



LABS

This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 1999-2002 Microsoft Corporation. All rights reserved.

Microsoft, Microsoft Press, MSN, MS-DOS, Windows, Windows NT, CodeView, MSDN, Visual C++, Visual Studio, Win32, and Win64 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Intel is a registered trademark of Intel Corporation. Itanium is a registered trademark of Intel Corporation.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Debugging Workshop Labs

TABLE OF CONTENTS

DEBUGGING WORKSHOP LABS	1
Module 2 Labs: Debugger Installation and Setup	5
Lab Objectives	5
Lab 2 – Debugger Installation and Setup	5
Exercise 1 – Installing Debugging Tools for Windows	5
Estimated Time to Complete this Lab: 10 minutes	5
Exercise 2: Installing Debugging Tools for Windows	6
Module 3 Labs: Introduction to Debugger Operation	11
Lab Objectives	11
Lab 3 – Introduction to Debugger Operation	11
Exercise 1 – Opening a Kernel-Mode Dump File	11
Exercise 2 – Collecting Information from Dump Files	11
Exercise 3 – Open memory1.dmp	11
Estimated Time to Complete this Lab: 45 minutes	11
Exercise 1: Opening a Kernel-Mode Dump File	12
Exercise 2: Collecting Information from Dump Files	16
Exercise 3: Open memory1.dmp	17
Module 4 Labs: Key Concepts and Data Structures	18
Lab Objectives	18
Lab 4 – Key Concepts and Data Structures	18
Exercise 1 – Viewing Processor-Specific Support Files	18
Exercise 2 – Viewing Installed Device Drivers	18
Exercise 3 – Viewing the Process Tree using TList	18
Exercise 4 – Looking at Pool Allocations by Tag	18
Estimated Time to Complete this Lab: 30 Minutes	18
Exercise 1: Viewing Processor-Specific Support Files	19
Exercise 2: Viewing Installed Device Drivers	21
Exercise 3: Viewing the Process Tree Using TList	23
Exercise 4: Looking at Pool Allocations by Tag	26
Module 5 Labs: Crash Dump Analysis I	28
Lab Objectives	28
Lab 5 – Crash Dump Analysis I	28

Exercise 1 – Analyzing a Kernel-Mode Dump File (Part 1)	28
Estimated Time to Complete this Lab: 75 minutes	28
Exercise 1: Analyzing a Kernel-Mode Dump File (Part 1)	29
Module 6 Labs: Kernel Debugging I	36
Lab Objectives	36
Lab 6 – Kernel Debugging I	36
Exercise 1 – Setting Up a Kernel Debugging Session	36
Exercise 2 – Debugging a “System Hang” Caused by a High-Priority Thread	36
Exercise 3 – Debugging a “System Hang” Caused by a Driver Looping at High IRQL	36
Estimated Time to Complete this Lab: minutes	36
Exercise 1: Setting up a Kernel Debugging Session	37
Part 1: Setting up the target computer in debug mode	37
Part 2: Starting a Debug Session With WinDbg on the Host System	38
Exercise 2: Debugging a “System Hang” Caused by a High-Priority Thread	42
Part 1: Creating the Problem Scenario	42
Part 2: Analyzing the Problem with WinDbg	44
Exercise 3: Debugging a “System Hang” at High IRQL	45
Part 1: Installing the Buggy application	45
Part 2: Creating the Problem Scenario	46
Part 3: Analyzing the Problem with WinDbg	46
Module 7 Labs: Understanding Disassembled Code	47
Lab Objectives	47
Lab 7 – Understanding Disassembled Code	47
Exercise 1 – Mapping Assembly to C / C++	47
Exercise 2 – Reading Assembly Language (Part 1)	47
Exercise 3 – Reading Assembly Language (Part 2)	47
Estimated Time to Complete this Lab: 30 Minutes	47
Exercise 1: Mapping Assembly to C / C++	48
Exercise 2: Reading Assembly Language (Part 1)	49
Exercise 3: Reading Assembly Language (Part 2)	50
Module 8 Labs: Call Stacks	51
Lab Objectives	51
Lab 8 – Call Stacks	51
Exercise 1 – Reading a Call Stack	51
Exercise 2 – Identifying Calling Conventions	51
Estimated Time to Complete this Lab: 30 Minutes	51
Exercise 1: Reading a Call Stack	52
Exercise 2: Identifying Calling Conventions	53
Module 9 Labs: Crash Dump Analysis II	54
Lab Objectives	54
Lab 9 – Crash Dump Analysis II	54
Exercise 1 – Using WinDbg to Analyze a Kernel-Mode Dump File (Part 3)	54
Estimated Time to Complete this Lab: 90 minutes	54
Exercise 1: Using WinDbg to Analyze the Kernel-Mode Dump (Part 3)	54
Exercise 1: Using WinDbg to Analyze the Kernel-Mode Dump (Part 3)	55
Continuing the analysis of the <i>memory.dmp</i> file	55
Module 10 Labs: Kernel Debugging II	60
Lab Objectives	60
Lab 10 – Kernel Debugging II	60
Exercise 1 – Using Some Basic Debugger Commands in Kernel Mode	60
Exercise 2 – Live Debugging Using “Buggy.sys”	60
Exercise 3 – Finding a Resource Deadlock	60

Exercise 4 – Using PoolMon to find a Kernel-Mode Memory Leak	60
Exercise 5 – Remote Debugging with WinDbg	60
Estimated Time to Complete this Lab: 90 minutes	60
Exercise 1: Using Some Basic Debugger Commands in Kernel Mode	61
Exercise 2: Live Debugging using “Buggy.Sys”	66
Part 1: Installing the Buggy application	66
Part 2: Using the Buggy application	67
Part 3: Choosing which problem you want to debug	67
Step 4 – Debugging the problem	68
Have some extra time?	68
Exercise 3: Finding a Resource Deadlock	69
Loading the <i>deadlock.dmp</i> file with WinDbg	69
Exercise 4: Using PoolMon to find a Kernel-Mode Memory Leak	70
Exercise 5: Remote Debugging with WinDbg	71
Starting a Remote Debugging Session With WinDbg	71
Connecting to a Remote Debugging Session With WinDbg	72
Module 11 Labs: User-Mode Debugging	74
Lab Objectives	74
Lab 11 – User-Mode Debugging	74
Exercise 1 – Attaching to Running Process	74
Exercise 2 – Using Some Basic Debugger Commands in User Mode	74
Exercise 3 – Analyzing a User-Mode Dump File	74
Exercise 4 – Using UMDH to Find a Memory Leak in an Application	74
Estimated Time to Complete this Lab: 60 minutes	74
Exercise 1: Attaching to Running Process	75
Attach to a Running Process Using WinDbg	75
Attach to a Running Process Using CDB	76
Exercise 2: Using Some Basic Debugger Commands in User Mode	78
Debugging a Process Using WinDbg	78
Exercise 3: Analyzing a User-Mode Dump File	87
Loading a user.dmp file using WinDbg	87
Exercise 4 – Using UMDH to Find a Memory Leak in an Application	89
Exercise 4 – Using UMDH to Find a Memory Leak in an Application	89

Module 2 Labs: Debugger Installation and Setup

Lab Objectives



Lab 2 – Debugger Installation and Setup

Exercise 1 – Installing Debugging Tools for Windows

Estimated Time to Complete this Lab: 10 minutes

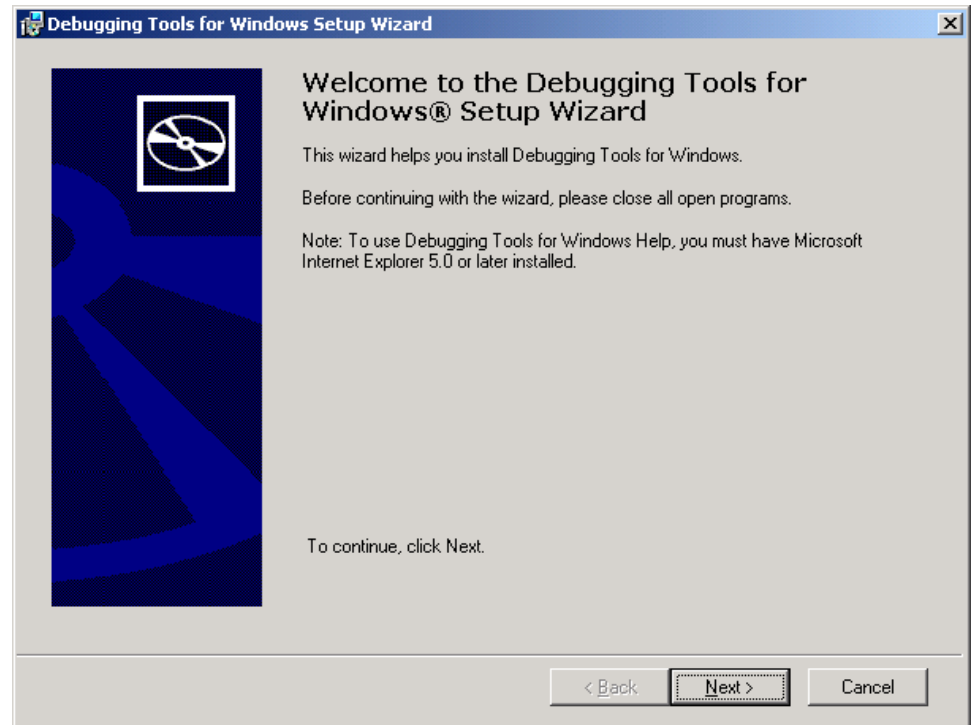
Exercise 2: Installing Debugging Tools for Windows

In this exercise you will install the *Debugging Tools for Windows* package with its default configuration.

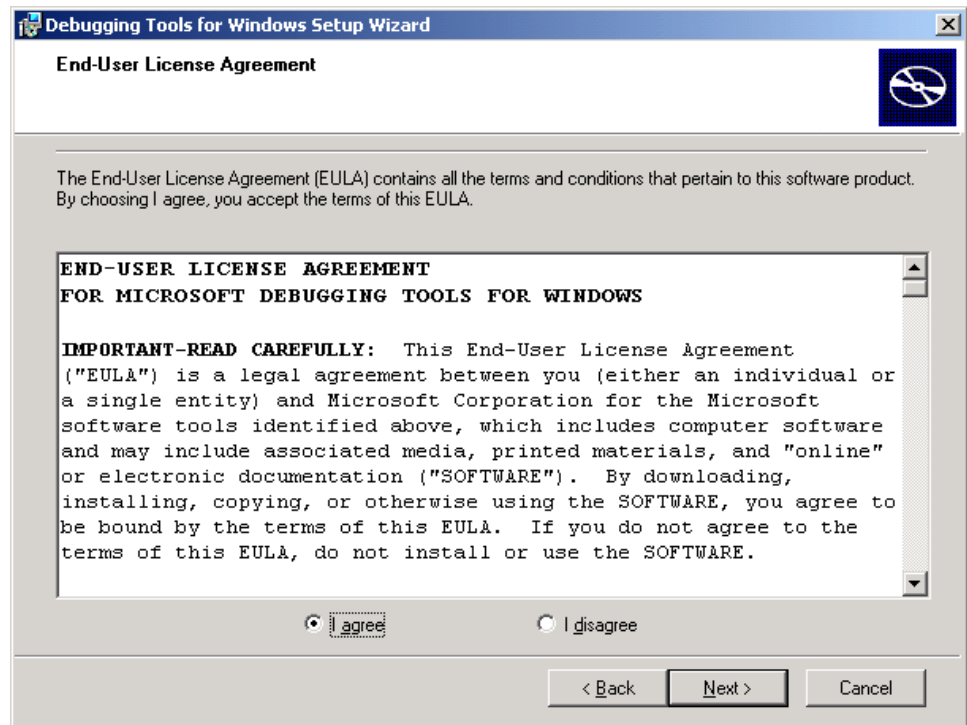
Note: Some of the dialogs may be slightly different from those printed here.

