properly fill out the request or forward the IRP to the underlying hardware stack. *(Device object specified.)*
0x24C	Non-fatal error	A driver has returned STATUS_PENDING but did not mark the IRP pending by a call to **IoMarkIrpPending**. (*IRP specified.*)
0x24D	Fatal error	A driver has passed an invalid device object to a function that requires a PDO. *(Device object specified.)*

Cause

See the description of each code in the Parameters section for a description of the cause.

Resolving the Problem

This bug check can only occur when Driver Verifier has been instructed to monitor one or more drivers. If you did not intend to use Driver Verifier, you should deactivate it. You might consider removing the driver which caused this problem as well.

If you are the driver writer, use the information obtained through this bug check to fix the bugs in your code.

For full details on Driver Verifier, see the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCA: PNP_DETECTED_FATAL_ERROR

The PNP_DETECTED_FATAL_ERROR bug check has a value of 0x000000CA. This indicates that the Plug and Play Manager encountered a severe error, probably as a result of a problematic Plug and Play driver.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 identifies the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x1	Address of newly-reported PDO	Address of older PDO which has been duplicated	Reserved	**Duplicate PDO:** A specific instance of a driver has enumerated multiple PDOs with identical device ID and unique IDs.
0x2	Address of purported PDO	Address of driver object	Reserved	**Invalid PDO:** An API which requires a PDO has been called with random memory, or with an FDO, or with a PDO which hasn't been initialized. (An uninitialized PDO is one that has not been returned to Plug and Play by **QueryDeviceRelation** or **QueryBusRelations**.)
0x3	Address of PDO whose IDs were queried	Address of ID buffer	**1:** DeviceID **2:** UniqueID **3:** HardwareIDs **4:** CompatibleIDs	**Invalid ID:** An enumerator has returned an ID which contains illegal characters or isn't properly terminated. (IDs must contain only characters in the ranges 0x20 - 0x2B and 0x2D - 0x7F.)
0x4	Address of PDO with DOE_DELETE_PENDING set	Reserved	Reserved	**Invalid enumeration of deleted PDO:** An enumerator has returned a PDO which it had previously deleted using **IoDeleteDevice**.
0x5	Address of PDO	Reserved	Reserved	**PDO freed while linked in devnode tree:** The object manager reference count on a PDO dropped to zero while the devnode was still linked in the tree. (This usually indicates that the driver is not adding a reference when returning the PDO in a query IRP.)
0x8	Address of PDO whose stack returned the invalid bus relation	Total number of PDOs returned as bus relations	The index (zero-based) at which the first NULL PDO was found	**Null pointer returned as a bus relation:** One or more of the devices present on the bus is a NULL PDO.
0x9	Connection type that was passed	Reserved	Reserved	**Invalid connection type passed to IoDisconnectInterruptEx:** A driver has passed an invalid connection type to **IoDisconnectInterruptEx**. The connection type passed to this routine must match the one returned by a corresponding successful call to **IoConnectInterruptEx**.
0xA	Driver object	IRQL after returning from driver callback	Combined APC disable count after returning from driver callback	**Incorrect notify callback behavior:** A driver failed to preserve IRQL or combined APC disable count across a Plug 'n' Play notification.
0xB	Related PDO	Removal relations	Reserved	**Deleted PDO reported as relation:** One of the removal relations for the device being removed has already been deleted.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCB: DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS

The DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS bug check has a value of 0x000000CB. This indicates that a driver or the I/O manager failed to release locked pages after an I/O operation.

Parameters

The four parameters listed in the message can have two possible meanings.

If a driver locked these pages, the parameters have the following meaning.

Parameter	Description
1	Calling address in the driver that locked the pages
2	Caller of the calling address in driver that locked the pages
3	Address of the MDL containing the locked pages
4	Number of locked pages

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

If the I/O manager locked these pages, the parameters have the following meaning.

Parameter	Description
1	Address of the dispatch routine of the top driver on the stack to which the IRP was sent
2	Address of the device object of the top driver on the stack to which the IRP was sent
3	Address of the MDL containing the locked pages
4	Number of locked pages

Comments

This bug check is issued only if the registry value **\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\TrackLockedPages** is equal to DWORD 1. If this value is not set, the system will issue the less-informative bug check 0x76 (PROCESS_HAS_LOCKED_PAGES).

Starting with Windows Vista, this bug check can also be issued by Driver Verifier when the Pool Tracking option is enabled.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCC: PAGE_FAULT_IN_FREED_SPECIAL_POOL

The PAGE_FAULT_IN_FREED_SPECIAL_POOL bug check has a value of 0x000000CC. This indicates that the system has referenced memory which was earlier freed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

The system has accessed memory in the special pool which was already freed by a driver. This usually indicates a system-driver synchronization problem.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Comments

This cannot be protected by a **try - except** handler — it can only be protected by a probe.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCD: PAGE_FAULT_BEYOND_END_OF_ALLOCATION

The PAGE_FAULT_BEYOND_END_OF_ALLOCATION bug check has a value of 0x000000CD. This indicates that the system accessed memory beyond the end of some driver's pool allocation.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

The driver allocated *n* bytes of memory from the special pool. Subsequently, the system referenced more than *n* bytes from this pool. This usually indicates a system-driver synchronization problem.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Comments

This cannot be protected by a **try - except** handler — it can only be protected by a probe.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCE: DRIVER_UNLOADED_WITHOUT_CANCELLING_PENDING_OPERATIONS

The DRIVER_UNLOADED_WITHOUT_CANCELLING_PENDING_OPERATIONS bug check has a value of 0x000000CE. This indicates that a driver failed to cancel pending operations before unloading.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

This driver failed to cancel lookaside lists, DPCs, worker threads, or other such items before unload.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xCF: TERMINAL_SERVER_DRIVER_MADE_INCORRECT_MEMORY_REFEREN

The TERMINAL_SERVER_DRIVER_MADE_INCORRECT_MEMORY_REFERENCE bug check has a value of 0x000000CF. This indicates that a driver has been incorrectly ported to the terminal server.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

The driver is referencing session space addresses from the system process context. This probably results from the driver queuing an item to a system worker thread.

This driver needs to comply with Terminal Server's memory management rules.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD0: DRIVER_CORRUPTED_MMPOOL

The DRIVER_CORRUPTED_MMPOOL bug check has a value of 0x000000D0. This indicates that the system attempted to access invalid memory at a process IRQL that was too high.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write
4	Address that referenced memory

Cause

The kernel attempted to access pageable memory (or perhaps completely invalid memory) when the IRQL was too high. The ultimate cause of this problem is almost certainly a driver that has corrupted the system pool.

In most cases, this bug check results if a driver corrupts a large allocation (PAGE_SIZE or larger). Smaller allocations result in bug check 0xC5 (DRIVER_CORRUPTED_EXPOOL).

Resolving the Problem

If you have recently installed any new software, check to see if it is properly installed. Check for updated drivers on the manufacturer's website.

To debug this error, use the special pool option of Driver Verifier. If this fails to reveal the driver that caused the error, use the Global Flags utility to enable the special pool by pool tag.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

An alternate method is to open the **\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management** registry key. In this key, create or edit the **ProtectNonPagedPool** value, and set it equal to DWORD 1. Then reboot. Then the system will unmap all freed nonpaged pool. This will prevent drivers from corrupting the pool. (This does not protect the pool from DMA hardware, however.)

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD1: DRIVER_IRQL_NOT_LESS_OR_EQUAL

The DRIVER_IRQL_NOT_LESS_OR_EQUAL bug check has a value of 0x000000D1. This indicates that a kernel-mode driver attempted to access pageable memory at a process IRQL that was too high.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write **8:** Execute
4	Address that referenced memory

Cause

A driver tried to access an address that is pageable (or that is completely invalid) while the IRQL was too high.

This bug check is usually caused by drivers that have used improper addresses.

If the first parameter has the same value as the fourth parameter, and the third parameter indicates an execute operation, this bug check was likely caused by a driver that was

trying to execute code when the code itself was paged out. Possible causes for the page fault include the following:

- The function was marked as pageable and was running at an elevated IRQL (which includes obtaining a lock).
- The function call was made to a function in another driver, and that driver was unloaded.
- The function was called by using a function pointer that was an invalid pointer.

Resolving the Problem

To begin debugging, use a kernel debugger to get a stack trace.

If the problem is caused by the driver that you are developing, make sure that the function that was executing at the time of the bug check is not marked as pageable or does not call any other inline functions that could be paged out.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD2: BUGCODE_ID_DRIVER

The BUGCODE_ID_DRIVER bug check has a value of 0x000000D2. This indicates that a problem occurred with an NDIS driver.

Parameters

Before this bug check occurs, a message is sent to the DbgPrint buffer. If a debugger is connected, this message will be displayed.

This message indicates the type of violation. The meanings of the bug check parameters depend on this message.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Message and Cause
Address of the miniport block	Number of bytes requested	0	1	**Allocating shared memory at raised IRQL.** A driver called **NdisMAllocateSharedMemory** with IRQL >= DISPATCH_LEVEL.
Address of the miniport block	The *Status* value submitted to **NdisMResetComplete**	The *AddressingReset* value submitted to **NdisMResetComplete**	0	**Completing reset when one is not pending.** A driver called **NdisMResetComplete**, but no reset was pending.
Address of the miniport block	Memory page containing address being freed	Address of shared memory signature	Virtual address being freed	**Freeing shared memory not allocated.** A driver called **NdisMFreeSharedMemory** or **NdisMFreeSharedMemoryAsync** with an address that is not located in NDIS shared memory.
Address of the miniport block	Address of the packet that is incorrectly included in the packet array	Address of the packet array	Number of packets in the array	**Indicating packet not owned by it.** The miniport's packet array is corrupt.
Address of the MiniBlock	Address of the driver object	0	0	**NdisAddDevice: AddDevice** called with a **MiniBlock** that is not on the **NdisMiniDriverList.**
Address of the MiniBlock	The MiniBlock's reference count	0	0	**NdisMUnload: MiniBlock** is getting unloaded but it is still on **NdisMiniDriverList.**
Address of the miniport block	Memory page	Wrapper context	Address of shared memory signature	**Overwrote past allocated shared memory.** The address being written to is not located in NDIS shared memory.

In the following instances of this bug check, the meaning of the parameters depends on the message *and* on the value of Parameter 4.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Message and Cause
Address of the miniport block	Address of the miniport interrupt	Address of the miniport timer queue	1	**Unloading without deregistering interrupt.** A miniport driver failed its initialization without deregistering its interrupt.
Address of the miniport block	Address of the miniport timer queue	Address of the miniport interrupt	2	**Unloading without deregistering interrupt.** A miniport driver did not deregister its interrupt during the halt process.
Address of the miniport block	Address of the miniport interrupt	Address of the miniport timer queue	1	**Unloading without deregistering timer.** A miniport driver failed its initialization without successfully canceling all its timers.
Address of the miniport block	Address of the miniport timer queue	Address of the miniport interrupt	2	**Unloading without deregistering timer.** A miniport driver halted without successfully canceling all its timers.