1. From the *d:\Labs\Debuggers* folder execute *dbg_x86_6.0.17.0.exe*.

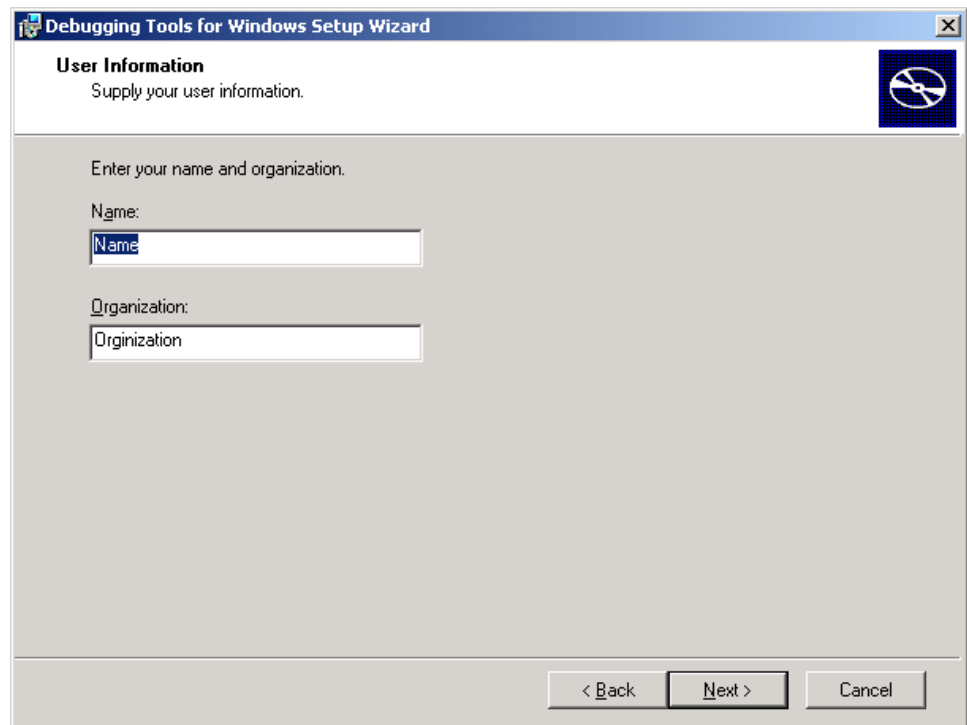
Note: Your instructor may provide this file in a different location or name.



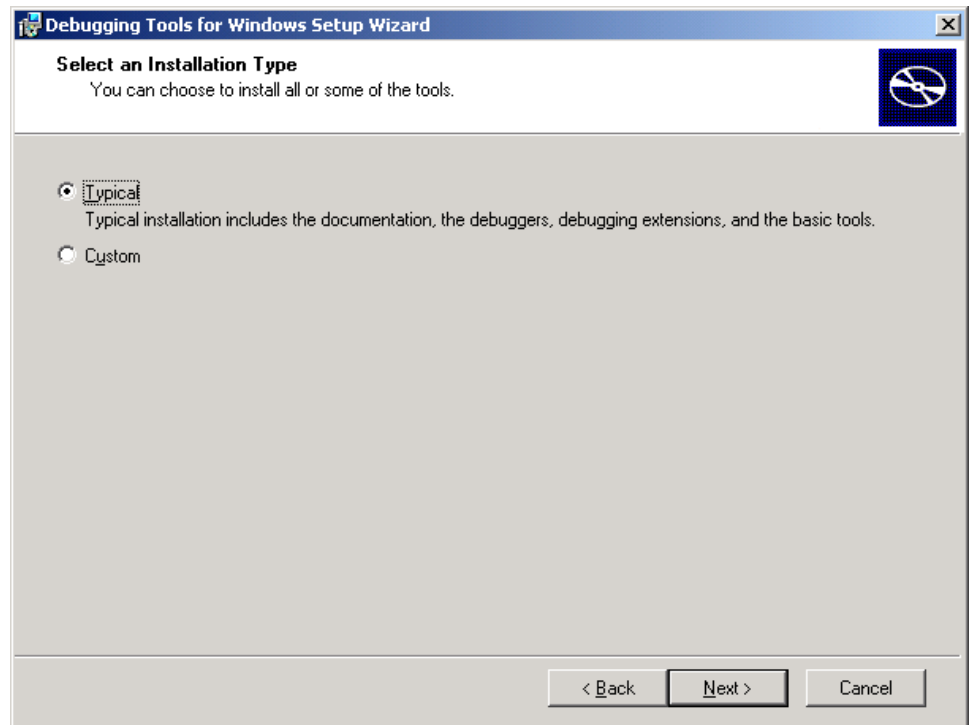
2. Select “*Next*” to continue.



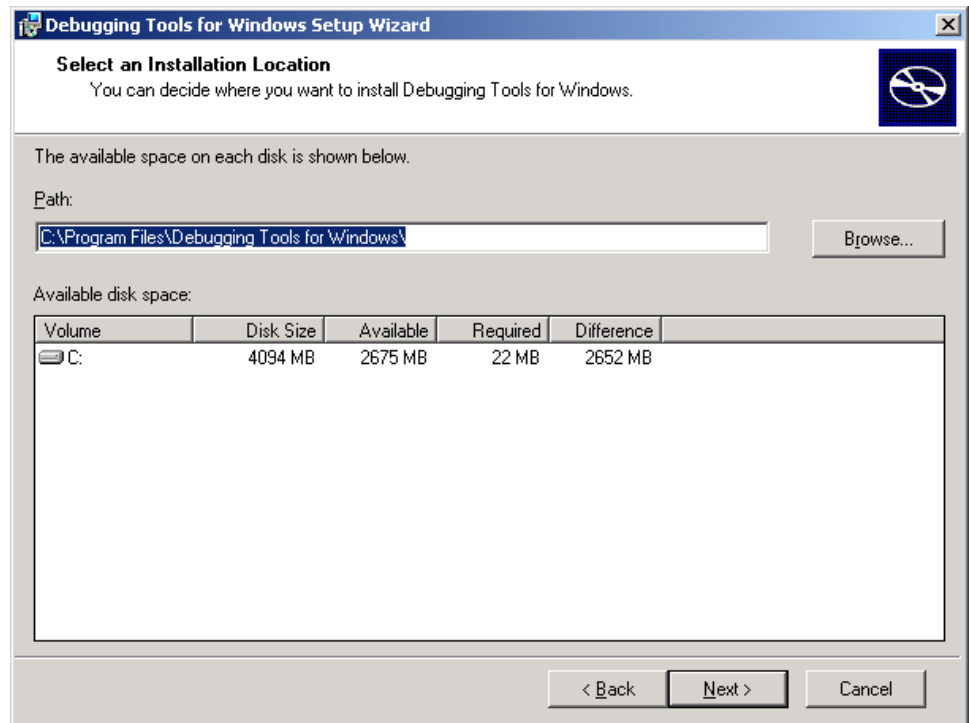
3. Select the “*I Agree*” radio button to accept the EULA. Then select “*Next*” to continue the installation.



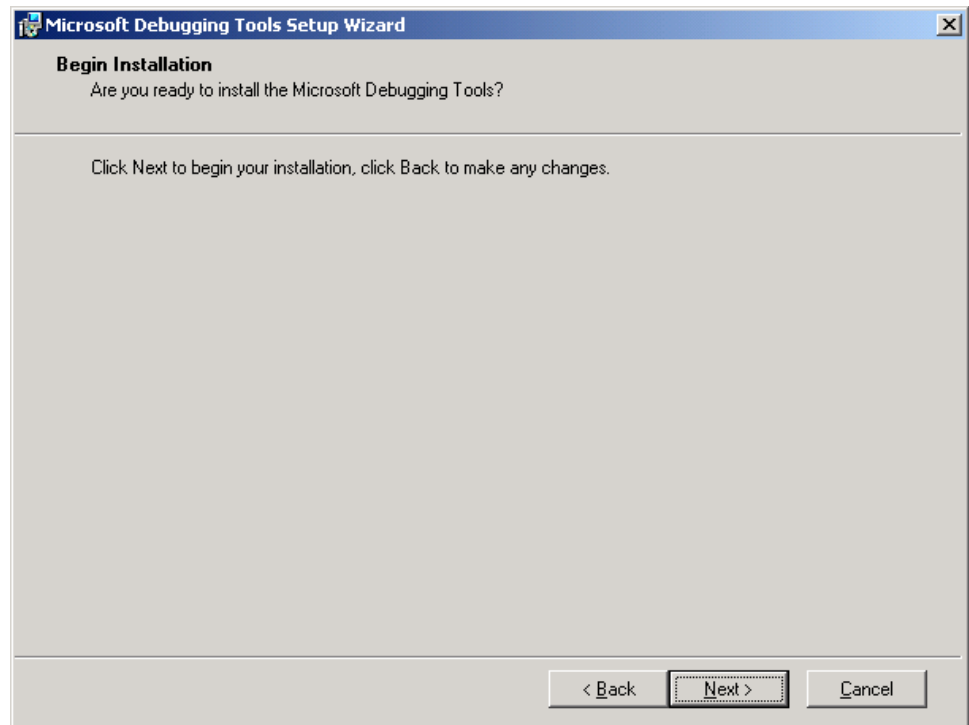
4. Enter a name and an organization and then select “*Next*.”



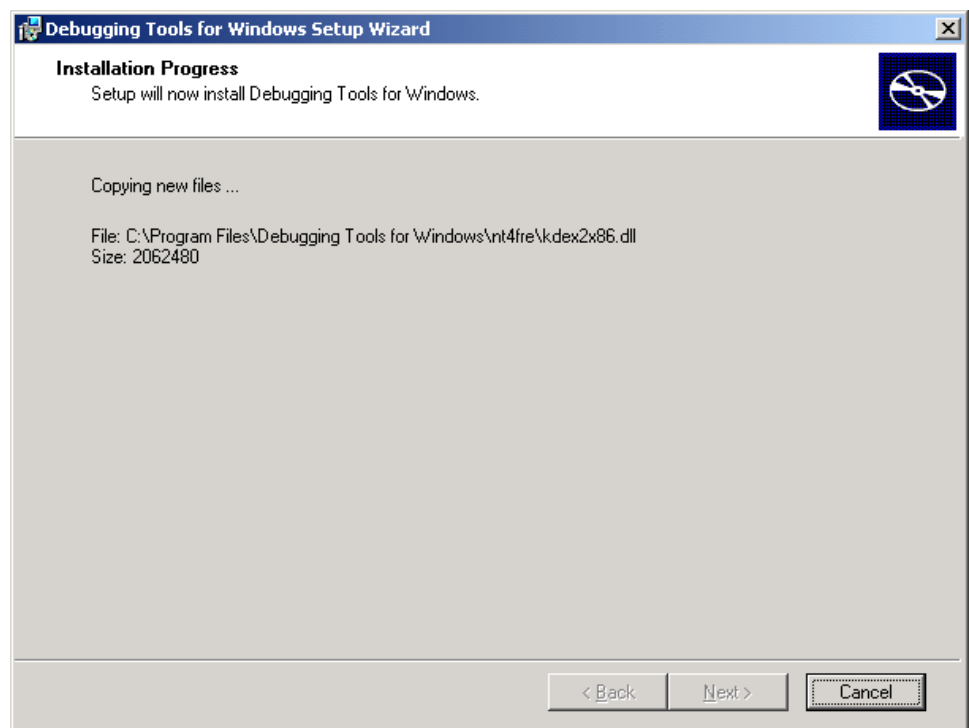
5. Select "Next" to accept the "Typical" installation type.



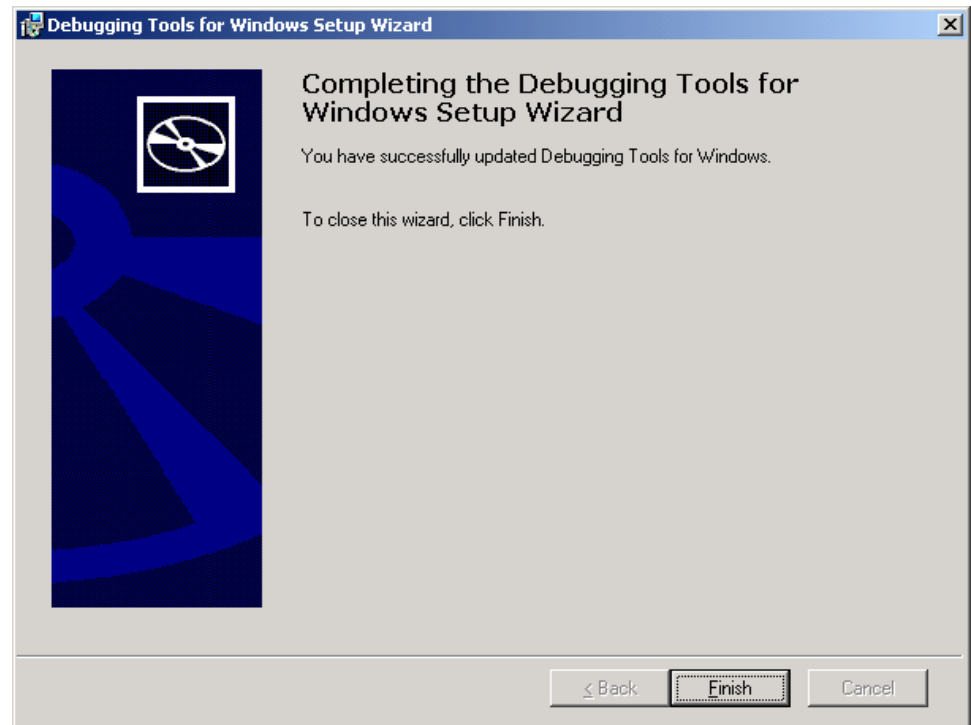
6. Accept the default installation location by selecting "Next."



7. Select "Next."



8. When the installation is complete a dialog box should appear informing you that the installation was successful:



9. Select "*Finish*" to complete the installation of the Debugging Tools for Windows.

Module 3 Labs: Introduction to Debugger Operation

Lab Objectives



Lab 3 – Introduction to Debugger Operation

Exercise 1 – Opening a Kernel-Mode Dump File

Exercise 2 – Collecting Information from Dump Files

Exercise 3 – Open memory1.dmp

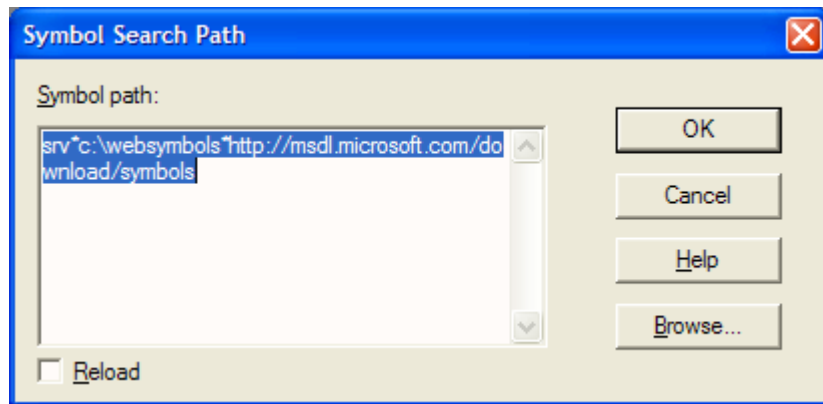
Estimated Time to Complete this Lab: 45 minutes

Exercise 1: Opening a Kernel-Mode Dump File

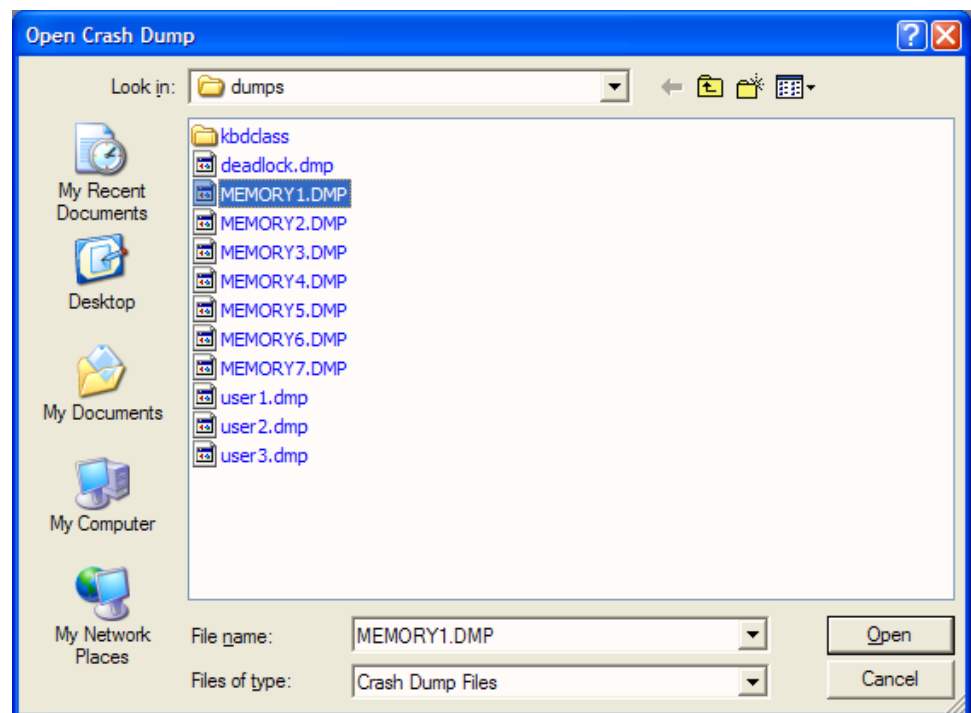
In this exercise, you will use WinDbg to load a kernel mode memory dump file for analysis. The file is located in the `c:\dumps` directory. (Your instructor may provide an alternate file path.)

1. Select **Start | Program | Microsoft Debugging Tools | WinDbg**.
2. Select **File | Symbol File Path**. The “Symbol Search Path” dialog should appear.
3. As shown below, set the symbol path to **srv*c:\websymbols*http://msdl.microsoft.com/download/symbols**

Note: Your instructor may provide an alternate symbol search path.

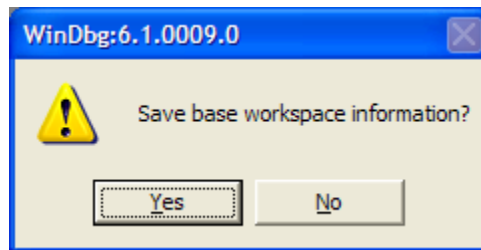


4. Select **File | Open Crash Dump**.



5. Select **MEMORY1.DMP** and then select **Open**.

6. A “save workspace information?” dialog box may appear. Click “No.” (We will say more about workspaces later.)



7. The debugger should open the dump file, display the bugcheck code and parameters, and report other information about the dump.

 A screenshot of the WinDbg 6.1.0009.0 interface. The title bar reads "Dump F:\seminars\int221\dumps\MEMORY1.DMP - WinDbg:6.1.0009.0". The menu bar includes File, Edit, View, Debug, Window, and Help. The Command window is active, displaying the following text:


```

Microsoft (R) Windows Debugger Version 6.1.0009.0
Copyright (C) Microsoft Corporation. All rights reserved.

Loading Dump File [F:\seminars\int221\dumps\MEMORY1.DMP]
Kernel Summary Dump File: Only kernel address space is available

Symbol search path is: srv*c:\websymbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 2000 Kernel Version 2195 UP Free x86 compatible
Product: Server
Kernel base = 0x80400000 PsLoadedModuleList = 0x8046a4c0
Debug session time: Wed Aug 16 16:05:41 2000
System Uptime: 0 days 0:17:15.291
Loading Kernel Symbols
.....
Loading unloaded module list
No unloaded module list present
Loading User Symbols
*****
*                                     *
*                               Bugcheck Analysis                               *
*                                     *
*****

Use !analyze -v to get detailed debugging information.

BugCheck 1E, {c0000005, 804676c8, 1, 0}

Probably caused by : ntoskrnl.exe ( nt!ExFreePool+b )

Followup: Pool_corruption
-----
kd>
  
```

 The status bar at the bottom shows "Ln 0, Col 0", "Sys 0:F:\semin", "Proc 000:0", "Thrd 000:0", and tabs for "ASM", "OVR", "CAPS", and "NUM".

8. Check the debugger output carefully for any signs of symbol problems. In this case there are none, so we can proceed with analysis. If any messages such as

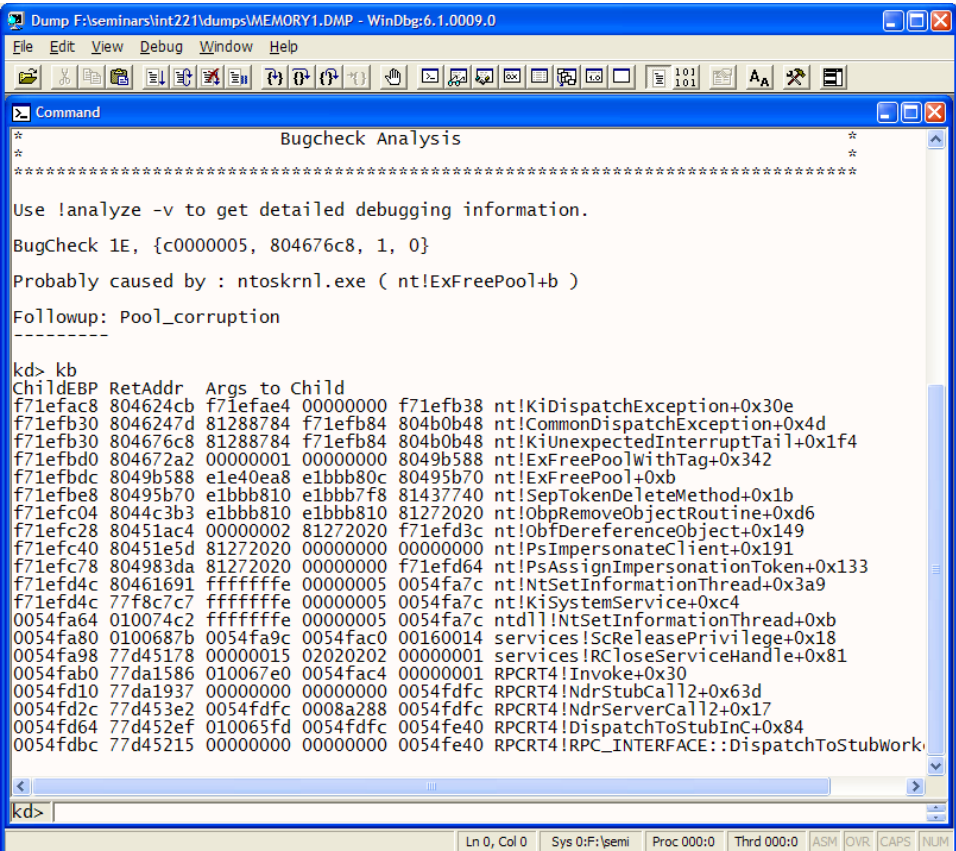
```
*** ERROR: Symbol file could not be found.  Defaulted to
export symbols for ntoskrnl.exe -
```

or

```
***** Kernel symbols are WRONG. Please fix symbols to do
analysis.
```

appear, you must correct the problem before proceeding. Refer to the workbook for information and examples of the **!sym noisy** and **.reload** commands.