Comments

This bug check code only occurs on Windows 2000 and Windows XP. In Windows Server 2003 and later, the corresponding code is bug check 0x7C (BUGCODE_NDIS_DRIVER).

On the checked build of Windows, only the **Allocating Shared Memory at Raised IRQL** and **Completing Reset When One is Not Pending** instances of this bug check can occur. All the other instances of bug check 0xD2 are replaced with ASSERTs. See Breaking Into the Debugger for details.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD3: DRIVER_PORTION_MUST_BE_NONPAGED

The DRIVER_PORTION_MUST_BE_NONPAGED bug check has a value of 0x000000D3. This indicates that the system attempted to access pageable memory at a process IRQL that was too high.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write
4	Address that referenced memory

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

This bug check is usually caused by drivers that have incorrectly marked their own code or data as pageable.

Resolving the Problem

To begin debugging, use a kernel debugger to get a stack trace.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD4: SYSTEM_SCAN_AT_RAISED_IRQL_CAUGHT_IMPROPER_DRIVER_UNLC

The SYSTEM_SCAN_AT_RAISED_IRQL_CAUGHT_IMPROPER_DRIVER_UNLOAD bug check has a value of 0x000000D4. This indicates that a driver did not cancel pending operations before unloading.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL at time of reference
3	**0:** Read **1:** Write
4	Address that referenced memory

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

This driver failed to cancel lookaside lists, DPCs, worker threads, or other such items before unload. Subsequently, the system attempted to access the driver's former location at a raised IRQL.

Resolving the Problem

To begin debugging, use a kernel debugger to get a stack trace. If the driver that caused the error has been identified, activate Driver Verifier and attempt to replicate this bug.

For full details on Driver Verifier, see the Windows Driver Kit.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD5: DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL

The DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL bug check has a value of 0x000000D5. This indicates that a driver has referenced memory which was earlier freed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

The Driver Verifier **Special Pool** option has caught the driver accessing memory which was earlier freed.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Comments

This cannot be protected by a **try - except** handler — it can only be protected by a probe.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD6: DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION

The DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION bug check has a value of 0x000000D6. This indicates the driver accessed memory beyond the end of its pool allocation.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory address referenced
2	**0:** Read **1:** Write
3	Address that referenced memory (if known)
4	Reserved

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

The driver allocated *n* bytes of memory and then referenced more than *n* bytes. The Driver Verifier **Special Pool** option detected this violation.

For information about the special pool, consult the Driver Verifier section of the Windows Driver Kit.

Comments

This cannot be protected by a **try - except** handler — it can only be protected by a probe.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD7: DRIVER_UNMAPPING_INVALID_VIEW

The DRIVER_UNMAPPING_INVALID_VIEW bug check has a value of 0x000000D7. This indicates a driver is trying to unmap an address that was not mapped.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address to unmap
2	**1:** The view is being unmapped **2:** The view is being committed
3	0
4	0

Comments

The driver that caused the error can be determined from the stack trace.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD8: DRIVER_USED_EXCESSIVE_PTES

The DRIVER_USED_EXCESSIVE_PTES bug check has a value of 0x000000D8. This indicates that there are no more system page table entries (PTE) remaining.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Pointer to the name of the driver that caused the error (Unicode string), or zero
2	Number of PTEs used by the driver that caused the error (if Parameter 1 is nonzero)
3	Total free system PTEs
4	Total system PTEs

If the driver responsible for the error can be identified, its name is printed on the blue screen and stored in memory at the location (PUNICODE_STRING) **KiBugCheckDriver**.

Cause

This is usually caused by a driver not cleaning up its memory use properly. Parameter 1 shows the driver which has consumed the most PTEs. The call stack will reveal which driver actually caused the bug check.

Resolving the Problem

Both drivers may need to be fixed. The total number of system PTEs may also need to be increased.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xD9: LOCKED_PAGES_TRACKER_CORRUPTION

The LOCKED_PAGES_TRACKER_CORRUPTION bug check has a value of 0x000000D9. This indicates that the internal locked-page tracking structures have been corrupted.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x01	The address of the internal lock tracking structure	The address of the memory descriptor list	The number of pages locked for the current process	The MDL is being inserted twice on the same process list.

0x02	The address of the internal lock tracking structure	The address of the memory descriptor list	The number of pages locked for the current process	The MDL is being inserted twice on the systemwide list.
0x03	The address of the first internal tracking structure found	The address of the internal lock tracking structure	The address of the memory descriptor list	The MDL was found twice in the process list when being freed.
0x04	The address of the internal lock tracking structure	The address of the memory descriptor list	0	The MDL was found in the systemwide list on free after it was removed.

Cause

The error is indicated by the value of Parameter 1.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDA: SYSTEM_PTE_MISUSE

The SYSTEM_PTE_MISUSE bug check has a value of 0x000000DA. This indicates that a page table entry (PTE) routine has been used in an improper way.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x01	The address of the internal lock tracking structure	The address of the memory descriptor list	The address of the duplicate internal lock tracking structure	The mapping being freed is a duplicate.
0x02	The address of the internal lock tracking structure	The number of mappings that the system expects to free	The number of mappings that the driver is requesting to free	The number of mappings being freed is incorrect.
0x03	The address of the first internal tracking structure found	The mapping address that the system expects to free	The mapping address that the driver is requesting to free	The mapping address being freed is incorrect.
0x04	The address of the internal lock tracking structure	The page frame number that the system expects should be first in the MDL	The page frame number that is currently first in the MDL	The first page of the mapped MDL has changed since the MDL was mapped.
0x05	The address of the first internal tracking structure found	The virtual address that the system expects to free	The virtual address that the driver is requesting to free	The start virtual address in the MDL being freed has changed since the MDL was mapped.
0x06	The MDL specified by the driver	The virtual address specified by the driver	The number of mappings to free (specified by the driver)	The MDL being freed was never (or is currently not) mapped.
0x07	The initial mapping	The number of mappings	Reserved	(*Windows 2000 only*) The mapping range is being double-allocated.
0x08	The initial mapping	The number of mappings the caller is freeing	The number of mappings the system thinks should be freed	(*Windows 2000 only*) The caller is asking to free an incorrect number of mappings.
0x09	The initial mapping	The number of mappings that the caller is freeing	The mapping index that the system thinks is already free	(*Windows 2000 only*) The caller is asking to free several mappings, but at least one of them is not allocated.
0x0A	**1:** The driver requested "bug check on failure" in the MDL. **0:** The driver did not request "bug check on failure" in the MDL.	The number of mappings that the caller is allocating	The type of mapping pool requested	(*Windows 2000 only*) The caller is asking to allocate zero mappings.
0x0B	The corrupt mapping	The number of mappings that the caller is allocating	The type of mapping pool requested	(*Windows 2000 only*) The mapping list was already corrupt at the time of this allocation. The corrupt mapping is located below the lowest possible mapping address.
0x0C	The corrupt mapping	The number of mappings that the caller is allocating	The type of mapping pool requested	(*Windows 2000 only*) The mapping list was already corrupt at the time of this allocation. The corrupt mapping is located above the lowest possible mapping address.
0x0D	The initial mapping	The number of mappings that the caller is freeing	The type of mapping pool	(*Windows 2000 only*) The caller is trying to free zero mappings.

0x0E	The initial mapping	The number of mappings that the caller is freeing	The type of mapping pool	(*Windows 2000 only)* The caller is trying to free mappings, but the guard mapping has been overwritten.
0x0F	The non-existent mapping	The number of mappings that the caller is trying to free	The type of mapping pool being freed	(*Windows 2000 only)* The caller is trying to free a non-existent mapping. The non-existent mapping is located below the lowest possible mapping address.
0x10	The non-existent mapping	The number of mappings the caller is trying to free	The type of mapping pool being freed	(*Windows 2000 only)* The caller is trying to free a non-existent mapping. The non-existent mapping is located above the highest possible mapping address.
0x11	The non-existent mapping	The number of mappings that the caller is trying to free	The type of mapping pool being freed	(*Windows 2000 only)* The caller is trying to free a non-existent mapping. The non-existent mapping is at the base of the mapping address space.
0x100	The number of mappings being requested	The caller's identifying tag	The address of the routine that called the caller of this routine	(*Windows XP and later only)* The caller requested 0 mappings.
0x101	The first mapping address	The caller's identifying tag	The owner's identifying tag	(*Windows XP and later only)* A caller is trying to free a mapping address range that it does not own.
0x102	The first mapping address	The caller's identifying tag	Reserved	(*Windows XP and later only)* The mapping address space that the caller is trying to free is apparently empty.
0x103	The address of the invalid mapping	The caller's identifying tag	The number of mappings in the mapping address space	(*Windows XP and later only)* The mapping address space that the caller is trying to free is still reserved. **MmUnmapReservedMapping** must be called before **MmFreeMappingAddress**.
0x104	The first mapping address	The caller's identifying tag	The owner's identifying tag	(*Windows XP and later only)* The caller is attempting to map an MDL to a mapping address space that it does not own.
0x105	The first mapping address	The caller's identifying tag	Reserved	(*Windows XP and later only)* The caller is attempting to map an MDL to an invalid mapping address space. The caller has mostly likely specified an invalid address.
0x107	The first mapping address	The address of the non-empty mapping	The last mapping address	(*Windows XP and later only)* The caller is attempting to map an MDL to a mapping address space that has not been properly reserved. The caller should have called **MmUnmapReservedMapping** prior to calling **MmMapLockedPagesWithReservedMapping**
0x108	The first mapping address	The caller's identifying tag	The owner's identifying tag	(*Windows XP and later only)* The caller is attempting to unmap a locked mapping address space that it does not own.
0x109	The first mapping address	The caller's identifying tag	Reserved	(*Windows XP and later only)* The caller is attempting to unmap a locked virtual address space that is apparently empty.
0x10A	The first mapping address	The number of mappings in the locked mapping address space	The number of mappings to unmap	(*Windows XP and later only)* The caller is attempting to unmap more mappings than actually exist in the locked mapping address space.
0x10B	The first mapping address	The caller's identifying tag	The number of mappings to unmap	(*Windows XP and later only)* The caller is attempting to unmap a portion of a locked virtual address space that is not currently mapped.
0x10C	The first mapping address	The caller's identifying tag	The number of mappings to unmap	(*Windows XP and later only)* The caller is not unmapping the entirety of the locked mapping address space.
0x200	The first mapping address	0	0	(*Windows XP and later only)* The caller is attempting to reserve a mapping address space that contains no mappings.
0x201 0x202	The first mapping address to reserve	The address of the mapping that has already been reserved	The number of mappings to reserve	(*Windows XP and later only)* One of the mappings that the caller is attempting to reserve has already been reserved.
0x300	The first mapping address to release	0	0	(*Windows XP and later only)* The caller is attempting to release a mapping address space that contains no mappings.
0x301	The address of the mapping	0	0	(*Windows XP and later only)* The caller is attempting to release a mapping that it is not permitted to release.
0x303	The first mapping address	The number of mappings to release	0	(*Windows XP and later only)* The caller is attempting to release a mapping address range that was not reserved.
0x304	The first mapping address	The number of mappings to release	0	(*Windows XP and later only)* The caller is attempting to release a mapping address range that begins in the middle of a different allocation.
0x305	The first mapping address	The number of mappings that the caller is trying to release	The number of mappings that should be released	(*Windows XP and later only)* The caller is attempting to release the wrong number of mappings.
0x306	The first mapping address	The free mapping address	The number of mappings to release	(*Windows XP and later only)* One of the mappings that the caller is attempting to release is already free.
0x400	The base address of the I/O space mapping	The number of pages to be freed	0	(*Windows XP and later only*) The caller is trying to free an I/O space mapping that the system is unaware of.