9. Next let's take a look at the stack trace. Type "**kb** <enter>".



```

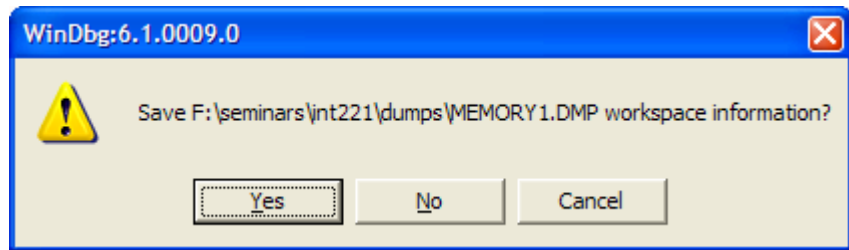
Dump F:\seminars\int221\dumps\MEMORY1.DMP - WinDbg:6.1.0009.0
File Edit View Debug Window Help
[Icons]
Command
Bugcheck Analysis
*****
Use !analyze -v to get detailed debugging information.
BugCheck 1E, {c0000005, 804676c8, 1, 0}
Probably caused by : ntoskrnl.exe ( nt!ExFreePool+b )
Followup: Pool_corruption
-----
kd> kb
ChildEBP RetAddr  Args to Child
f71efac8 804624cb f71efae4 00000000 f71efb38 nt!KiDispatchException+0x30e
f71efb30 8046247d 81288784 f71efb84 804b0b48 nt!CommonDispatchException+0x4d
f71efb30 804676c8 81288784 f71efb84 804b0b48 nt!KiUnexpectedInterruptTail+0x1f4
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 80495b70 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77da5178 00000015 02020202 00000001 services!RcCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall2+0x63d
0054fd2c 77da53e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall2+0x17
0054fd64 77da52ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77da5215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWork
kd>
Ln 0, Col 0 Sys 0:F:\semi Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

The output should be identical (or at least similar) to that shown here. The stack backtrace shows the sequence of procedure calls leading up to the crash. The most recent calls are at the top of the stack trace. For example, The most recent routine called here was **KiDispatchException**; **KiDispatchException** was called by **CommonDispatchException**, which was called by **KiUnexpectedInterruptTail**, etc.

10. To end the debugging session, use either Shift-F5 or Debug | Stop Debugging. Alternately, you can close the WinDbg process completely, with Alt-F4, File | Exit.

11. A “Save workspace information?” dialog box will appear. Click the “No” button.



Exercise 2: Collecting Information from Dump Files

In this exercise, you will use WinDbg to load a series of kernel mode memory dump files and gather information from them. The files are located in the `c:\dumps` directory.

1. Select **Start | Program | Microsoft Debugging Tools | WinDbg**.
Alternately, if WinDbg is already running from the previous exercise, you can use Shift-F5, or the Debug | Stop Debugging command, to close that debugging session while leaving WinDbg running.
2. Open the Symbol File Path dialog. Set the symbol file path to that used in the previous exercise.
3. Open the Open Crash Dump dialog. Open **memory2.dmp** from the same path used in the previous exercise.
4. Inspect the debugger output for symbol file problems. If any exist, resolve them before proceeding.
5. Inspect the stack trace for the names of the routines currently executing.
6. Repeat the above sequence for **memory3.dmp** through **memory7.dmp**, and for **deadlock.dmp**.

Exercise 3: Open memory1.dmp

We will be using the debugger to inspect **memory1.dmp** while reviewing the information in the next module. Therefore, start the debugger once again, open **memory1.dmp** , and have it available for discussion during the next lecture period.

Module 4 Labs: Key Concepts and Data Structures

Lab Objectives



Lab 4 – Key Concepts and Data Structures

Exercise 1 – Viewing Processor-Specific Support Files

Exercise 2 – Viewing Installed Device Drivers

Exercise 3 – Viewing the Process Tree using TList

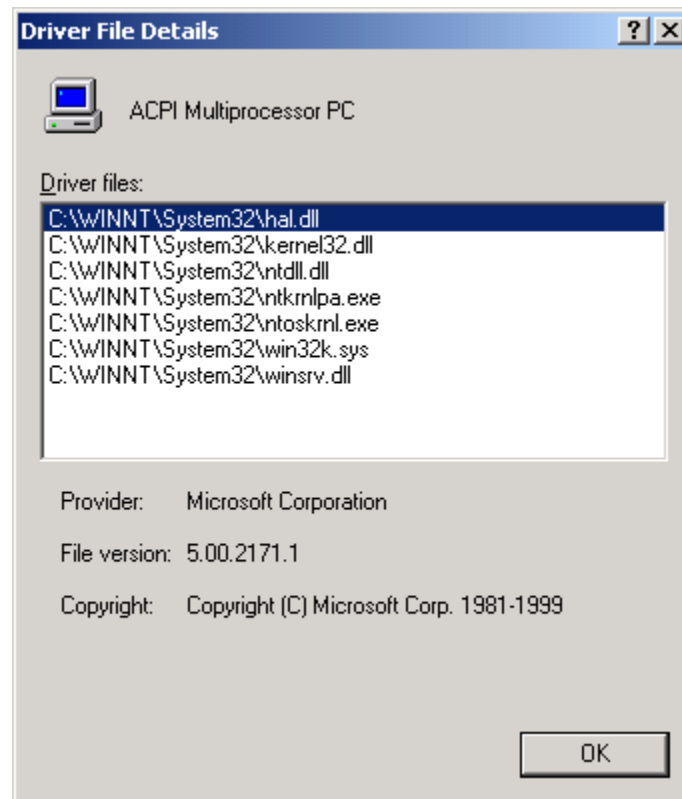
Exercise 4 – Looking at Pool Allocations by Tag

Estimated Time to Complete this Lab: 30 Minutes

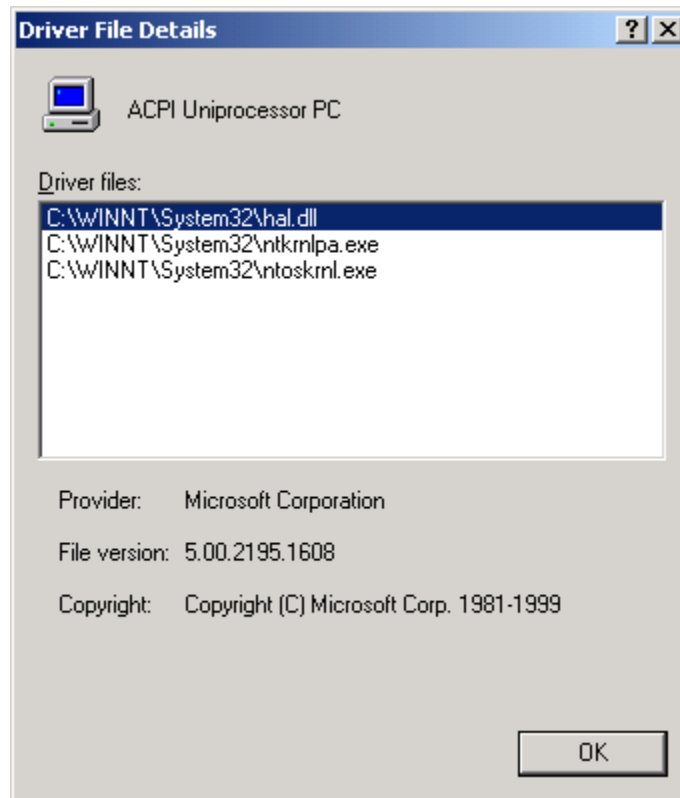
Exercise 1: Viewing Processor-Specific Support Files

1. Open the System properties (either by selecting System from Control Panel or by right-clicking on the My Computer icon on your desktop and selecting Properties).
2. Click on the Hardware tab.
3. Click Device Manager.
4. Expand the Computer object.
5. Double-click on the child node underneath Computer.
6. Click on the Driver tab.
7. Click Driver Details.

You should see this dialog box for a multiprocessor system:



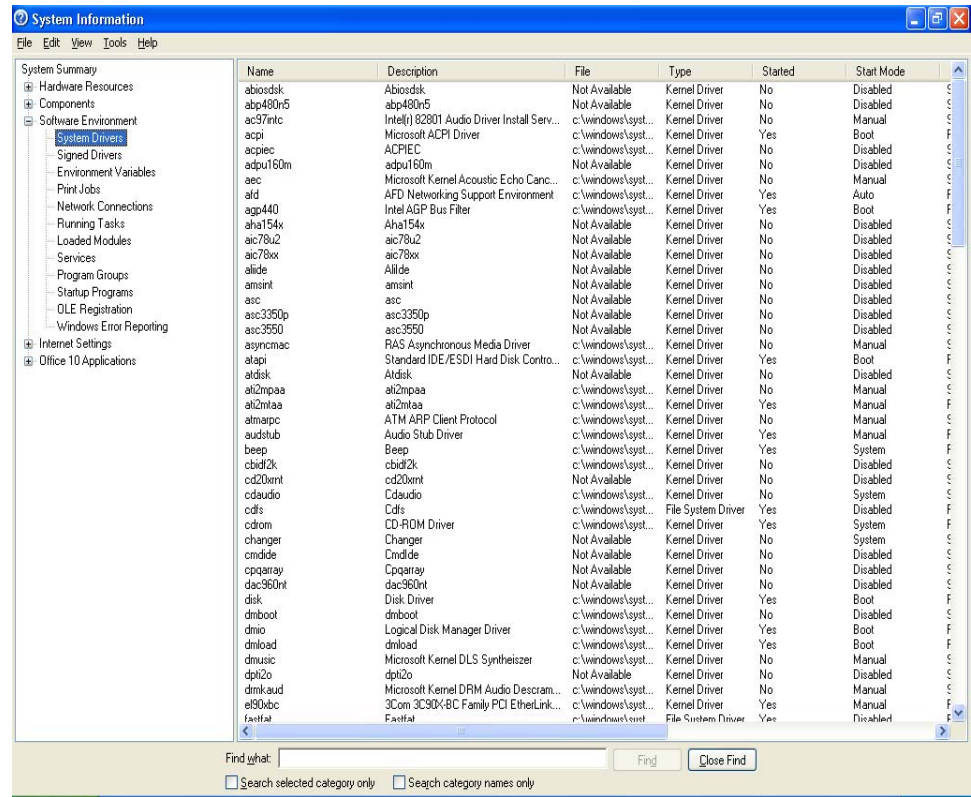
You should see this dialog box for a uniprocessor system:



Exercise 2: Viewing Installed Device Drivers

You can list the installed drivers by running MSInfo32.exe. (From the Start menu, select Run, then type MSInfo32.exe.) From within MSInfo32.exe, expand Software Environment, then select System Drivers.

Here's a sample output of the list of installed drivers:



This window displays the list of device drivers defined in the registry, their type, and their state (Running or Stopped). Device drivers and Win32 service processes are both defined in the same place: HKLM\SYSTEM\CurrentControlSet\Services. However, they are distinguished by a type code—type 1 is a kernel-mode device driver, and type 2 is a file system driver.

Alternatively, you list the currently loaded device drivers with the Drivers utility (*Drivers.exe* in the Windows Resource Kit) or the PStat utility (*PStat.exe* in the Platform SDK). These tools have been included in the *labs/tools* folder on your CD for class.

Here is a partial output from the Drivers utility:

```
C:\>drivers
```

ModuleName	Code	Data	Bss	Paged	Init	LinkDate
ntoskrnl.exe	429184	96896	0	775360	138880	Tue Dec 07 18:41:11 1999
hal.dll	25856	6016	0	16160	10240	Tue Nov 02 20:14:22 1999
BOOTVID.DLL	5664	2464	0	0	320	Wed Nov 03 20:24:33 1999
ACPI.sys	92096	8960	0	43488	4448	Wed Nov 10 20:06:04 1999
WMILIB.SYS	512	0	0	1152	192	Sat Sep 25 14:36:47 1999
pci.sys	12704	1536	0	31264	4608	Wed Oct 27 19:11:08 1999
isapnp.sys	14368	832	0	22944	2048	Sat Oct 02 16:00:35 1999
compbatt.sys	2496	0	0	2880	1216	Fri Oct 22 18:32:49 1999
BATTC.SYS	800	0	0	2976	704	Sun Oct 10 19:45:37 1999
intelide.sys	1760	32	0	0	128	Thu Oct 28 19:20:03 1999

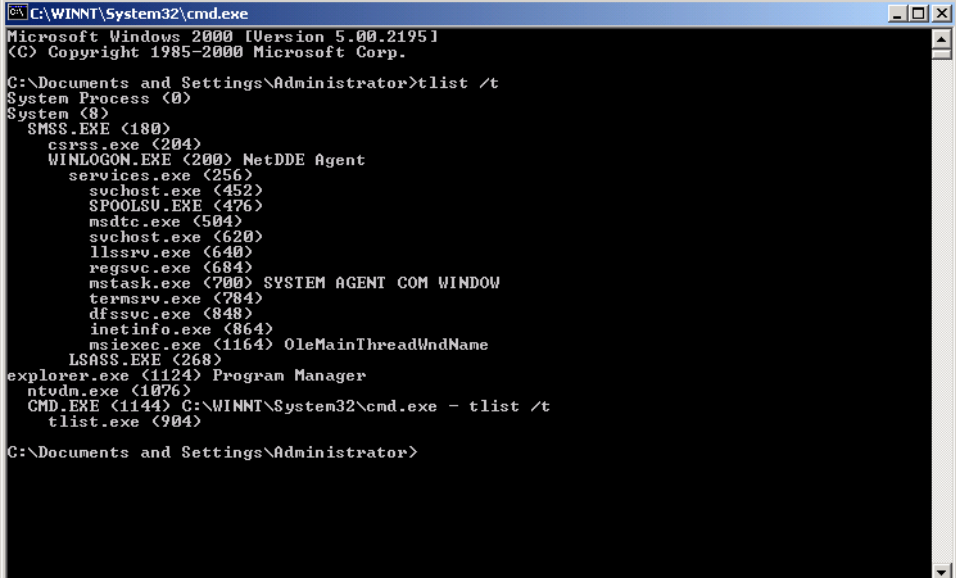
PCIINDEX.SYS	4544	480	0	10944	1632	Wed Oct 27 19:02:19 1999
pcmcia.sys	32800	8864	0	23680	6240	Fri Oct 29 19:20:08 1999
ftdisk.sys	4640	32	0	95072	3392	Mon Nov 22 14:36:23 1999

Total 4363360 580320 0 3251424 432992

Exercise 3: Viewing the Process Tree Using TList

Viewing the Process Tree

One unique attribute about a process that most tools don't display is the parent or creator process ID. You can retrieve this value with the Performance tool (or programmatically) by querying the Creating Process ID. The Windows Support Tools command **tlist /t** uses the information in the attribute to display a *process tree* that shows the relationship of a process to its parent. Here's an example of output from **tlist /t**:



```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>tlist /t
System Process (0)
System (8)
  SMSS.EXE (180)
    csrss.exe (204)
      WINLOGON.EXE (200) NetDDE Agent
        services.exe (256)
          svchost.exe (452)
            SPOOLSV.EXE (476)
              msdtc.exe (504)
                svchost.exe (620)
                  llssrv.exe (640)
                    regsvc.exe (684)
                      mstask.exe (700) SYSTEM AGENT COM WINDOW
                        termsrv.exe (784)
                          dfssvc.exe (848)
                            inetinfo.exe (864)
                              msieexec.exe (1164) OleMainThreadWndName
                                LSASS.EXE (268)
                                  explorer.exe (1124) Program Manager
                                    ntvdm.exe (1076)
                                      CMD.EXE (1144) C:\WINNT\System32\cmd.exe - tlist /t
                                        tlist.exe (904)

C:\Documents and Settings\Administrator>
```

TList indents each process to show its parent/child relationship. Processes whose parents aren't alive are left-justified, because even if a grandparent process exists, there's no way to find that relationship. Windows maintains only the creator process ID, not a link back to the creator of the creator, and so forth.

To demonstrate the fact that Windows doesn't keep track of more than just the parent process ID, follow these steps:

1. Open a Command Prompt window.
2. Type *start cmd* (which runs a second Command Prompt).
3. Bring up Task Manager.
4. Switch to the second Command Prompt.
5. Type *mspaint* (which runs Microsoft Paint).
6. Click the intermediate (second) Command Prompt window.
7. Type *exit*. (Notice that Paint remains.)
8. Switch to Task Manager.
9. Click the Applications tab.
10. Right-click on the Command Prompt task, and select Go To Process.
11. Click on the Cmd.exe process highlighted in gray.
12. Right-click on this process, and select End Process Tree.
13. Click Yes in the Task Manager Warning message box.

The first Command Prompt window will disappear, but you should still see the Paintbrush window because it was the grandchild of the Command Prompt process you terminated; and because the intermediate process (the parent of Paintbrush) was terminated, there was no link between the parent and the grandchild.