Cause

The error is indicated by the value of Parameter 1.

A stack trace will identify the driver that caused the error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDB: DRIVER_CORRUPTED_SYSPTES

The DRIVER_CORRUPTED_SYSPTES bug check has a value of 0x000000DB. This indicates that an attempt was made to touch memory at an invalid IRQL, probably due to corruption of system PTEs.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Memory referenced
2	IRQL
3	**0:** Read **1:** Write
4	Address in code which referenced memory

Cause

A driver tried to access pageable (or completely invalid) memory at too high of an IRQL. This bug check is almost always caused by drivers that have corrupted system PTEs.

Resolving the Problem

If this bug check occurs, the culprit can be detected by editing the registry. In the **\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management** registry key, create or edit the **TrackPtes** value, and set it equal to DWORD 3. Then reboot. The system will then save stack traces, and if the driver commits the same error, the system will issue bug check 0xDA (SYSTEM_PTE_MISUSE). Then the stack trace will identify the driver that caused the error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDC: DRIVER_INVALID_STACK_ACCESS

The DRIVER_INVALID_STACK_ACCESS bug check has a value of 0x000000DC. This indicates that a driver accessed a stack address that lies below the stack pointer of the stack's thread.

Parameters

None

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDE: POOL_CORRUPTION_IN_FILE_AREA

The POOL_CORRUPTION_IN_FILE_AREA bug check has a value of 0x000000DE. This indicates that a driver has corrupted pool memory that is used for holding pages destined for disk.

Parameters

None

Cause

When the Memory Manager dereferenced the file, it discovered this corruption in pool memory.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDF: IMPERSONATING_WORKER_THREAD

The IMPERSONATING_WORKER_THREAD bug check has a value of 0x000000DF. This indicates that a workitem did not disable impersonation before it completed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The worker routine that caused this error
2	The parameter passed to this worker routine
3	A pointer to the work item
4	Reserved

Cause

A worker thread was impersonating another process, and failed to disable impersonation before it returned.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE0: ACPI_BIOS_FATAL_ERROR

The ACPI_BIOS_FATAL_ERROR bug check has a value of 0x000000E0. This indicates that one of your computer components is faulty.

Parameters

The parameters for this bug check are issued by the BIOS, not by Windows. They can only be interpreted by the hardware vendor.

Cause

Your computer's BIOS has reported that a component in the system is so faulty that there is no way for Windows to operate. The BIOS is indicating that there is no alternative but to issue a bug check.

Resolving the Problem

You can determine which component is faulty by running the diagnostic disk or tool that was included with your computer.

If you do not have this tool, you must contact the system vendor and report this error message to them. They will be able to help you correct this hardware problem. This enables Windows to operate.

Microsoft cannot address this error. Only the hardware vendor is qualified to analyze it.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE1: WORKER_THREAD_RETURNED_AT_BAD_IRQL

The WORKER_THREAD_RETURNED_AT_BAD_IRQL bug check has a value of 0x000000E1. This indicates that a worker thread completed and returned with IRQL >= DISPATCH_LEVEL.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Address of the worker routine
2	IRQL that the worker thread returned at
3	Work item parameter
4	Work item address

Cause

A worker thread completed and returned with IRQL >= DISPATCH_LEVEL.

Resolving the Problem

To find the driver that caused the error, use the **ln (List Nearest Symbols)** debugger command:

```
kd> ln address
```

where *address* is the worker routine address given in Parameter 1.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE2: MANUALLY_INITIATED_CRASH

The MANUALLY_INITIATED_CRASH bug check has a value of 0x000000E2. This indicates that the user deliberately initiated a crash dump from either the kernel debugger or the keyboard.

Parameters

None

Comments

For more information about manually-initiated crash dumps, see Forcing a System Crash. .

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE3: RESOURCE_NOT_OWNED

The RESOURCE_NOT_OWNED bug check has a value of 0x000000E3. This indicates that a thread tried to release a resource it did not own.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Address of resource
2	Address of thread
3	Address of owner table (if it exists)
4	Reserved

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE4: WORKER_INVALID

The WORKER_INVALID bug check has a value of 0x000000E4. This indicates that memory that should not contain an executive worker item does contain such an item, or that a currently active worker item was queued.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the code position.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x0	Address of worker item	Start of pool block	End of pool block	An active worker item was freed.
0x1	Address of worker item	Queue number	0	An active worker item was queued.
0x2	Address of worker item	Address of I/O worker routine	0	A queued I/O worker item was freed.
0x3	Address of worker item	Address of invalid object	0	An attempt was made to initialize an I/O worker item with an invalid object.

Cause

This is usually caused by a driver freeing memory which still contains an executive worker item.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE6: DRIVER_VERIFIER_DMA_VIOLATION

The DRIVER_VERIFIER_DMA_VIOLATION bug check has a value of 0x000000E6. This is the bug check code for all Driver Verifier **DMA Verification** violations.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 is the only parameter of interest. This parameter identifies the exact violation. If a debugger is attached, an informative message is displayed in the debugger.

Parameter 1	Cause of Error and Debugger Message
0x00	This code can represent two kinds of errors: 1. The driver tried to flush too many bytes to the end of the map register file. *The number of bytes permitted and the number of bytes attempted are displayed.* 2. Windows has run out of contiguous map registers. *The number of map registers needed and the largest block of contiguous map registers is displayed.*
0x01	The performance counter has decreased. *The old and new values of the counter are displayed.*
0x02	The performance counter has increased too fast. *The counter value is displayed in the debugger.*
0x03	The driver freed too many DMA common buffers. Usually this means it freed the same buffer two times.
0x04	The driver freed too many DMA adapter channels. Usually this means it freed the same adapter channel two times.
0x05	The driver freed too many DMA map registers. Usually this means it freed the same map register two times. *The number of active map registers is displayed.*
0x06	The driver freed too many DMA scatter/gather lists. Usually this means it freed the same scatter/gather list two times. *The number of lists allocated and the number of lists freed is displayed.*
0x07	The driver tried to release the adapter without first freeing all its common buffers. *The adapter address and the number of remaining buffers is displayed.*
0x08	The driver tried to release the adapter without first freeing all adapter channels, common buffers, or scatter/gather lists. *The adapter address and the number of remaining items is displayed.*
0x09	The driver tried to release the adapter without first freeing all map registers. *The adapter address and the number of remaining map registers is displayed.*
0x0A	The driver tried to release the adapter without first freeing all its scatter/gather lists. *The adapter address and the number of remaining scatter/gather lists is displayed.*
0x0B	HV_TOO_MANY_ADAPTER_CHANNELSThe driver has allocated too many adapter channels at the same time. . (Only one adapter channel is permitted per adapter.)
0x0C	The driver tried to allocate too many map registers at the same time. *The number requested and the number allowed are displayed.*
0x0D	The driver did not flush its adapter buffers. *The number of bytes that the driver tried to map and the maximum number of bytes allowed are displayed.*
0x0E	The driver tried a DMA transfer without locking the buffer. The buffer in question was in paged memory. *The address of the MDL is displayed.*
0x0F	The driver or the hardware wrote outside its allocated DMA buffer. *The nature of the error (overrun or underrun) is displayed, as well as the relevant addresses.*
0x10	The driver tried to free its map registers while some were still mapped. *The number of map registers still mapped is displayed.*
0x11	The driver has too many outstanding reference counts for the adapter. *The number of reference counts and the adapter address are displayed.*
0x13	The driver called a DMA routine at an improper IRQL. *The required IRQL and the actual IRQL are displayed.*
0x14	The driver called a DMA routine at an improper IRQL. *The required IRQL and the actual IRQL are displayed.*
0x15	The driver tried to allocate too many map registers. *The number requested and the number allowed are displayed.*
0x16	The driver tried to flush a buffer that is not mapped. *The address of the buffer is displayed.*
0x18	The driver tried a DMA operation by using an adapter that was already released and no longer exists. *The adapter address is displayed.*
0x19	The driver passed a null DMA_ADAPTER value to a HAL routine.
0x1B	The driver passed an address and MDL to a HAL routine. However, this address is not within the bounds of this MDL. *The address passed and the address of the MDL are displayed.*
0x1D	The driver tried to map an address range that was already mapped. *The address range and the current mapping for that range are displayed.*
0x1E	The driver called **HalGetAdapter**. This function is obsolete — you must use **IoGetDmaAdapter** instead.
0x1F	HV_BAD_MDLThe driver referenced an invalid system address — either before the first MDL, or after the end of the first MDL, or by using a transfer length that is longer than the MDL buffer and crosses a page boundary within the MDL. . *Either the invalid address and the first MDL address, or the MDL address and the extra transfer length are displayed.*
0x20	The driver tried to flush a map register that hasn't been mapped. *The map register base, flushing address, and MDL are displayed.*
0x21	The driver tried to map a zero-length buffer for transfer.

Cause

See the description of each code in the Parameters section for a description of the cause.

Resolving the Problem

This bug check can only occur when Driver Verifier has been instructed to monitor one or more drivers. If you did not intend to use Driver Verifier, you should deactivate it. You might also consider removing the driver that caused this problem.

If you are the driver writer, use the information obtained through this bug check to fix the bugs in your code.

The Driver Verifier **DMA Verification** option is only available in Windows XP and later versions. For full details on Driver Verifier, see the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE7: INVALID_FLOATING_POINT_STATE

The INVALID_FLOATING_POINT_STATE bug check has a value of 0x000000E7. This indicates that a thread's saved floating-point state is invalid.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates which validity check failed. Parameter 4 is not used. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Cause of Error
0x0	The flags field	0	The saved context flags field is invalid. Either FLOAT_SAVE_VALID is not set, or some reserved bits are nonzero.
0x1	The saved IRQL	The current IRQL	The current processor's IRQL is not the same as when the floating-point context was saved.
0x2	The saved address of the thread that owns this floating-point context	The current thread	The saved context does not belong to the current thread.