```

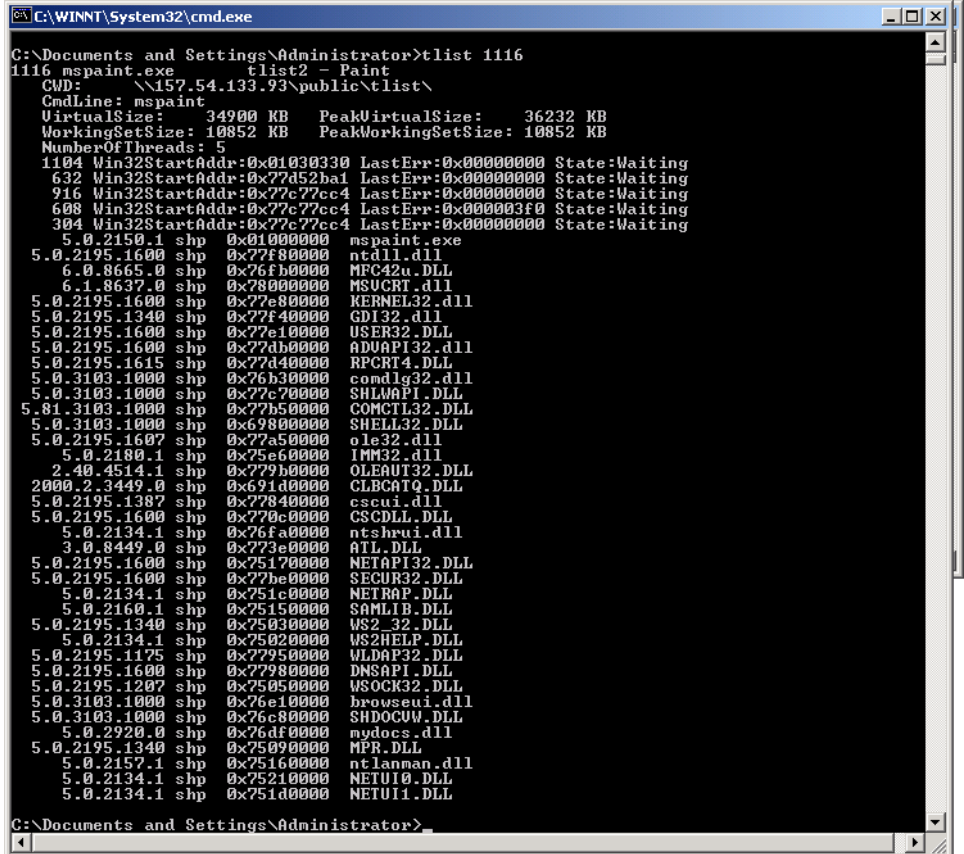
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>tasklist /t
System Process (0)
System (8)
  SMSS.EXE (180)
    csrss.exe (204)
      WINLOGON.EXE (200) NetDDE Agent
        services.exe (256)
          svchost.exe (452)
            SPoolSV.EXE (476)
              msdtc.exe (504)
                svchost.exe (620)
                  llssrv.exe (640)
                    regsvc.exe (684)
                      mstask.exe (700) SYSTEM AGENT COM WINDOW
                        termsrv.exe (784)
                          dfssvc.exe (848)
                            inetinfo.exe (864)
                              msimexec.exe (1164) OleMainThreadWndName
                                LSASS.EXE (268)
                                  explorer.exe (1124) Program Manager
                                    ntvdm.exe (1076)
                                      taskmgr.exe (340) Windows Task Manager
                                        CMD.EXE (1052) C:\WINNT\System32\cmd.exe - tasklist /t
                                          tlist.exe (916)
                                            mspaint.exe (1116) untitled - Paint

C:\Documents and Settings\Administrator>_
  
```

Notice that `tlst.exe` will also give you thread and loaded module information for a process when you enter `tlst <pid>`.

1. Type "`tlst <pid>`" from the command window. Using the Process Id for `mspaint.exe`.



```

C:\WINNT\System32\cmd.exe
C:\Documents and Settings\Administrator>tlst 1116
1116 mspaint.exe      tlst2 - Paint
   CWD:      \\157.54.133.93\public\tlst\
   CmdLine:  mspaint
   VirtualSize:  34900 KB   PeakVirtualSize:  36232 KB
   WorkingSetSize: 10852 KB   PeakWorkingSetSize: 10852 KB
   NumberOfThreads: 5
   1104 Win32StartAddr:0x01030330 LastErr:0x00000000 State:Waiting
   632 Win32StartAddr:0x77d52ba1 LastErr:0x00000000 State:Waiting
   916 Win32StartAddr:0x77c77cc4 LastErr:0x00000000 State:Waiting
   608 Win32StartAddr:0x77c77cc4 LastErr:0x0000003f0 State:Waiting
   304 Win32StartAddr:0x77c77cc4 LastErr:0x00000000 State:Waiting
   5.0.2150.1 shp 0x01000000 mspaint.exe
   5.0.2195.1600 shp 0x77f80000 ntapi.dll
   6.0.8665.0 shp 0x76fb0000 Ntfs42a.DLL
   6.1.8637.0 shp 0x78000000 MSUCRT.dll
   5.0.2195.1600 shp 0x77e80000 KERNEL32.dll
   5.0.2195.1340 shp 0x77f40000 GDI32.dll
   5.0.2195.1600 shp 0x77e10000 USER32.DLL
   5.0.2195.1600 shp 0x77db0000 ADVAPI32.dll
   5.0.2195.1615 shp 0x77d40000 RPCRT4.DLL
   5.0.3103.1000 shp 0x76b30000 comdlg32.dll
   5.0.3103.1000 shp 0x77c70000 SHLWAPI.DLL
   5.81.3103.1000 shp 0x77b50000 COMCTL32.DLL
   5.0.3103.1000 shp 0x69800000 SHELL32.DLL
   5.0.2195.1607 shp 0x77a50000 ole32.dll
   5.0.2180.1 shp 0x75e60000 IMM32.dll
   2.40.4514.1 shp 0x779b0000 OLEAUT32.DLL
   2000.2.3449.0 shp 0x691d0000 CLBCATQ.DLL
   5.0.2195.1387 shp 0x77840000 cscui.dll
   5.0.2195.1600 shp 0x770c0000 GSCDLL.DLL
   5.0.2134.1 shp 0x76fa0000 ntshrui.dll
   3.0.8449.0 shp 0x773e0000 ATL.DLL
   5.0.2195.1600 shp 0x75170000 NETAPI32.DLL
   5.0.2195.1600 shp 0x77be0000 SECUR32.DLL
   5.0.2134.1 shp 0x751c0000 NETRAP.DLL
   5.0.2160.1 shp 0x75150000 SAMLIB.DLL
   5.0.2195.1340 shp 0x75030000 WS2_32.DLL
   5.0.2134.1 shp 0x75020000 WS2HELP.DLL
   5.0.2195.1175 shp 0x77950000 WLDAP32.DLL
   5.0.2195.1600 shp 0x77980000 DNSAPI.DLL
   5.0.2195.1207 shp 0x75050000 WSOCK32.DLL
   5.0.3103.1000 shp 0x76e10000 browseui.dll
   5.0.3103.1000 shp 0x76c80000 SHDOCUW.DLL
   5.0.2920.0 shp 0x76df0000 ngydocs.dll
   5.0.2195.1340 shp 0x75090000 MPR.DLL
   5.0.2157.1 shp 0x75160000 ntlanman.dll
   5.0.2134.1 shp 0x75210000 NETUI0.DLL
   5.0.2134.1 shp 0x751d0000 NETUI1.DLL

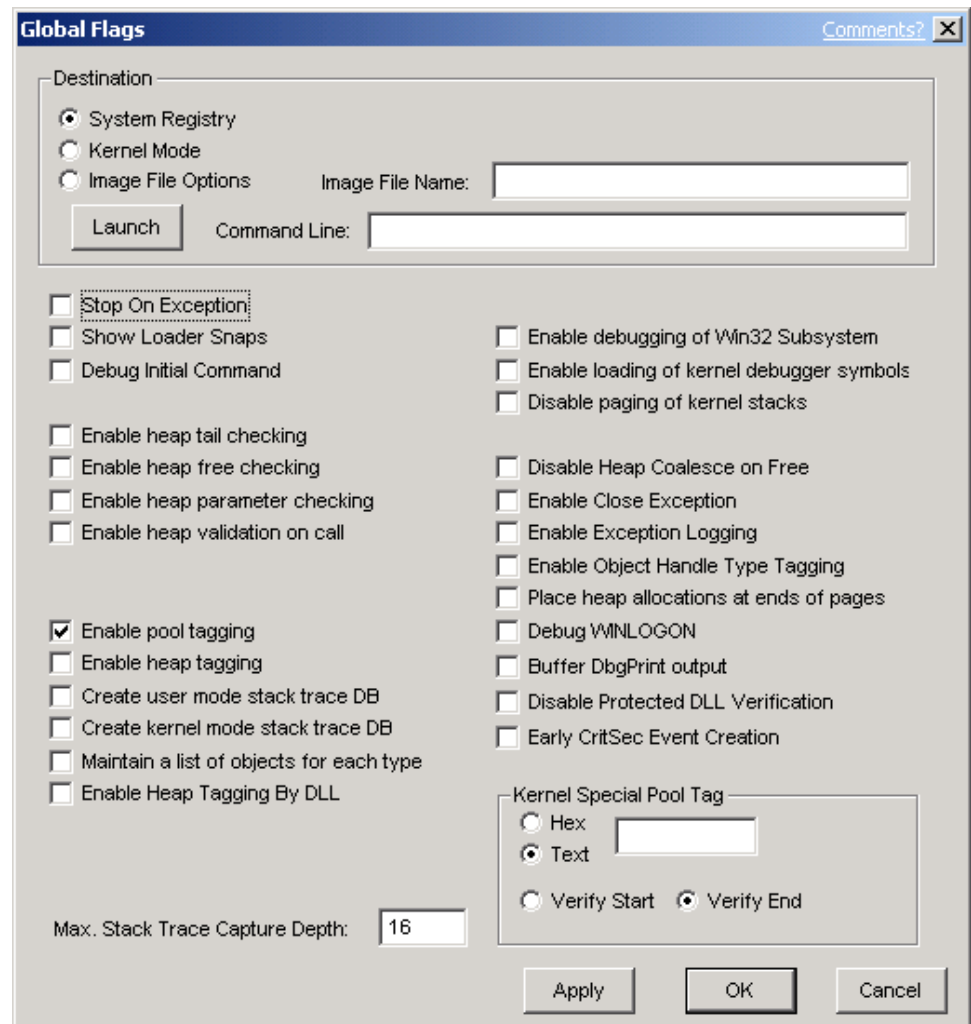
C:\Documents and Settings\Administrator>

```

Exercise 4: Looking at Pool Allocations by Tag

When kernel memory is allocated from either the paged or non-paged pool, a tag is used to help us track who is making those allocations. Pool can be allocated using the *ExAllocatePool* without specifying a tag, but this is rare. In this exercise we will enable pool tagging and take a look at memory being allocated and de-allocated in real time.

1. Select **Start | Run**.
2. Enter `gflags <enter>`
3. Select “Enable Pool Tagging.”
4. Select “Apply”, then select “OK” to continue.



5. This change requires a reboot. To see the error message `poolmon.exe` returns without pool tagging enabled, run `poolmon.exe` from a command prompt:

```
C:\Documents and Settings\toddwe>poolmon
Query pooltags failed (returned: c0000002)
Did you remember to enable pool tagging with gflags.exe and reboot?
```

6. Restart the system.
7. Run poolmon.exe from a command prompt. You should see something like this:

```

Memory: 261676K Avail: 136472K PageFlts: 1720 InRam Krnl: 1612K P:23112K
Commit: 120952K Limit: 632852K Peak: 125452K Pool N: 8560K P:23268K
Tag Type Allocs Frees Diff Bytes Per Alloc
( Paged 34 ( 0) 34 ( 0) 0 0 ( 0) 0
1MEM Nonp 1 ( 0) 0 ( 0) 1 3232 ( 0) 3232
2MEM Nonp 1 ( 0) 0 ( 0) 1 3936 ( 0) 3936
3MEM Nonp 3 ( 0) 0 ( 0) 3 288 ( 0) 96
8042 Paged 8 ( 0) 8 ( 0) 0 0 ( 0) 0
8042 Nonp 5 ( 0) 0 ( 0) 5 4064 ( 0) 812
AGP Paged 5 ( 0) 3 ( 0) 2 384 ( 0) 192
AcdM Nonp 1 ( 0) 0 ( 0) 1 12288 ( 0) 12288
AcdN Nonp 1 ( 0) 0 ( 0) 1 4096 ( 0) 4096
AcpA Nonp 25 ( 0) 22 ( 0) 3 256 ( 0) 85
AcpA Paged 1 ( 0) 0 ( 0) 1 544 ( 0)
544

```

8. Now, change the way you view the information using the keys shown below:

P - Sorts tag list by Paged, Non-Paged, or mixed – cycles between these three.

B - Sorts tags by max byte usage.

M - Sorts tags by max byte allocation.

T - Sort tags alphabetically by tag name.

E - Display totals across bottom – cycles between paged and non-paged totals.

A - Sorts tags by allocation size.

F - Sorts tags by "frees".

S - Sorts tags by the differences of allocs and frees.

Q - Quit.

Module 5 Labs: Crash Dump Analysis I

Lab Objectives



Lab 5 – Crash Dump Analysis I

Exercise 1 – Analyzing a Kernel-Mode Dump File (Part 1)

Estimated Time to Complete this Lab: 75 minutes

Exercise 1: Analyzing a Kernel-Mode Dump File (Part 1)

In this exercise, you will use WinDbg to load the sample kernel mode memory dump files again, this time performing a first level of analysis of each. Following are detailed instructions for the first dump file:

1. Load *memory1.dmp* by repeating steps 1,through 8 from Lab Module 3, Exercise 1.
2. The first thing we need to look at when looking at a memory dump file is the crash dump analysis performed by Windbg itself. Type “!analyze -v <enter>” to see the interpretation made by the debugger:

```
kd> !analyze -v
*****
*
*                               Bugcheck Analysis                               *
*
*****

KMODE_EXCEPTION_NOT_HANDLED (1e)
This is a very common bugcheck. Usually the exception address pinpoints
the driver/function that caused the problem. Always note this address
as well as the link date of the driver/image that contains this address.
Arguments:
Arg1: c0000005, The exception code that was not handled
Arg2: 804676c8, The address that the exception occurred at
Arg3: 00000001, Parameter 0 of the exception
Arg4: 00000000, Parameter 1 of the exception

Debugging Details:
-----

EXCEPTION_CODE: (NTSTATUS) 0xc0000005 - The instruction at "0x%08lx" referenced memory at
"0x%08lx". The memory could not be "%s".

FAULTING_IP:
nt!ExFreePoolWithTag+342
804676c8 890a          mov     [edx],ecx

EXCEPTION_PARAMETER1: 00000001

EXCEPTION_PARAMETER2: 00000000

WRITE_ADDRESS: 00000000

DEFAULT_BUCKET_ID: DRIVER_FAULT

BUGCHECK_STR: 0x1E

TRAP_FRAME: f71efb38 -- (.trap ffffffff71efb38)
ErrCode = 00000002
eax=e1e40fc0 ebx=00000000 ecx=00000000 edx=00000000 esi=e1e40ea0 edi=81438428
eip=804676c8 esp=f71efbac ebp=f71efbd0 iopl=0         nv up ei pl zr na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010246
nt!ExFreePoolWithTag+342:
804676c8 890a          mov     [edx],ecx          ds:0023:00000000=????????
Resetting default context

LAST_CONTROL_TRANSFER: from 804672a2 to 804676c8

STACK_TEXT:
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 80495b70 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
```

```

f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall12+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall12+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
0054fddc 77d4fa80 0054fdfc 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStub+0x5e
0054fe44 77d4f9d7 00000000 000fc328 001112f0 RPCRT4!OSF_SCALL::DispatchHelper+0xa4
0054fe58 77d4f779 00000000 00000000 00000001 RPCRT4!OSF_SCALL::DispatchRPCCall+0x115
0054fe90 77d4f4d8 000fc310 00000003 00000000 RPCRT4!OSF_SCALL::ProcessReceivedPDU+0x43
0054feb0 77d4f0bc 000fc310 0000002c 00015f90 RPCRT4!OSF_SCALL::BeginRpcCall+0xd0
0054ff10 77d4f033 00000000 000fc310 0000002c RPCRT4!OSF_SCONNECTION::ProcessReceiveComplete+0x235
0054ff20 77d5bb62 00088860 0000000c 00000000 RPCRT4!ProcessConnectionServerReceivedEvent+0x1b
0054ff74 77d5ba15 77d4b7bf 00088860 00070178 RPCRT4!LOADABLE_TRANSPORT::ProcessIOEvents+0x9d
0054ff78 77d4b7bf 00088860 00070178 00070640 RPCRT4!ProcessIOEventsWrapper+0x9
0054ffa8 77d4b771 0008b120 0054ffec 77e92ca8 RPCRT4!BaseCachedThreadRoutine+0x4f
0054ffb4 77e92ca8 0008d2c0 00070178 00070640 RPCRT4!ThreadStartRoutine+0x18
0054ffec 00000000 77d4b759 0008d2c0 00000000 KERNEL32!BaseThreadStart+0x52

```

```

FOLLOWUP_IP:
nt!ExFreePool+b
804672a2 c20400          ret     0x4

```

```
FOLLOWUP_NAME: Pool_corruption
```

```
SYMBOL_NAME: nt!ExFreePool+b
```

```
MODULE_NAME: nt
```

```
IMAGE_NAME: ntoskrnl.exe
```

```
DEBUG_FLR_IMAGE_TIMESTAMP: 384d9b17
```

```
STACK_COMMAND: .trap ffffffff71efb38 ; kb
```

```
BUCKET_ID: 0x1E_W_nt!ExFreePool+b
```

```
Followup: Pool_corruption
```

```
-----
```

3. This bug check is “Stop 0x1E” (Unhandled Exception). This tells us the system encountered an exception and none of the exception handlers handled it. So it gets passed on up to the debugger if one is available; if not, as in this case, it crashes the system with a Stop 0x1E.
4. The **!analyze -v** output includes a stack trace. Inspect this for evidence of the faulty routines. In the case of an unhandled exception, the routine at the top of the stack as displayed by **!analyze -v** is the routine that encountered the exception. This is not, however, the routine that was ultimately at fault.
5. Note that the stack trace included within the **!analyze -v** output is slightly different from that displayed by **kb** or **kv**. Use one of these commands to display the stack, and notice the differences.

```

kd> kb
ChildEBP RetAddr  Args to Child
f71efac8 804624cb f71efae4 00000000 f71efb38 nt!KiDispatchException+0x30e
f71efb30 8046247d 81288784 f71efb84 804b0b48 nt!CommonDispatchException+0x4d
f71efb30 804676c8 81288784 f71efb84 804b0b48 nt!KiUnexpectedInterruptTail+0x1f4
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 8049b570 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149

```



```

f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall12+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall12+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100

```

6. Use the **!process 0 0** command to display a brief list of processes and their executable image names.

```

kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 814370a0 SessionId: 0 Cid: 0008 Peb: 00000000 ParentCid: 0000
  DirBase: 00030000 ObjectTable: 81437b88 TableSize: 148.
  Image: System

PROCESS 812b9520 SessionId: 0 Cid: 0098 Peb: 7ffdf000 ParentCid: 0008
  DirBase: 025c9000 ObjectTable: 812c3ca8 TableSize: 33.
  Image: smss.exe

PROCESS 812a17a0 SessionId: 0 Cid: 00b4 Peb: 7ffdf000 ParentCid: 0098
  DirBase: 0325e000 ObjectTable: 812b6228 TableSize: 347.
  Image: csrss.exe

PROCESS 81296320 SessionId: 0 Cid: 00c8 Peb: 7ffdf000 ParentCid: 0098
  DirBase: 035c3000 ObjectTable: 812a5928 TableSize: 352.
  Image: winlogon.exe

PROCESS 81286aa0 SessionId: 0 Cid: 00e4 Peb: 7ffdf000 ParentCid: 00c8
  DirBase: 038c2000 ObjectTable: 81288768 TableSize: 719.
  Image: services.exe

PROCESS 812839a0 SessionId: 0 Cid: 00f8 Peb: 7ffdf000 ParentCid: 00c8
  DirBase: 0393d000 ObjectTable: 81283908 TableSize: 338.
  Image: lsass.exe

PROCESS 8123fd60 SessionId: 0 Cid: 01ac Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 046c5000 ObjectTable: 812682c8 TableSize: 216.
  Image: svchost.exe

PROCESS 81258a40 SessionId: 0 Cid: 01d0 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 04833000 ObjectTable: 8123e468 TableSize: 132.
  Image: SPOOLSV.EXE

PROCESS 8120e340 SessionId: 0 Cid: 0210 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 06369000 ObjectTable: 8122b228 TableSize: 151.
  Image: msdtc.exe

PROCESS 811df300 SessionId: 0 Cid: 0284 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 06734000 ObjectTable: 812109e8 TableSize: 185.
  Image: svchost.exe

PROCESS 811fca80 SessionId: 0 Cid: 0294 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 069c0000 ObjectTable: 812078e8 TableSize: 120.
  Image: llssrv.exe

PROCESS 811c16e0 SessionId: 0 Cid: 02c4 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 069ec000 ObjectTable: 811fc228 TableSize: 30.
  Image: regsvc.exe

PROCESS 811c97a0 SessionId: 0 Cid: 02e0 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 06c1c000 ObjectTable: 811bfb08 TableSize: 93.
  Image: mstask.exe

PROCESS 811db020 SessionId: 0 Cid: 0310 Peb: 7ffdf000 ParentCid: 00e4
  DirBase: 067e3000 ObjectTable: 8128ece8 TableSize: 526.
  Image: inetinfo.exe

```

```

PROCESS 811774a0 SessionId: 0 Cid: 0354 Peb: 7ffdf000 ParentCid: 00e4
DirBase: 03bfa000 ObjectTable: 811eca68 TableSize: 41.
Image: dfssvc.exe

PROCESS 810ce020 SessionId: 0 Cid: 0418 Peb: 7ffdf000 ParentCid: 0458
DirBase: 05d13000 ObjectTable: 810cf9a8 TableSize: 253.
Image: explorer.exe

PROCESS 810b83a0 SessionId: 0 Cid: 0258 Peb: 7ffdf000 ParentCid: 00e4
DirBase: 00a84000 ObjectTable: 811eba28 TableSize: 85.
Image: msieexec.exe

PROCESS 810abae0 SessionId: 0 Cid: 04bc Peb: 7ffdf000 ParentCid: 0498
DirBase: 02f1e000 ObjectTable: 8127c388 TableSize: 356.
Image: IEXPLORE.EXE

PROCESS 810d16c0 SessionId: 0 Cid: 0470 Peb: 7ffdf000 ParentCid: 01ac
DirBase: 04336000 ObjectTable: 81179488 TableSize: 81.
Image: mdm.exe

PROCESS 810a3520 SessionId: 0 Cid: 0518 Peb: 7ffdf000 ParentCid: 00e4
DirBase: 06030000 ObjectTable: 811f6448 TableSize: 120.
Image: clisvcl.exe

PROCESS 810a21c0 SessionId: 0 Cid: 0508 Peb: 7ffdf000 ParentCid: 04c8
DirBase: 07fcc000 ObjectTable: 81189ba8 TableSize: 75.
Image: launch32.exe

PROCESS 81122620 SessionId: 0 Cid: 0544 Peb: 7ffdf000 ParentCid: 0518
DirBase: 06726000 ObjectTable: 8110a368 TableSize: 112.
Image: SMSAPM32.exe

PROCESS 81099340 SessionId: 0 Cid: 01dc Peb: 7ffdf000 ParentCid: 00e4
DirBase: 00a76000 ObjectTable: 810c4868 TableSize: 47.
Image: hinv32.exe

PROCESS 81098020 SessionId: 0 Cid: 0404 Peb: 7ffdf000 ParentCid: 0544
DirBase: 00d13000 ObjectTable: 810fe828 TableSize: 43.
Image: SWDist32.exe

```

7. Use the **!thread** command to display the currently running thread.