Cause

While restoring the previously-saved floating-point state for a thread, the state was found to be invalid.

Parameter 1 indicates which validity check failed.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE8: INVALID_CANCEL_OF_FILE_OPEN

The INVALID_CANCEL_OF_FILE_OPEN bug check has a value of 0x000000E8. This indicates that an invalid file object was passed to **IoCancelFileOpen**.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The file object passed to **IoCancelFileOpen**
2	The device object passed to **IoCancelFileOpen**
3	Reserved
4	Reserved

Cause

The file object passed to **IoCancelFileOpen** is invalid. It should have reference of one. The driver that called **IoCancelFileOpen** is at fault.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xE9: ACTIVE_EX_WORKER_THREAD_TERMINATION

The ACTIVE_EX_WORKER_THREAD_TERMINATION bug check has a value of 0x000000E9. This indicates that an active executive worker thread is being terminated.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The exiting ETHREAD
2	Reserved
3	Reserved
4	Reserved

Cause

An executive worker thread is being terminated without having gone through the worker thread rundown code. This is forbidden; work items queued to the **ExWorkerQueue** must not terminate their threads.

A stack trace should indicate the cause.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xEA: THREAD_STUCK_IN_DEVICE_DRIVER

The THREAD_STUCK_IN_DEVICE_DRIVER bug check has a value of 0x000000EA. This indicates that a thread in a device driver is endlessly spinning.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	A pointer to the stuck thread object
2	A pointer to the DEFERRED_WATCHDOG object
3	A pointer to the offending driver name
4	*In the kernel debugger:* The number of times the "intercepted" bug check 0xEA was hit *On the blue screen:* 1

Cause

A device driver is spinning in an infinite loop, most likely waiting for hardware to become idle.

This usually indicates problem with the hardware itself, or with the device driver programming the hardware incorrectly. Frequently, this is the result of a bad video card or a bad display driver.

Resolving the Problem

Use the **.thread (Set Register Context)** command together with Parameter 1. Then use **kb (Display Stack Backtrace)** to find the location where the thread is stuck.

If the kernel debugger is already connected and running when Windows detects a time-out condition. Then **DbgBreakPoint** will be called instead of **KeBugCheckEx**. A detailed message will be printed to the debugger. See Sending Output to the Debuggefor more information.

This message will include what would have been the bug check parameters. Because no actual bug check was issued, the **.bugcheck (Display Bug Check Data)** command will not be useful. The four parameters can also be retrieved from Watchdog's global variables by using **dd watchdog!g_WdBugCheckData L5**" on a 32-bit system, or **dq watchdog!g_WdBugCheckData L5**" on a 64-bit system.

Debugging this error in an interactive manner such as this will enable you to find an offending thread, set breakpoints in it, and then use **g (Go)** to return to the spinning code to debug it further.

On multiprocessor machines (OS build 3790 or earlier), you can hit a time out if the spinning thread is interrupted by a hardware interrupt and an ISR or DPC routine is running at the time of the bug check. This is because the time out's work item can be delivered and handled on the second CPU and the same time. If this occurs, you must look deeper at the offending thread's stack to determine the spinning code which caused the time out to occur. Use the **dds (Display Words and Symbols)** command to do this.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xEB: DIRTY_MAPPED_PAGES_CONGESTION

The DIRTY_MAPPED_PAGES_CONGESTION bug check has a value of 0x000000EB. This indicates that no free pages are available to continue operations.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The total number of dirty pages
2	The number of dirty pages destined for the page file
3	*Windows Server 2003 only:* The size of the nonpaged pool available at the time of the bug check (in pages) Windows Vista *and later versions:* Reserved
4	*Windows Server 2003 only:* The number of transition pages that are currently stranded Windows Vista *and later versions:* The most recent modified write error status

Cause

The file system driver stack has deadlocked and most of the modified pages are destined for the file system. Because the file system is non-operational, the system has crashed because none of the modified pages can be reused without losing data. Any file system or filter driver in the stack may be at fault.

To see general memory statistics, use the **!vm 3** extension.

This bug check can occur for any of the following reasons:

- A driver has blocked, deadlocking the modified or mapped page writers. Examples of this include mutex deadlocks or accesses to paged out memory in file system drivers or filter drivers. This indicates a driver bug.

  If Parameter 1 or Parameter 2 is large, this is a possibility. Use **!vm 3**.

- A storage driver is not processing requests. Examples of this are stranded queues and unresponsive drives. This indicates a driver bug.

  If Parameter 1 or Parameter 2 is large, this is a possibility. Use **!process 0 7**.

- *Windows Server 2003 only:* Not enough pool is available for the storage stack to write out modified pages. This indicates a driver bug.

  If Parameter 3 is small, this is a possibility. Use **!vm** and **!poolused 2**.

-

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xEC: SESSION_HAS_VALID_SPECIAL_POOL_ON_EXIT

The SESSION_HAS_VALID_SPECIAL_POOL_ON_EXIT bug check has a value of 0x000000EC. This indicates that a session unload occurred while a session driver still held memory.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The session ID
2	The number of special pool pages that are leaking
3	Reserved
4	Reserved

Cause

This error is caused by a session driver not freeing its special pool allocations prior to a session unload. This indicates a bug in *win32k.sys*, *atmfd.dll*, *rdpdd.dll*, or a video driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xED: UNMOUNTABLE_BOOT_VOLUME

The UNMOUNTABLE_BOOT_VOLUME bug check has a value of 0x000000ED. This indicates that the I/O subsystem attempted to mount the boot volume and it failed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The device object of the boot volume
2	The status code from the file system that describes why it failed to mount the volume
3	Reserved
4	Reserved

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xEF: CRITICAL_PROCESS_DIED

The CRITICAL_PROCESS_DIED bug check has a value of 0x000000EF. This indicates that a critical system process died.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The process object
2	Reserved
3	Reserved
4	Reserved

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF1: SCSI_VERIFIER_DETECTED_VIOLATION

The SCSI_VERIFIER_DETECTED_VIOLATION bug check has a value of 0x000000F1. This is the bug check code for all Driver Verifier **SCSI Verification** violations.

Parameters

The four bug check parameters are displayed on the blue screen. Parameter 1 identifies the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x1000	First argument passed	Second argument passed	Reserved	The miniport driver passed bad arguments to **ScsiPortInitialize**.
0x1001	Delay, in microseconds	Reserved	Reserved	The miniport driver called **ScsiPortStallExecution** and specified a delay greater than 0.1 second, stalling the processor too long.
0x1002	Address of routine that took too long	Address of miniport's HW_DEVICE_EXTENSION	Duration of the routine, in microseconds	A miniport routine called by the port driver took longer than 0.5 second to execute. (0.5 seconds is the limit for most routines. However, the **HwInitialize** routine is allowed 5 seconds, and the **FindAdapter** routine is exempt.)
0x1003	Address of miniport's HW_DEVICE_EXTENSION	Address of the SRB	Reserved	The miniport driver completed a request more than once.
0x1004	Address of the SRB	Address of miniport's HW_DEVICE_EXTENSION	Reserved	The miniport driver completed a request with an invalid SRB status.
0x1005	Address of miniport's HW_DEVICE_EXTENSION	Address of LOGICAL_UNIT_EXTENSION	Reserved	The miniport driver called **ScsiPortNotification** to ask for **NextLuRequest**, but an untagged request is still active.
0x1006	Address of miniport's HW_DEVICE_EXTENSION	Invalid virtual address	Reserved	The miniport driver passed an invalid virtual address to **ScsiPortGetPhysicalAddress**. (This usually means the address supplied doesn't map to the common buffer area.)
0x1007	Address of ADAPTER_EXTENSION	Address of miniport's HW_DEVICE_EXTENSION	Reserved	The reset hold period for the bus ended, but the miniport driver still has outstanding requests.

Cause

See the description of each code in the Parameters section for an explanation of the cause.

Resolving the Problem

This bug check can only occur when Driver Verifier has been instructed to monitor one or more drivers. If you did not intend to use Driver Verifier, you should deactivate it. You might consider removing the driver which caused this problem as well.

If you are the driver writer, use the information obtained through this bug check to fix the bugs in your code.

The Driver Verifier **SCSI Verification** option is only available in Windows XP and later. For full details on Driver Verifier, see the Windows Driver Kit.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF3: DISORDERLY_SHUTDOWN

The DISORDERLY_SHUTDOWN bug check has a value of 0x000000F3. This indicates that Windows was unable to shut down due to lack of memory.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The total number of dirty pages
2	The number of dirty pages destined for the page file
3	*Windows Server 2003 only*: The size of the nonpaged pool available at the time of the bug check (in pages) Windows Vista *and later:* Reserved
4	*Windows Server 2003 only:* The current shut down stage Windows Vista *and later:* The most recent modified write error status

Cause

Windows attempted to shut down, but there were no free pages available to continue operations.

Because applications were not terminated and drivers were not unloaded, they continued to access pages even after the modified writer had terminated. This causes the system to run out of pages, since the page files could be used.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF4: CRITICAL_OBJECT_TERMINATION

The CRITICAL_OBJECT_TERMINATION bug check has a value of 0x000000F4. This indicates that a process or thread crucial to system operation has unexpectedly exited or been terminated.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The terminating object type: **0x3:** Process **0x6:** Thread
2	The terminating object
3	The process image file name
4	Pointer to an ASCII string containing an explanatory message

Cause

Several processes and threads are necessary for the operation of the system. When they are terminated for any reason, the system can no longer function.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF5: FLTMGR_FILE_SYSTEM

The FLTMGR_FILE_SYSTEM bug check has a value of 0x000000F5. This indicates that an unrecoverable failure occurred in the Filter Manager.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the type of violation. The meaning of the other parameters depends on the value of Parameter 1.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of error
0x66	Pointer to the callback data structure for the operation.	0	0	The minifilter returned FLT_PREOP_SUCCESS_WITH_CALLBACK or FLT_PREOP_SYNCHRONIZE from a preoperation callback, but did not register a corresponding postoperation callback.
0x67	Pointer to the callback data structure for the operation.	0	Error NTSTATUS code for the operation	An internal object ran out of space, and the system is unable to allocate new space.
0x68	Handle for the object.	0	NTSTATUS code returned by **ObReferenceObjectByHandle**	Unexpected failure referencing an object.
0x6A	File object pointer for the file.	0	0	The file-open or file-create request could not be canceled, because one or more handles have been created for the file.
0x6B	Frame ID	0	Thread	Invalid BACKPOCKET IRPCTRL state.
0x6C	Frame ID	BackPocket List	Thread	Too many nested PageFaults for BACKPOCKETED IRPCTR.
0x6D	Address of the minifilter's context structure	Address of the CONTEXT_NODE structure	0	The context structure was dereferenced too many times. This means that the reference count on the Filter Manager's CONTEXT_NODE structure went to zero while it was still attached to its associated object.
0x6E	Address of the minifilter's context structure	Address of the CONTEXT_NODE structure	0	The context structure was referenced after being freed.