```

kd> !thread
THREAD 81272020 Cid e4.15c Teb: 7ffda000 Win32Thread: e1b7e888 RUNNING
IRP List:
  810bcae8: (0006,00b8) Flags: 00000970 Mdl: 00000000
  810ab008: (0006,00b8) Flags: 00000800 Mdl: 00000000
  8121a3a8: (0006,00b8) Flags: 00000970 Mdl: 00000000
  810d8e68: (0006,00b8) Flags: 00000800 Mdl: 00000000
  81219228: (0006,00b8) Flags: 00000970 Mdl: 00000000
  810fea28: (0006,00b8) Flags: 00000970 Mdl: 00000000
  810c44c8: (0006,00b8) Flags: 00000970 Mdl: 00000000
  81254808: (0006,00b8) Flags: 00000970 Mdl: 00000000
  81165448: (0006,00b8) Flags: 00000970 Mdl: 00000000
  8126c008: (0006,00b8) Flags: 00000970 Mdl: 00000000
  811f12a8: (0006,00b8) Flags: 00000970 Mdl: 00000000
  8121d748: (0006,00b8) Flags: 00000970 Mdl: 00000000
  8125ea48: (0006,00b8) Flags: 00000970 Mdl: 00000000
  8117b308: (0006,00b8) Flags: 00000970 Mdl: 00000000
  811b4468: (0006,00b8) Flags: 00000970 Mdl: 00000000
  812ae848: (0006,00b8) Flags: 00000970 Mdl: 00000000
Not impersonating
Owning Process 81286aa0
WaitTime (seconds) 103380
Context Switch Count 5193 LargeStack
UserTime 0:00:00.0260
KernelTime 0:00:00.0630
Start Address KERNEL32!BaseThreadStartThunk (0x77e92c50)
Win32 Start Address RPCRT4!ThreadStartRoutine (0x77d4b759)
Stack Init f71f0000 Current f71ef2bc Base f71f0000 Limit f71ed000 Call 0
Priority 10 BasePriority 9 PriorityDecrement 0 DecrementCount 0

ChildEBP RetAddr Args to Child

```

```

f71efac8 804624cb f71efae4 00000000 f71efb38 nt!KiDispatchException+0x30e
f71efb30 8046247d 81288784 f71efb84 804b0b48 nt!CommonDispatchException+0x4d
f71efb30 804676c8 81288784 f71efb84 804b0b48 nt!KiUnexpectedInterruptTail+0x1f4
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 80495b70 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall2+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall2+0x17
0054fd64 77d452ef 010065fd 0054fdfc 81272020 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
0054fddc 77d4fa80 0054fdfc 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStub+0x5e
0054fe44 77d4f9d7 00000000 000fc328 001112f0 RPCRT4!OSF_SCALL::DispatchHelper+0xa4
0054fe58 77d4f779 00000000 00000000 00000001 RPCRT4!OSF_SCALL::DispatchRPCCall+0x115
0054fe90 77d4f4d8 000fc310 00000003 00000000 RPCRT4!OSF_SCALL::ProcessReceivedPDU+0x43
0054feb0 77d4f0bc 000fc310 0000002c 00015f90 RPCRT4!OSF_SCALL::BeginRpcCall+0xd0
0054ff10 77d4f033 00000000 000fc310 0000002c RPCRT4!OSF_SCONNECTION::ProcessReceiveComplete+0x235
0054ff20 77d5bb62 00088860 0000000c 00000000 RPCRT4!ProcessConnectionServerReceivedEvent+0x1b
0054ff74 77d5ba15 77d4b7bf 00088860 00070178 RPCRT4!LOADABLE_TRANSPORT::ProcessIOEvents+0x9d
0054ff78 77d4b7bf 00088860 00070178 00070640 RPCRT4!ProcessIOEventsWrapper+0x9
0054ffa8 77d4b771 0008b120 0054ffec 77e92ca8 RPCRT4!BaseCachedThreadRoutine+0x4f
0054fffb 77e92ca8 0008d2c0 00070178 00070640 RPCRT4!ThreadStartRoutine+0x18
0054ffec 00000000 77d4b759 0008d2c0 00000000 KERNEL32!BaseThreadStart+0x52

```

8. Use the **!process** command to display complete details for the process that owns that thread. Inspecting the output from **!thread**, we find that the owning process address is 81286aa0, so **!process 81286aa0** can be used,
9. Use the **!drivers** command to display the list of drivers that were loaded in the system.

```

kd> !drivers
System Driver and Image Summary
Base      Code Size      Data Size      Image Name      Creation Time
80400000 142dc0 (1292 k) 4d680 (310 k) ntoskrnl.exe    Tue Dec 07 15:41:11 1999
80062000 13c40 ( 80 k) 34e0 ( 14 k) hal.dll        Sat Oct 30 15:48:14 1999
f7410000 1760 ( 6 k) 1000 ( 4 k) BOOTVID.DLL   Wed Nov 03 17:24:33 1999
f7000000 bdc0 ( 48 k) 22a0 ( 9 k) pci.sys       Wed Oct 27 16:11:08 1999
f7010000 99c0 ( 39 k) 18e0 ( 7 k) isapnp.sys    Sat Oct 02 13:00:35 1999
f75c8000 760 ( 2 k) 520 ( 2 k) intelide.sys  Thu Oct 28 16:20:03 1999
f7280000 42e0 ( 17 k) e80 ( 4 k) PCIIDEX.SYS  Wed Oct 27 16:02:19 1999
f7288000 64a0 ( 26 k) a20 ( 3 k) MountMgr.sys  Fri Oct 22 15:48:06 1999
fcd4e000 192c0 (101 k) 2b00 (11 k) ftdisk.sys    Mon Nov 22 11:36:23 1999
f7500000 12e0 ( 5 k) 640 ( 2 k) Diskperf.sys  Thu Sep 30 17:30:40 1999
f75c9000 740 ( 2 k) 560 ( 2 k) WMILIB.SYS    Sat Sep 25 11:36:47 1999
f7502000 d80 ( 4 k) b40 ( 3 k) dmload.sys    Tue Nov 30 11:47:49 1999
fcd2c000 1a380 (105 k) 6be0 (27 k) dmio.sys      Tue Nov 30 11:47:49 1999
f7414000 21a0 ( 9 k) 720 ( 2 k) PartMgr.sys   Thu Oct 14 17:59:16 1999
fcd17000 11b80 ( 71 k) 2c00 (11 k) atapi.sys     Sat Dec 04 12:19:32 1999
f7290000 58e0 ( 23 k) e60 ( 4 k) disk.sys      Fri Oct 22 15:27:46 1999
f7020000 6d20 ( 28 k) 1240 ( 5 k) CLASSPNP.SYS Wed Oct 06 16:55:45 1999
fcd05000 e100 ( 57 k) 3be0 (15 k) Dfs.sys       Tue Nov 30 16:23:01 1999
fccf4000 c820 ( 51 k) 4000 (16 k) KSecDD.sys    Fri Oct 22 16:38:14 1999
fcc71000 7ae60 (492 k) 7840 (31 k) Ntfs.sys      Mon Nov 29 23:37:55 1999
fcc48000 23320 (141 k) 5680 (22 k) NDIS.sys      Mon Nov 29 23:37:30 1999
fcc32000 117e0 ( 70 k) 3680 (14 k) Mup.sys       Fri Nov 05 14:31:58 1999
f7298000 43e0 ( 17 k) 8e0 ( 3 k) agp440.sys    Tue Sep 28 16:37:32 1999
f75cb000 2e0 ( 1 k) 4a0 ( 2 k) audstub.sys   Sat Sep 25 11:35:33 1999
f7050000 b680 ( 46 k) c20 ( 4 k) rasl2tp.sys   Mon Nov 29 23:09:07 1999
f7484000 1700 ( 6 k) 840 ( 3 k) ndistapi.sys  Tue Oct 12 16:54:43 1999
f6fc9000 13400 ( 77 k) 2ac0 (11 k) ndiswan.sys   Mon Nov 29 23:09:01 1999

```

```

f7490000 2d20 ( 12 k) f20 ( 4 k) TDI.SYS Mon Nov 29 23:19:49 1999
f7060000 a620 ( 42 k) 1100 ( 5 k) raspttp.sys Mon Nov 29 23:09:13 1999
f7358000 3740 ( 14 k) 9c0 ( 3 k) ptilink.sys Wed Oct 13 16:29:00 1999
f7368000 33e0 ( 13 k) a40 ( 3 k) raspti.sys Fri Oct 08 13:45:10 1999
f7070000 c5e0 ( 50 k) 20e0 ( 9 k) parallel.sys Fri Oct 22 15:00:54 1999
f7080000 a000 ( 40 k) 20c0 ( 9 k) VIDEOPRT.SYS Sat Nov 06 13:55:20 1999
f6f8f000 c900 ( 51 k) 4540 ( 18 k) atiragem.sys Fri Nov 05 15:43:11 1999
f73a8000 5d00 ( 24 k) 980 ( 3 k) cdrom.sys Wed Oct 27 16:46:36 1999
f73c8000 3da0 ( 16 k) ec0 ( 4 k) USB.D.SYS Sat Oct 09 13:41:58 1999
f73b8000 70a0 ( 29 k) 900 ( 3 k) uhcd.sys Tue Oct 05 13:45:47 1999
f7090000 d000 ( 52 k) 1c00 ( 7 k) el90xbc5.sys Tue Oct 19 10:09:18 1999
f6f4e000 17b40 ( 95 k) 3cc0 ( 16 k) KS.SYS Tue Nov 30 00:51:38 1999
f6f6a000 1c740 ( 114 k) 7a40 ( 31 k) portcls.sys Fri Nov 05 23:53:25 1999
f70a0000 6720 ( 26 k) 3420 ( 14 k) esl371mp.sys Fri Nov 05 14:33:27 1999
f75cc000 580 ( 2 k) 540 ( 2 k) swenum.sys Sat Sep 25 11:36:31 1999
f6f39000 dc0 ( 4 k) 137c0 ( 78 k) update.sys Mon Oct 25 12:28:24 1999
f70b0000 8460 ( 34 k) 2940 ( 11 k) i8042prt.sys Wed Dec 01 23:34:06 1999
f72f8000 44a0 ( 18 k) 1720 ( 6 k) kbdclass.sys Tue Oct 26 16:12:37 1999
f7308000 48c0 ( 19 k) 1540 ( 6 k) parport.sys Sat Sep 25 11:36:43 1999
f70c0000 c200 ( 49 k) 2dc0 ( 12 k) serial.sys Mon Oct 25 12:27:55 1999
f74ac000 2a60 ( 11 k) 720 ( 2 k) serenum.sys Tue Oct 19 15:36:55 1999
f7328000 0 ( 0 k) 0 ( 0 k) fdc.sys unavailable
f7338000 3ac0 ( 15 k) 1660 ( 6 k) mouclass.sys Fri Oct 01 16:33:11 1999
f70d0000 8420 ( 34 k) 1600 ( 6 k) NDPProxy.SYS Thu Sep 30 16:25:35 1999
f7350000 4860 ( 19 k) 1e80 ( 8 k) EFS.SYS Sat Oct 09 13:59:21 1999
f70e0000 8940 ( 35 k) f20 ( 4 k) usbbhub.sys Fri Nov 12 15:29:21 1999
f74c4000 1b00 ( 7 k) 680 ( 2 k) gameenum.sys Sat Sep 25 11:35:57 1999
f7380000 3a60 ( 15 k) d40 ( 4 k) flpydisk.sys Mon Sep 27 20:47:21 1999
f750c000 12a0 ( 5 k) 640 ( 2 k) Fs_Rec.SYS Sat Sep 25 11:39:38 1999
f75d0000 2a0 ( 1 k) 480 ( 2 k) Null.SYS Sat Sep 25 11:34:58 1999
f75d1000 720 ( 2 k) 540 ( 2 k) Beep.SYS Wed Oct 20 15:18:59 1999
f74d4000 2800 ( 10 k) aa0 ( 3 k) vga.sys Sat Sep 25 11:37:40 1999
f75d2000 4a0 ( 2 k) 800 ( 2 k) mnmdm.sys Sat Sep 25 11:37:40 1999
f73a0000 4140 ( 17 k) e20 ( 4 k) Msfs.SYS Tue Oct 26 16:21:32 1999
f7100000 7940 ( 31 k) 1380 ( 5 k) Npfs.SYS Sat Oct 09 16:58:07 1999
f7514000 1380 ( 5 k) 7e0 ( 2 k) rasacd.sys Sat Sep 25 11:41:23 1999
f6eae000 43500 ( 270 k) 7020 ( 29 k) tcpip.sys Mon Nov 29 23:38:42 1999
f7110000 7400 ( 29 k) 1000 ( 4 k) msgpc.sys Mon Nov 29 23:37:21 1999
f72b8000 63e0 ( 25 k) 12a0 ( 5 k) wanarp.sys Sat Oct 30 15:36:06 1999
f6e89000 21980 ( 135 k) 2840 ( 11 k) netbt.sys Mon Nov 29 23:37:39 1999
f7120000 6f20 ( 28 k) fa0 ( 4 k) netbios.sys Tue Oct 12 12:34:19 1999
f6e67000 1de80 ( 120 k) 3220 ( 13 k) rdbss.sys Tue Nov 30 00:52:29 1999
f6dcf000 52ca0 ( 332 k) a180 ( 41 k) mrxsm.sys Tue Nov 30 00:52:10 1999
f75d3000 740 ( 2 k) 560 ( 2 k) dump_WMILIB.SYS Sat Sep 25 11:36:47 1999
f6dba000 11b80 ( 71 k) 2c00 ( 11 k) dump_atapi.sys Sat Dec 04 12:19:32 1999
a0000000 177d60 (1504 k) 2d5a0 (182 k) win32k.sys Tue Nov 30 00:51:03 1999
f6d6f000 1f360 ( 125 k) 2c00 ( 11 k) atiraged.dll Tue Nov 30 01:31:17 1999
f643d000 170c0 ( 93 k) 25a0 ( 10 k) afd.sys Mon Nov 29 23:12:04 1999
f6362000 fc40 ( 64 k) 2120 ( 9 k) wdmaud.sys Wed Oct 27 11:40:45 1999
f652f000 9520 ( 38 k) 1f40 ( 8 k) sysaudio.sys Mon Oct 25 12:28:14 1999
f7574000 d40 ( 4 k) 860 ( 3 k) ParVdm.SYS Mon Sep 27 20:28:16 1999
f60aa000 35520 ( 214 k) 59e0 ( 23 k) srv.sys Mon Nov 29 23:38:21 1999
f7250000 d820 ( 55 k) 1280 ( 5 k) CdFs.SYS Mon Oct 25 12:23:52 1999
f604c000 21360 ( 133 k) 2a00 ( 11 k) Fastfat.SYS Mon Oct 25 12:20:50 1999
f6096000 2160 ( 9 k) ac0 ( 3 k) spud.sys Fri Nov 19 15:36:27 1999
f5f36000 11f20 ( 72 k) 2ac0 ( 11 k) ipsec.sys Mon Nov 29 23:08:54 1999
f5eea000 16f60 ( 92 k) ccc0 ( 52 k) kmixer.sys Tue Nov 09 22:52:30 1999

```

No unloaded module list present

Loading User Symbols