Cause

The cause of the problem is indicated by the value of Parameter 1. See the table in the Parameters section.

Resolving the Problem

If Parameter 1 equals **0x66**, you can debug this problem by verifying that your minifilter driver has registered a post-operation callback for this operation. The current operation can be found in the callback data structure. (See Parameter 2.) Use the **!fltkd.cbd** debugger extension.

If Parameter 1 equals **0x67**, you should verify that you do not have a nonpaged pool leak somewhere in the system.

If Parameter 1 equals **0x6A**, make sure that your minifilter driver does not reference this file object (see Parameter 2) to get a handle at any point during your minifilter's processing of this operation.

If Parameter 1 equals **0x6B** or **0x6C**, then a non-recoverable internal state error has occurred which will cause the operating system to bug check.

If Parameter 1 equals **0x6D**, make sure that your minifilter driver does not call **FltReleaseContext** too many times for the given context (see Parameter 2).

If Parameter 1 equals 0x6E, make sure that your minifilter driver does not call **FltReferenceContext** after the given context has been deleted (see Parameter 2).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF6: PCI_VERIFIER_DETECTED_VIOLATION

The PCI_VERIFIER_DETECTED_VIOLATION bug check has a value of 0x000000F6. This indicates that an error occurred in the BIOS or another device being verified by the PCI driver.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 is the only parameter of interest; this identifies the nature of the failure detected.

Parameter 1	Cause of Error
0x01	An active bridge was reprogrammed by the BIOS during a docking event.
0x02	The PMCSR register was not updated within the spec-mandated time.
0x03	A driver has written to Windows-controlled portions of a PCI device's configuration space.

Cause

The PCI driver detected an error in a device or BIOS being verified.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF7: DRIVER_OVERRAN_STACK_BUFFER

The DRIVER_OVERRAN_STACK_BUFFER bug check has a value of 0x000000F7. This indicates that a driver has overrun a stack-based buffer.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The actual security check cookie from the stack
2	The expected security check cookie
3	The bit-complement of the expected security check cookie
4	0

Cause

A driver overran a stack-based buffer (or local variable) in a way that would have overwritten the function's return address and jumped back to an arbitrary address when the function returned.

This is the classic "buffer overrun" hacking attack. The system has been brought down to prevent a malicious user from gaining complete control of it.

Resolving the Problem

Use the **kb (Display Stack Backtrace)** command to get a stack trace.

The last routine on the stack before the buffer overrun handlers and bug check call is the one that overran its local variable.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF8: RAMDISK_BOOT_INITIALIZATION_FAILED

The RAMDISK_BOOT_INITIALIZATION_FAILED bug check has a value of 0x000000F8. This indicates that an initialization failure occurred while attempting to boot from the RAM disk.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Indicates the cause of the failure. **1:** No LoaderXIPRom descriptor was found in the loader memory list. **2:** Unable to open the RAM disk driver (*ramdisk.sys* or \Device\Ramdisk). **3:** FSCTL_CREATE_RAM_DISK failed. **4:** Unable to create GUID string from binary GUID. **5:** Unable to create symbolic link pointing to the RAM disk device.
2	NTSTATUS code
3	0
4	0

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xF9: DRIVER_RETURNED_STATUS_REPARSE_FOR_VOLUME_OPEN

The DRIVER_RETURNED_STATUS_REPARSE_FOR_VOLUME_OPEN bug check has a value of 0x000000F9. This indicates that a driver returned STATUS_REPARSE to an IRP_MJ_CREATE request with no trailing names.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The device object that was opened
2	The device object to which the IRP_MJ_CREATE request was issued
3	Address of the Unicode string containing the new name of the file (to be reparsed)
4	Information returned by the driver for the IRP_MJ_CREATE request

Comments

STATUS_REPARSE should be returned only for IRP_MJ_CREATE requests with trailing names, as that indicates the driver is supporting name spaces.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xFA: HTTP_DRIVER_CORRUPTED

The HTTP_DRIVER_CORRUPTED bug check has a value of 0x000000FA. This indicates that the HTTP kernel driver (*Http.sys*) has reached a corrupted state and cannot recover.

Parameters

The four bug check parameters are displayed on the blue screen. Parameter 1 identifies the exact state of the HTTP kernel driver.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x1	Address of work item	Name of the file that contains the work item check	Line number of the work item check within the file	A work item is invalid. This will eventually result in thread pool corruption and an access violation.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xFC: ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY

The ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY bug check has a value of 0x000000FC. This indicates that an attempt was made to execute non-executable memory.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The virtual address whose execution was attempted
2	The contents of the page table entry (PTE)
3	Reserved
4	Reserved

Resolving the Problem

When possible, the Unicode string of the driver name that attempted to execute non-executable memory is printed on the bug check screen and is also saved in **KiBugCheckDriver**. Otherwise, the driver in question can often be found by running a stack trace and then reviewing the current instruction pointer.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xFD: DIRTY_NOWRITE_PAGES_CONGESTION

The DIRTY_NOWRITE_PAGES_CONGESTION bug check has a value of 0x000000FD. This indicates that there are no free pages available to continue basic system operations.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Total number of dirty pages
2	Number of non-writeable dirty pages
3	Reserved
4	Most recently modified write-error status

Cause

This bug check usually occurs because the component that owns the modified non-writeable pages failed to write out these pages after marking the relevant files as "do not write" to memory management. This indicates a driver bug.

Resolving the Problem

For more information about which driver is causing the problem, use the **!vm 3** extension, followed by **!memusage 1** .

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xFE: BUGCODE_USB_DRIVER

The BUGCODE_USB_DRIVER bug check has a value of 0x000000FE. This indicates that an error has occurred in a Universal Serial Bus (USB) driver.

Parameters

The four bug check parameters are displayed on the blue screen. Parameter 1 identifies the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of Error
0x1	Reserved	Reserved	Reserved	An internal error has occurred in the USB stack.
0x2	Address of the pending IRP	Address of the IRP that was passed in	Address of the USB request block (URB) that caused the error	The USB client driver has submitted a URB that is still attached to another IRP that is pending in the bus driver.
0x3	Reserved	Reserved	Reserved	The USB miniport driver has generated a bug check. This usually happens in response to a catastrophic hardware failure.
0x4	Address of the IRP	Address of the URB	Reserved	The caller has submitted an IRP that is already pending in the USB bus driver.
0x5	Device extension pointer of the host controller	PCI vendor, product id for the controller	Pointer to endpoint data structure	A hardware failure has occurred due to a bad physical address found in a hardware data structure. This is not due to a driver bug.
Ox6	Object address	Signature that was expected	Reserved	An Internal data structure (object) has been corrupted.
0x7	Pointer to usbport.sys debug log	Message string	File name	Please consult the provided message string for details.
0x8	Reserved Type	Reserved	Reserved	Reserved

Cause

See the description of each code in the Parameters section for an explanation of the cause.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xFF: RESERVE_QUEUE_OVERFLOW

The RESERVE_QUEUE_OVERFLOW bug check has a value of 0x000000FF. This indicates that an attempt was made to insert a new item into a reserve queue, causing the queue to overflow.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the reserve queue

2	The size of the reserve queue
3	0
4	0

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x100: LOADER_BLOCK_MISMATCH

The LOADER_BLOCK_MISMATCH bug check has a value of 0x00000100. This indicates that either the loader block is invalid, or it does not match the system that is being loaded.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	3
2	The size of the loader black extension
3	The major version of the loader block
4	The minor version of the loader block

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x101: CLOCK_WATCHDOG_TIMEOUT

The CLOCK_WATCHDOG_TIMEOUT bug check has a value of 0x00000101. This indicates that an expected clock interrupt on a secondary processor, in a multi-processor system, was not received within the allocated interval.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Clock interrupt time-out interval, in nominal clock ticks
2	0
3	The address of the processor control block (PRCB) for the unresponsive processor
4	0

Cause

The specified processor is not processing interrupts. Typically, this occurs when the processor is nonresponsive or is deadlocked.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x103: MUP_FILE_SYSTEM

[This is preliminary documentation and subject to change.]

The MUP_FILE_SYSTEM bug check has a value of 0x00000103. This bug check indicates that the multiple UNC provider (MUP) has encountered invalid or unexpected data. As a result, the MUP cannot channel a remote file system request to a network redirector, the Universal Naming Convention (UNC) provider.

Parameters

These bug check parameters are displayed on the blue screen. Parameter 1 identifies the type of violation.

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of error
0x1	The address of the pending	The address of the file object whose file	The address of the device object.	The MUP could not locate the file context that corresponds to a file object. This typically indicates that the MUP is seeing an I/O request for a file object for which MUP did not see a

	IRP.	context could not be found.		corresponding IRP_MJ_CREATE request. The likely cause of this bug check is a filter driver error.
0x2	The address of the expected file context.	The address that was actually retrieved from the file object.	Reserved	A file context is known to exist for the file object, but was not what was expected (for example, it might be NULL).
0x3	The address of the IRP context.	The IRP completion status code.	The driver object of the UNC provider that completed the IRP (might be NULL).	The IRP completion status was unexpected or invalid. This bug check occurs only when you are using a Checked Build of Windows and should only be caused by file system filter drivers that are attached to legacy network redirectors. Legacy redirectors use **FsRtlRegisterUncProvider** to register with MUP. This bug check detects filter drivers that return an NTSTATUS that is not STATUS_SUCCESS in IRP_MJ_CLEANUP or IRP_MJ_CLOSE requests.
0x4	Address of the IRP	Address of the file object	The file context for the file object	An I/O operation was started on a file object before the create request for the file object was completed.

Comments

The MUP maintains context information on a per-file object basis for all file objects it handles.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x104: AGP_INVALID_ACCESS

The AGP_INVALID_ACCESS bug check has a value of 0x00000104. This indicates that the GPU wrote to a range of Accelerated Graphics Port (AGP) memory that had not previously been committed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Offset (in ULONG) within the AGP verifier page to the first ULONG data that is corrupted
2	0
3	0
4	0

Cause

Typically, this bug check is caused by an unsigned or improperly tested video driver. It can also be caused by an old BIOS.

Resolving the Problem

Check for display driver and computer BIOS updates.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x105: AGP_GART_CORRUPTION

The AGP_GART_CORRUPTION bug check has a value of 0x00000105. This indicates that the Graphics Aperture Remapping Table (GART) is corrupt.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The base address (virtual) of the GART
2	The offset into the GART where the corruption occurred
3	The base address (virtual) of the GART cache (a copy of the GART)
4	0

Cause

This bug check is typically caused by improper direct memory access (DMA) by a driver.

Resolving the Problem

Enable Driver Verifier for any unsigned drivers. Remove them or disable them one by one until the erring driver is identified.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x106: AGP_ILLEGALLY_REPROGRAMMED

The AGP_ILLEGALLY_REPROGRAMMED bug check has a value of 0x00000106. This indicates that the Accelerated Graphics Port (AGP) hardware has been reprogrammed by an unauthorized agent.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The originally programmed AGP command register value
2	The current command register value
3	0
4	0

Cause

This bug check is typically caused by an unsigned, or improperly tested, video driver.

Resolving the Problem

Check the video manufacturer's Web site for updated display drivers or use VGA mode.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x108: THIRD_PARTY_FILE_SYSTEM_FAILURE

The THIRD_PARTY_FILE_SYSTEM_FAILURE bug check has a value of 0x00000108. This indicates that an unrecoverable problem has occurred in a third-party file system or file system filter.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Identifies the file system that failed. Possible values include: **1:** Polyserve (*Psfs.sys*)
2	The address of the exception record.
3	The address of the context record.
4	Reserved.

Cause

One possible cause of this bug check is disk corruption. Corruption in the third-party file system or bad blocks (sectors) on the hard disk can induce this error. Corrupted SCSI and IDE drivers can also adversely affect the Windows operating system's ability to read and write to disk, thus causing the error.

Another possible cause is depletion of nonpaged pool memory. If the nonpaged pool is completely depleted, this error can stop the system.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command with Parameter 3, and then use **kb (Display Stack Backtrace)**.

To resolve a disk corruption problem: Check Event Viewer for error messages from SCSI, IDE, or other disk controllers in the system that might help pinpoint the device or driver that is causing the error. Try disabling any virus scanners, backup programs, or disk defragmenter tools that continually monitor the system. You should also run hardware diagnostics supplied by the file system or the file system filter manufacturer.

To resolve a nonpaged pool memory depletion problem: Add new physical memory to the computer. This will increase the quantity of nonpaged pool memory available to the kernel.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x109: CRITICAL_STRUCTURE_CORRUPTION

The CRITICAL_STRUCTURE_CORRUPTION bug check has a value of 0x00000109. This indicates that the kernel has detected critical kernel code or data corruption.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Reserved
2	Reserved
3	Reserved
4	The type of the corrupted region. (*See the following table later on this page.*)

The value of Parameter 4 indicates the type of corrupted region.

Parameter 4	Type of Corrupted Region, Type of Corruption, or Type of Action Taken That Caused the Corruption
0x0	A generic data region
0x1	A function modification or the Itanium-based function location
0x2	A processor interrupt dispatch table (IDT)
0x3	A processor global descriptor table (GDT)
0x4	A type-1 process list corruption
0x5	A type-2 process list corruption
0x6	A debug routine modification
0x7	A critical MSR modification

Cause

There are generally three different causes for this bug check:

1. A driver has inadvertently, or deliberately, modified critical kernel code or data. Microsoft Windows Server 2003 with Service Pack 1 (SP1) and later versions of Windows for x64-based computers do not allow the kernel to be patched except through authorized Microsoft-originated hot patches. For more information, see Patching Policy for x64-based Systems.
2. A developer attempted to set a normal kernel breakpoint using a kernel debugger that was not attached when the system was started. Normal breakpoints (**bp**) can only be set if the debugger is attached at start time. Processor breakpoints (**ba**) can be set at any time.
3. A hardware corruption occurred. For example, the kernel code or data could have been stored in memory that failed.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10A: APP_TAGGING_INITIALIZATION_FAILED

The APP_TAGGING_INITIALIZATION_FAILED bug check has a value of 0x0000010A.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10C: FSRTL_EXTRA_CREATE_PARAMETER_VIOLATION

The FSRTL_EXTRA_CREATE_PARAMETER_VIOLATION bug check has a value of 0x0000010C. This indicates that a violation was detected in the File system Run-time library (FsRtl) Extra Create Parameter (ECP) package.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The type of violation. (*See the following table later on this page for more details*).
2	0
3	The address of the ECP.
4	The starting address of the ECP list.

The value of Parameter 1 indicates the type of violation.

Parameter 1	Type of Violation
0x1	The ECP signature is invalid, due to either a bad pointer or memory corruption.
0x2	The ECP has undefined flags set.
0x3	The ECP was not allocated by the FsRtl.
0x4	The ECP has flags set that are illegal for a parameter passed by a create caller.
0x5	The ECP is corrupted; its size is smaller than the header size.
0x6	The ECP that is being freed has non-empty list pointers; it might still be part of an ECP list.
0x11	The ECP list signature is invalid, due to either a bad pointer or memory corruption.
0x12	The ECP list has undefined flags set.
0x13	The ECP list was not allocated by the FsRtl.
0x14	The ECP list has flags set that are illegal for a parameter list passed by a create caller.
0x15	The ECP list passed by the create caller is empty.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10D: WDF_VIOLATION

The WDF_VIOLATION bug check has a value of 0x0000010D. This indicates that Kernel-Mode Driver Framework (KMDF) detected that Windows found an error in a framework-based driver.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 indicates the specific error code of the bug check. Parameter 4 is reserved.

Parameter 1	Parameter 2	Parameter 3	Cause of Error
0x1	Pointer to a WDF_POINTER_ROUTINE_TIMED_OUT_DATA structure	Reserved	A framework-based driver has timed out during a power operation. This typically means that the device stack did not set the DO_POWER_PAGABLE bit and a driver attempted a pageable operation after the paging device stack was powered down.
0x2	Reserved	Reserved	An attempt is being made to acquire a lock that is currently being held.
0x3	WDFREQUEST handle	The number of outstanding references that remain on both buffers	Windows Driver Framework Verifier has encountered a fatal error. In particular, an I/O request was completed, but a framework request object cannot be deleted because there are outstanding references to the input buffer, the output buffer, or both.
0x4	Reserved	The caller's address	A NULL parameter was passed to a function that required a non-NULL value.
0x5	The handle value passed in	Reserved	A framework object handle of the incorrect type was passed to a framework object method.
0x6			*See table below.*
0x7	The handle of the framework object	Reserved	A driver attempted to delete a framework object incorrectly by calling **WdfObjectDereference** to delete a handle instead of calling **WdfObjectDelete**.
0x8	The handle of the DMA transaction object	Reserved	An operation occurred on a DMA transaction object while it was not in the correct state.
0x9			*Currently unused.*
0xA	A pointer to a WDF_QUEUE_FATAL_ERROR_DATA structure	Reserved	A fatal error has occurred while processing a request that is currently in the queue.

0xB			*See table below.*
0xC	WDFDEVICE handle	Pointer to new PnP IRP	A new state-changing PnP IRP arrived while the driver was processing another state-changing PnP IRP.
0xD	WDFDEVICE handle	Pointer to power IRP	A device's power policy owner received a power IRP that it did not request. There might be multiple power policy owners, but only one is allowed. A KMDF driver can change power policy ownership by calling **WdfDeviceInitSetPowerPolicyOwnership**.
0xE	IRQL at which the event callback function was called.	IRQL at which the event callback function returned.	An event callback function did not return at the same IRQL at which it was called. The callback function changed the IRQL directly or indirectly (for example, by acquiring a spinlock, which raises IRQL to DISPATCH_LEVEL, but not releasing the spinlock).
0xF	Address of an event callback function.	Reserved	An event callback function entered a critical region, but it did not leave the critical region before returning.

If Parameter 1 is equal to 0x6, then a fatal error was made in handling a WDF request. In this case, Parameter 2 further specifies the type of fatal error that has been made, as defined by the enumeration WDF_REQUEST_FATAL_ERROR.

Parameter 2	Parameter 3	Cause of Error
0x1	The address of the IRP	No more I/O stack locations are available to format the underlying IRP.
0x2	The WDF request handle value	An attempt was made to format a framework request object that did not contain an IRP.
0x3	The WDF request handle value	The driver attempted to send a framework request that has already been sent to an I/O target.
0x4	A pointer to a WDR_REQUEST_FATAL_ERROR_INFORMATION_LENGTH_MISMATCH_DATA structure that contains a pointer to the IRP, a WDF request handle value, an IRP major function, and the number of bytes attempted to be written	The driver has completed a framework request, but has written more bytes to the output buffer than are specified in the IRP.

If Parameter 1 is equal to 0xB, then an attempt to acquire or release a lock was invalid. In this case, Parameter 3 further specifies the error that has been made.

Parameter 2	Parameter 3	Cause of Error
The handle value	0x0	A handle passed to **WdfObjectAcquireLock** or **WdfObjectReleaseLock** represents an object that does not support synchronization locks.
A WDF spin lock handle	0x1	The spin lock is being released by a thread that did not acquire it.

Cause

See the description of each code in the Parameters section for an explanation of the cause.

Resolving the Problem

Typically, the dump file will yield further information on the driver that caused this bug check.

If Parameter 1 is equal to **0x2**, examine the caller's stack to determine the lock in question.

If Parameter 1 is equal to **0x3**, the driver's Kernel-Mode Driver Framework error log will include details about the outstanding references.

If Parameter 1 is equal to **0x4**, use the **In** debugger command with the value of *Parameter 3* as its argument to determine which function requires a non-NULL parameter.

If Parameter 1 is equal to **0x7**, use the **!wdfkd.wdfhandle** *Parameter 2* extension command to determine the handle type.

If Parameter 1 is equal to **0xA**, then the WDF_QUEUE_FATAL_ERROR_DATA structure will indicate either the problematic request or the queue handle. It will also indicate the NTSTATUS, if not STATUS_SUCCESS, when available.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10E: VIDEO_MEMORY_MANAGEMENT_INTERNAL

The VIDEO_MEMORY_MANAGEMENT_INTERNAL bug check has a value of 0x0000010E. This indicates that the video memory manager has encountered a condition that it is unable to recover from.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 is the only parameter of interest; this identifies the exact violation. Values for Parameter 1 that do not appear in this table must be individually examined.

Parameter 1	Cause of Error
0x1	An attempt was made to rotate a non-rotate range.
0x2	An attempt was made to destroy a non-empty process heap.
0x3	An attempt to unmap from an aperture segment failed.
0x4	A rotation in a must-succeed path failed.
0x5	A deferred command failed.
0x6	An attempt was made to reallocate resources for an allocation that was having its eviction canceled.
0x7	An invalid attempt was made to defer free usage.
0x8	The split direct memory access (DMA) buffer contains an invalid reference.
0x9	An attempt to evict an allocation failed.
0xA	An invalid attempt to use a pinned allocation was made.
0xB	A driver returned an invalid error code from **BuildPagingBuffer**.
0xC	A resource leak was detected in a segment.
0xD	A segment is being used improperly.
0xE	An attempt to map an allocation into an aperture segment failed.
0xF	A driver returned an invalid error code from **AcquireSwizzlingRange**.
0x10	A driver returned an invalid error code from **ReleaseSwizzlingRange**.
0x11	An invalid attempt to use an aperture segment was made.
0x12	A driver overflowed the provided DMA buffer.
0x13	A driver overflowed the provided private data buffer.
0x14	An attempt to purge all segments failed.
0x15	An attempt was made to free a virtual address descriptor (VAD) that was still in the rotated state
0x16	A driver broke the guaranteed DMA buffer model contract.
0x17	An unexpected system command failure occurred.
0x18	An attempt to release a pinned allocation's resource failed.
0x19	A driver failed to patch a DMA buffer.
0x1A	The owner of a shared allocation was freed.
0x1B	An attempt was made to release an aperture range that is still in use.

Cause

This bug check is usually caused by a video driver behaving improperly.

Resolving the Problem

If the problem persists, check Windows Update for an updated video driver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x10F: RESOURCE_MANAGER_EXCEPTION_NOT_HANDLED

The RESOURCE_MANAGER_EXCEPTION_NOT_HANDLED bug check has a value of 0x0000010F. This indicates that the kernel transaction manager detected that a kernel-mode resource manager has raised an exception in response to a direct call-back. The resource manager is in an unexpected and unrecoverable state.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the exception record
2	The address of the context record
3	The address of the exception code
4	The address of the resource manager

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x111: RECURSIVE_NMI

[This is preliminary documentation and subject to change.]

The RECURSIVE_NMI bug check has a value of 0x00000111. This bug check indicates that a non-maskable-interrupt (NMI) occurred while a previous NMI was in progress.

Comments

This bug check occurs when there is an error in the system management interrupt (SMI) code, and an SMI interrupts an NMI and enables interrupts. Execution then continues with NMIs enabled, and another NMI interrupts the NMI in progress.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x112: MSRPC_STATE_VIOLATION

The MSRPC_STATE_VIOLATION bug check has a value of 0x00000112. This indicates that the *Msrpc.sys* driver has initiated a bug check.

Parameters

The following parameters are displayed on the blue screen. Parameters 1 and 2 are the only parameters of interest. Parameter 1 indicates the state violation type; the value for Parameter 2 is determined by the value of Parameter 1.

Parameter 1	Parameter 2	Cause of Error
0x01	The exception code	A non-continuable exception was continued by the caller.
0x02	The error	The advanced local procedure call (ALPC) returned an invalid error.
0x03	The session to the server	The caller unloaded the Microsoft remote procedure call (MSRPC) driver while it was still in use. It is likely that open binding handles remain.
0x04 and 0x05	The session to the server	An invalid close command was received from the ALPC.
0x06	The binding handle	An attempt was made to bind a remote procedure call (RPC) handle a second time.
0x07	The binding handle	An attempt was made to perform an operation on a binding handle that was not bound.
0x08	The binding handle	An attempt was made to set security information on a binding handle that was already bound.
0x09	The binding handle	An attempt was made to set an option on a binding handle that was already bound.
0x0A	The call object	An attempt was made to cancel an invalid asynchronous remote procedure call.
0x0B	The call object	An attempt was made to push on an asynchronous pipe call when it was not expected.
0x0C and 0x0E	The pipe object	An attempt was made to push on an asynchronous pipe without waiting for notification.
0x0F	The pipe object	An attempt was made to synchronously terminate a pipe a second time.
0x15	The object closest to the error	An RPC internal error occurred.
0x16	Reserved	Two causally ordered calls were issued in an order that cannot be enforced by the RPC.
0x17	The call object	A server manager routine did not unsubscribe from notifications prior to completing the call.
0x18	The async handle	An invalid operation on the asynchronous handle occurred.

Cause

The most common cause of this bug check is that the caller of the *Msrpc.sys* driver violated the state semantics for such a call.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x113: VIDEO_DXGKRNL_FATAL_ERROR

The VIDEO_DXGKRNL_FATAL_ERROR bug check has a value of 0x00000113. This indicates that the dxg kernel has detected a violation.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x114: VIDEO_SHADOW_DRIVER_FATAL_ERROR

The VIDEO_SHADOW_DRIVER_FATAL_ERROR bug check has a value of 0x00000114. This indicates that the shadow driver has detected a violation.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x115: AGP_INTERNAL

The AGP_INTERNAL bug check has a value of 0x00000115. This indicates that the accelerated graphics port (AGP) driver has detected a violation.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x116: VIDEO_TDR_ERROR

The VIDEO_TDR_ ERROR bug check has a value of 0x00000116. This indicates that an attempt to reset the display driver and recover from a timeout failed.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The pointer to the internal TDR recovery context, if available.
2	A pointer into the responsible device driver module (for example, the owner tag).
3	The error code of the last failed operation, if available.
4	Reserved.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x117: VIDEO_TDR_TIMEOUT_DETECTED

The VIDEO_TDR_TIMEOUT_DETECTED bug check has a value of 0x00000117. This indicates that the display driver failed to respond in a timely fashion.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The pointer to the internal TDR recovery context, if available.
2	A pointer into the responsible device driver module (for example, the owner tag).
3	The secondary driver-specific bucketing key.
4	Reserved.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x119: VIDEO_SCHEDULER_INTERNAL_ERROR

The VIDEO_SCHEDULER_INTERNAL_ERROR bug check has a value of 0x00000119. This indicates that the video scheduler has detected a fatal violation.

Parameters

The following parameters are displayed on the blue screen. Parameter 1 is the only parameter of interest and identifies the exact violation.

Parameter 1	Cause of Error
0x1	The driver has reported an invalid fence ID.
0x2	The driver failed upon the submission of a command.
0x3	The driver failed upon patching the command buffer.
0x4	The driver reported an invalid flip capability.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x11A: EM_INITIALIZATION_FAILURE

The EM_INITIALIZATION_FAILURE bug check has a value of 0x0000011A.

This bug check appears very infrequently.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x11B: DRIVER_RETURNED_HOLDING_CANCEL_LOCK

[This is preliminary documentation and subject to change.]

The DRIVER_RETURNED_HOLDING_CANCEL_LOCK bug check has a value of 0x0000011B. This bug check indicates that a driver has returned from a *cancel* routine that holds the global cancel lock. This causes all later cancellation calls to fail, and results in either a deadlock or another bug check.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	The address of the IRP that was canceled (might not be valid).
2	The address of the *cancel* routine.

Comments

The cancel spin lock should have been released by the *cancel* routine.

The driver calls the IoCancelIrpIoCancelIrp function to cancel an individual I/O request packet (IRP). This function acquires the cancel spin lock, sets the cancel flag in the IRP, and then calls the *cancel* routine specified by the appropriate field in the IRP, if a routine was specified. The *cancel* routine is expected to release the cancel spin lock. If there is no *cancel* routine, the cancel spin lock is released.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x11C: ATTEMPTED_WRITE_TO_CM_PROTECTED_STORAGE

[This is preliminary documentation and subject to change.]

The ATTEMPTED_WRITE_TO_CM_PROTECTED_STORAGE bug check has a value of 0x0000011C. This bug check indicates that an attempt was made to write to the read-only protected storage of the configuration manager.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address for the attempted write
2	PTE contents
3	Reserved
4	Reserved

Comments

When it is possible, the name of the driver that is attempting the write operation is printed as a Unicode string on the bug check screen and then saved in KiBugCheckDriver.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x11D: EVENT_TRACING_FATAL_ERROR

[This is preliminary documentation and subject to change.]

The EVENT_TRACING_FATAL_ERROR bug check has a value of 0x0000011D. This bug check indicates that the Event Tracing subsystem has encountered an unexpected fatal error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x121: DRIVER_VIOLATION

[This is preliminary documentation and subject to change.]

The DRIVER_VIOLATION bug check has a value of 0x00000121. This bug check indicates that a driver has caused a violation.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Describes the type of violation
2	Reserved
3	Reserved

Comments

Use a kernel debugger and view the call stack to determine the name of the driver that caused the violation.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x122: WHEA_INTERNAL_ERROR

[This is preliminary documentation and subject to change.]

The WHEA_INTERNAL_ERROR bug check has a value of 0x00000122. This bug check indicates that an internal error in the Windows Hardware Error Architecture (WHEA) has occurred.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x124: WHEA_UNCORRECTABLE_ERROR

[This is preliminary documentation and subject to change.]

The WHEA_UNCORRECTABLE_ERROR bug check has a value of 0x00000124. This bug check indicates that a fatal hardware error has occurred. This bug check uses the error data that is provided by the Windows Hardware Error Architecture (WHEA).

Parameters

Parameter 1	Parameter 2	Parameter 3	Parameter 4	Cause of error

0x0	Address of WHEA_ERROR_RECORD structure.	High 32 bits of MCi_STATUS MSR for the MCA bank that had the error.	Low 32 bits of MCi_STATUS MSR for the MCA bank that had the error.	A machine check exception occurred. These parameter descriptions apply if the processor is based on the x64 architecture, or the x86 architecture that has the MCA feature available (for example, Intel Pentium Pro, Pentium IV, or Xeon).
0x1	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A corrected machine check exception occurred.
0x2	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A corrected platform error occurred.
0x3	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A nonmaskable Interrupt (NMI) error occurred.
0x4	Address of WHEA_ERROR_RECORD structure.	Reserved	Reserved.	An uncorrectable PCI Express error occurred.
0x5	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A generic hardware error occurred.
0x6	Address of WHEA_ERROR_RECORD structure	Reserved.	Reserved.	An IA64 INIT error occurred.
0x7	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A BOOT error occurred.
0x8	Address of WHEA_ERROR_RECORD structure	Reserved.	Reserved.	A Scalable Coherent Interface (SCI) generic error occurred.
0x9	Address of WHEA_ERROR_RECORD structure.	Length, in bytes, of the SAL log.	Address of the SAL log.	An uncorrectable IA-64 machine check abort error occurred.
0xA	Address of WHEA_ERROR_RECORD structure	Reserved.	Reserved.	A corrected IA-64 machine check error occurred.
0xB	Address of WHEA_ERROR_RECORD structure.	Reserved.	Reserved.	A corrected IA-64 platform error occurred.

Comments

Parameter 1 identifies the type of error source that reported the error. Parameter 2 holds the address of the WHEA_ERROR_RECORD structure that describes the error condition.

For information about WHEA, see Windows Hardware Error Architecture Design Guide within the WDK documentation .

Note This bug check is not supported in Windows versions prior to Windows Vista. Instead, machine check exceptions are reported through bug check 0x9C.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x127: PAGE_NOT_ZERO

[This is preliminary documentation and subject to change.]

The PAGE_NOT_ZERO bug check has a value of 0x00000127. This bug check indicates that a page that should have been filled with zeros was not. This bug check might occur because of a hardware error or because a privileged component of the operating system modified a page after freeing it.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address that maps the corrupted page
2	Physical page number
3	Zero (Reserved)
4	Zero (Reserved)

Debugging Tools for Windows

December 09, 2009

Debugging Tools for Windows

Bug Check 0x12B: FAULTY_HARDWARE_CORRUPTED_PAGE

[This is preliminary documentation and subject to change.]

The FAULTY_HARDWARE_CORRUPTED_PAGE bug check has a value of 0x00000128. This bug check indicates that a single-bit error was found in this page. This is a hardware memory error.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Virtual address maps to the corrupted page
2	Physical page number
3	Zero (Reserved)
4	Zero (Reserved)

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x12C: EXFAT_FILE_SYSTEM

[This is preliminary documentation and subject to change.]

The EXFAT_FILE_SYSTEM bug check has a value of 0x0000012C. This bug check indicates that a problem occurred in the Extended File Allocation Table (exFAT) file system.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	Specifies source file and line number information. The high 16 bits (the first four hexadecimal digits after the "0x") determine the source file by its identifier number. The low 16 bits determine the source line in the file where the bug check occurred.
2	If **FppExceptionFilter** is on the stack, this parameter specifies the address of the exception record.
3	If **FppExceptionFilter** is on the stack, this parameter specifies the address of the context record.
4	Reserved.

Cause

This bug check is caused by the file system as a last resort when its internal accounting is in an unsupportable state and to continue poses a large risk of data loss. The file system never causes this bug check when the on disk structures are corrupted, the disk sectors go bad, or a memory allocation fails. Bad sectors could lead to a bug check, for example, when a page fault occurs in kernel code or data and the memory manager cannot read the pages. However, for this bug check, the file system is not the cause.

Resolving the Problem

To debug this problem: Use the **.cxr (Display Context Record)** command together with Parameter 3, and then use **kb (Display Stack Backtrace)**.

Send feedback on this topic
Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1000007E: SYSTEM_THREAD_EXCEPTION_NOT_HANDLED_M

The SYSTEM_THREAD_EXCEPTION_NOT_HANDLED_M bug check has a value of 0x1000007E. This indicates that a system thread generated an exception which the error handler did not catch.

Bug check 0x1000007E has the same meaning and parameters as bug check 0x7E (SYSTEM_THREAD_EXCEPTION_NOT_HANDLED).

Send feedback on this topic
Debugging Tools for Windows

December 09, 2009

Debugging Tools for Windows

Bug Check 0x1000007F: UNEXPECTED_KERNEL_MODE_TRAP_M

The UNEXPECTED_KERNEL_MODE_TRAP_M bug check has a value of 0x1000007F. This indicates that a trap was generated by the Intel CPU and the kernel failed to catch this trap.

Bug check 0x1000007F has the same meaning and parameters as bug check 0x7F (UNEXPECTED_KERNEL_MODE_TRAP).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x1000008E: KERNEL_MODE_EXCEPTION_NOT_HANDLED_M

The KERNEL_MODE_EXCEPTION_NOT_HANDLED_M bug check has a value of 0x1000008E. This indicates that a kernel-mode program generated an exception which the error handler did not catch.

Bug check 0x1000008E has the same meaning and parameters as bug check 0x8E (KERNEL_MODE_EXCEPTION_NOT_HANDLED).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0x100000EA: THREAD_STUCK_IN_DEVICE_DRIVER_M

The THREAD_STUCK_IN_DEVICE_DRIVER_M bug check has a value of 0x100000EA. This indicates that a thread in a device driver is endlessly spinning.

Bug check 0x100000EA has the same meaning and parameters as bug check 0xEA (THREAD_STUCK_IN_DEVICE_DRIVER).

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC0000218: STATUS_CANNOT_LOAD_REGISTRY_FILE

The STATUS_CANNOT_LOAD_REGISTRY_FILE bug check has a value of 0xC0000218. This indicates that a registry file could not be loaded.

Parameters

This bug check will display a descriptive text message. The name of the damaged file is displayed as part of the message.

Cause

This error occurs if a necessary registry hive file cannot be loaded. Usually this means the file is corrupt or is missing.

In rare instances, this error can be caused by a driver that has corrupted the registry image in memory, or by a memory error in this region.

Resolving the Problem

Try running the Emergency Recovery Disk (ERD) and allow the system to repair any errors that it detects. If the problem is a missing or corrupt registry file, this will usually fix the problem.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC000021A: STATUS_SYSTEM_PROCESS_TERMINATED

The STATUS_SYSTEM_PROCESS_TERMINATED bug check has a value of 0xC000021A. This means that an error has occurred in a crucial user-mode subsystem.

Parameters

The following parameters are displayed on the blue screen.

Parameter	Description
1	A string that identifies the problem
2	The error code
3	Reserved
4	Reserved

Cause

This error occurs when a user-mode subsystem, such as WinLogon or the Client Server Run-Time Subsystem (CSRSS), has been fatally compromised and security can no longer be guaranteed. In response, the operating system switches to kernel mode. Microsoft Windows cannot run without WinLogon or CSRSS. Therefore, this is one of the few cases where the failure of a user-mode service can shut down the system.

Mismatched system files can also cause this error. This can occur if you have restored your hard disk from a backup. Some backup programs might skip restoring system files that they determine are in use.

Resolving the Problem

Running the kernel debugger is not useful in this situation because the actual error occurred in a user-mode process.

Resolving an error in a user-mode device driver, system service, or third-party application: Because bug check 0xC000021A occurs in a user-mode process, the most common culprits are third-party applications. If the error occurred after the installation of a new or updated device driver, system service, or third-party application, the new software should be removed or disabled. Contact the manufacturer of the software about a possible update.

If the error occurs during system startup, restart your computer, and press F8 at the character-based menu that displays the operating system choices. At the resulting Windows **Advanced Options** menu, choose the **Last Known Good Configuration** option. This option is most effective when only one driver or service is added at a time. If this does not resolve the error, try manually removing the offending software. If the system partition is formatted with file allocation table (FAT), use an MS-DOS startup disk to gain access to the computer's hard disk. If the system partition is formatted with NTFS file system, you might be able to use Safe Mode to rename or delete the faulty software. If the faulty software is used as part of the system startup process in Safe Mode, you need to start the computer using the Recovery Console in order to access the file. If a newly installed piece if hardware is suspected, remove it to see if this resolves the issue.

Try running the Emergency Recovery Disk (ERD) and allow the system to repair any errors that it detects.

Resolving a mismatched system file problem: If you have recently restored your hard disk from a backup, check if there is an updated version of the Backup/Restore program available from the manufacturer. Make sure the latest Windows Service Pack is installed.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xC0000221: STATUS_IMAGE_CHECKSUM_MISMATCH

The STATUS_IMAGE_CHECKSUM_MISMATCH bug check has a value of 0xC0000221. This indicates that a driver or a system DLL has been corrupted.

Parameters

This bug check will display a descriptive text message. The name of the damaged file is displayed as part of the message.

Cause

This bug check results from a serious error in a driver or other system file. The file header checksum does not match the expected checksum.

This can also be caused by faulty hardware in the I/O path to the file (a disk error, faulty RAM, or a corrupted page file).

Resolving the Problem

To remedy this error, run the Emergency Recovery Disk (ERD) and allow the system to repair or replace the missing or damaged driver file on the system partition.

You can also run an in-place upgrade over the existing copy of Windows. This preserves all registry settings and configuration information, but replaces all system files. If any Service Packs and/or hotfixes had previously been applied, you need to reinstall them afterward in the appropriate order (latest Service Pack, then any post-Service Pack hotfixes in the order in which they were originally installed, if applicable).

If a specific file was identified in the bug check message as being corrupted, you can try replacing that individual file manually. If the system partition is formatted with FAT, you can start from an MS-DOS startup disk and copy the file from the original source onto the hard disk. If you have a dual-boot machine, you can boot to your other operating system and replace the file.

If you want to replace the file on a single-boot system with an NTFS partition, you need to restart the system, press F8 at the operating system **Loader** menu, and choose **Safe Mode with Command Prompt**. From there, copy a fresh version of the file from the original source onto the hard disk. If the file is used as part of the system startup process in Safe Mode, you need to start the computer using the Recovery Console in order to access the file. If these methods fail, try reinstalling Windows and then restoring the system from a backup.

Note If the original file from the product CD has a filename extension ending in an _ (underscore), the file needs to be uncompressed before it can be used. The Recovery Console's **Copy** command automatically detects compressed files and expands them as they are copied to the target location. If you are using Safe Mode to access a drive, use the **Expand** command to uncompress and copy the file to the target folder. You can use the **Expand** command in the command line environment of Safe Mode.

Resolving a disk error problem: Disk errors can be a source of file corruption. Run **Chkdsk /f /r** to detect and resolve any file system structural corruption. You must restart the system before the disk scan begins on a system partition.

Resolving a RAM problem: If the error occurred immediately after RAM was added to the system, the paging file might be corrupted or the new RAM itself might be either faulty or incompatible.

To determine if newly added RAM is causing a bug check

1. Return the system to the original RAM configuration.
2. Use the Recovery Console to access the partition containing the paging file and delete the file *pagefile.sys*.
3. While still in the Recovery Console, run **Chkdsk /r** on the partition that contained the paging file.
4. Restart the system.
5. Set the paging file to an optimal level for the amount of RAM added.
6. Shutdown the system and add your RAM.

   The new RAM must meet the system manufacturer's specifications for speed, parity, and type (that is, fast page-mode (FPM) versus extended data out (EDO) versus synchronous dynamic random access memory (SDRAM)). Try to match the new RAM to the existing installed RAM as closely as possible. RAM can come in many different capacities, and more importantly, in different formats (single inline memory modules — SIMM — or dual inline memory modules — DIMM). The electrical contacts can be either gold or tin and it is not wise to mix these contact types.

If you experience the same error message after reinstalling the new RAM, run hardware diagnostics supplied by the system manufacturer, especially the memory scanner. For details on these procedures, see the owner's manual for your computer.

When you can log on to the system again, check the System Log in Event Viewer for additional error messages that might help pinpoint the device or driver that is causing the error.

Disabling memory caching of the BIOS might also resolve this error.

Debugging Tools for Windows
December 09, 2009

Debugging Tools for Windows

Bug Check 0xDEADDEAD: MANUALLY_INITIATED_CRASH1

The MANUALLY_INITIATED_CRASH1 bug check has a value of 0xDEADDEAD. This indicates that the user deliberately initiated a crash dump from either the kernel debugger or the keyboard.

Parameters

None

Comments

For details on manually-initiated crash dumps, see Forcing a System Crash.

Debugging Tools for Windows
December 09, 2009