```

01000000 14a00 ( 83 k) e00 ( 4 k) services.exe Mon Oct 25 12:20:14 1999
77f80000 4a400 ( 297 k) 2b000 (172 k) ntdll.dll Wed Oct 27 13:06:08 1999
77d40000 67a00 ( 415 k) 4c00 ( 19 k) RPCRT4.DLL Thu Dec 02 15:29:06 1999
77e80000 5d200 ( 373 k) 55800 (342 k) KERNEL32.DLL Tue Nov 30 23:37:24 1999
77db0000 4f400 ( 317 k) 7e00 ( 32 k) ADVAPI32.DLL Tue Nov 30 23:37:24 1999
75170000 46600 ( 282 k) 5200 ( 21 k) NETAPI32.DLL Sat Dec 04 18:28:08 1999
78000000 32000 ( 200 k) 13000 ( 76 k) MSVCRT.DLL Wed Sep 29 18:51:35 1999
77be0000 a600 ( 42 k) 1000 ( 4 k) SECUR32.DLL Tue Nov 30 23:37:25 1999
751c0000 1e00 ( 8 k) 800 ( 2 k) NETRAP.DLL Tue Nov 30 01:31:07 1999
75150000 8600 ( 34 k) 2e00 ( 12 k) SAMLIB.DLL Tue Nov 30 01:31:08 1999
75030000 f600 ( 62 k) 1c00 ( 7 k) WS2_32.DLL Tue Nov 30 01:31:09 1999
75020000 3600 ( 14 k) c00 ( 3 k) WS2HELP.DLL Tue Nov 30 01:31:09 1999
77950000 1ce00 ( 116 k) 9c00 ( 39 k) WLDAP32.DLL Tue Nov 30 23:37:27 1999
77980000 1d800 ( 118 k) 2c00 ( 11 k) DNSAPI.DLL Tue Nov 30 23:37:27 1999

```

75050000	2200 (9 k)	2e00 (12 k)	WSOCK32.DLL	Tue Nov 30 01:31:09 1999
77e10000	57200 (349 k)	b400 (45 k)	USER32.DLL	Tue Nov 30 23:37:24 1999
77f40000	36600 (218 k)	2800 (10 k)	GDI32.DLL	Fri Nov 12 00:44:52 1999
767a0000	12800 (74 k)	2e00 (12 k)	UMPNPMGR.DLL	Tue Nov 30 23:37:35 1999
77c10000	4de00 (312 k)	c200 (49 k)	USERENV.DLL	Tue Nov 30 23:37:25 1999
76810000	2e600 (186 k)	9c00 (39 k)	SCESRV.DLL	Tue Nov 30 23:37:35 1999
77bf0000	cc00 (51 k)	1000 (4 k)	NTDSAPI.DLL	Tue Nov 30 23:37:25 1999
76890000	9600 (38 k)	1400 (5 k)	eventlog.dll	Tue Nov 30 23:37:35 1999
77360000	13400 (77 k)	2c00 (11 k)	dhcpcsvc.dll	Tue Nov 30 23:37:29 1999
77520000	e00 (4 k)	800 (2 k)	ICMP.DLL	Tue Nov 30 23:37:29 1999
77340000	d800 (54 k)	3400 (13 k)	IPHLAPI.DLL	Tue Nov 30 23:37:29 1999
77320000	12400 (73 k)	1400 (5 k)	MPRAPI.DLL	Tue Nov 30 23:37:29 1999
77a50000	e0200 (897 k)	11600 (70 k)	OLE32.DLL	Tue Nov 30 23:37:25 1999
779b0000	85000 (532 k)	f000 (60 k)	OLEAUT32.DLL	Tue Nov 30 23:37:26 1999
773b0000	21600 (134 k)	9e00 (40 k)	ACTIVEDS.DLL	Tue Nov 30 23:37:29 1999
77380000	1de00 (120 k)	1c00 (7 k)	ADSLDPC.DLL	Tue Nov 30 23:37:29 1999
77830000	9a00 (39 k)	1000 (4 k)	RTUTILS.DLL	Tue Nov 30 23:37:27 1999
77890000	60a00 (387 k)	29a00 (167 k)	SETUPAPI.DLL	Tue Nov 30 23:37:27 1999
774e0000	2b800 (174 k)	3a00 (15 k)	RASAPI32.DLL	Tue Nov 30 23:37:29 1999
774c0000	c600 (50 k)	1200 (5 k)	RASMAN.DLL	Tue Nov 30 23:37:29 1999
77530000	1ca00 (115 k)	2400 (9 k)	TAPI32.DLL	Tue Nov 30 23:37:28 1999
77b50000	64400 (401 k)	22800 (138 k)	COMCTL32.DLL	Tue Nov 30 23:37:25 1999
77c70000	42600 (266 k)	4600 (18 k)	SHLWAPI.DLL	Tue Nov 30 23:37:25 1999
77cc0000	6d600 (438 k)	f000 (60 k)	CLBCATQ.DLL	Tue Nov 30 23:37:24 1999
768a0000	13200 (77 k)	2c00 (11 k)	dnssrslvr.dll	Tue Nov 30 23:37:35 1999
76880000	1800 (6 k)	800 (2 k)	lmhsvc.dll	Tue Nov 30 23:37:35 1999
74fd0000	c400 (49 k)	1000 (4 k)	msafsd.dll	Tue Nov 30 01:31:09 1999
75010000	3000 (12 k)	e00 (4 k)	wshtcpip.dll	Tue Nov 30 01:31:09 1999
65780000	7a00 (31 k)	1000 (4 k)	WINSTA.DLL	Thu Dec 02 15:30:10 1999
76770000	15800 (86 k)	2200 (9 k)	wkssvc.dll	Tue Nov 30 23:37:36 1999
76670000	7c00 (31 k)	2e00 (12 k)	CRYPTDLL.DLL	Tue Nov 30 23:37:36 1999
77840000	7600 (30 k)	1400 (5 k)	rnr20.dll	Tue Nov 30 23:37:27 1999
74b40000	3200 (13 k)	1400 (5 k)	alrsvc.dll	Tue Nov 30 01:31:14 1999
768c0000	1c00 (7 k)	e00 (4 k)	dmserver.dll	Tue Nov 30 23:37:35 1999
770b0000	0 (0 k)	3c00 (15 k)	CFGMR32.DLL	Tue Nov 30 23:37:31 1999
767e0000	ea00 (59 k)	5000 (20 k)	Srvsvc.dll	Tue Nov 30 23:37:35 1999
77800000	16c00 (91 k)	4600 (18 k)	WINSPOOL.DRV	Tue Nov 30 23:37:27 1999
76870000	6c00 (27 k)	1800 (6 k)	msgsvc.dll	Tue Nov 30 23:37:35 1999
768d0000	d400 (53 k)	1e00 (8 k)	cryptsvc.dll	Tue Nov 30 23:37:35 1999
76850000	12400 (73 k)	9600 (38 k)	psbase.dll	Tue Nov 30 23:37:35 1999
74ff0000	b600 (46 k)	4600 (18 k)	mswsock.dll	Tue Nov 30 01:31:09 1999
76800000	2a00 (11 k)	1a00 (7 k)	seclogon.dll	Tue Nov 30 23:37:35 1999
777e0000	3c00 (15 k)	c00 (3 k)	winrnr.dll	Tue Nov 30 23:37:27 1999
777f0000	e00 (4 k)	800 (2 k)	rasadhlp.dll	Tue Nov 30 23:37:27 1999
767c0000	14200 (81 k)	1c00 (7 k)	trkwks.dll	Tue Nov 30 23:37:35 1999
76790000	8800 (34 k)	3600 (14 k)	w32time.dll	Tue Nov 30 23:37:36 1999
768f0000	a800 (42 k)	1400 (5 k)	browser.dll	Tue Nov 30 23:37:34 1999
7ca00000	1c000 (112 k)	4000 (16 k)	rsabase.dll	Tue Oct 12 16:30:37 1999
77440000	69600 (422 k)	ac00 (43 k)	CRYPT32.dll	Tue Nov 30 23:37:29 1999
77430000	c200 (49 k)	600 (2 k)	MSASN1.DLL	Tue Nov 30 23:37:29 1999
76750000	7800 (30 k)	9e00 (40 k)	wmicore.dll	Tue Nov 30 23:37:36 1999
70170000	b0800 (706 k)	68600 (418 k)	ESENT.dll	Tue Nov 30 01:32:20 1999
775a0000	11a800 (1130 k)	123800 (1166 k)	shell32.dll	Tue Nov 30 23:37:27 1999

10. Load the additional *memoryN.dmp* files and perform a similar level of analysis on each. .

Module 6 Labs: Kernel Debugging I

Lab Objectives



Lab 6 – Kernel Debugging I

Exercise 1 – Setting Up a Kernel Debugging Session

Exercise 2 – Debugging a “System Hang” Caused by a High-Priority Thread

Exercise 3 – Debugging a “System Hang” Caused by a Driver Looping at High IRQL

Estimated Time to Complete this Lab: minutes

Exercise 1: Setting up a Kernel Debugging Session

In this exercise you will set up and verify a two-machine, “live” kernel debugging environment.

This lab assumes that you have completed the labs in Modules 2 and 3.

Part 1: Setting up the target computer in debug mode

This lab demonstrates how to configure the target computer to startup in “debug mode.” (With Windows 2000 and later versions, you can also select F8 during startup and select debug mode.)

1. Select **Start | Run** and type “cmd” into the test box and select OK or press <enter>.
2. If necessary, change to the c: drive, by typing “c:<enter>”.
3. Change into the root directory of c: drive, by typing “cd\<enter>”.
4. Type “attrib boot.ini -r -s -h <enter>”.
5. Type “notepad boot.ini <enter>”.

The *boot.ini* file will look similar to this:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
```

Note This sample is from an x86 computer running Windows 2000 Professional.

Note The lines in *boot.ini* can be quite long. Depending on your settings in Notepad, Notepad may or may not display them “wrapped” to fit the width of the window. In later steps you will save a modified version of this file. Do not save the file with any new line breaks!

6. If there is more than one line following [operating systems] in *boot.ini*, each specifies a disk partition and directory from which a Windows operating system may be started. Identify one of these which you want to be able to boot in debug mode.
7. Make a copy of the line specifying the operating system you want to be able to boot in debug mode. The sample file shown above would now look like this:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
```

8. Append **/debugport=com1** to the end of the new line you just added to the file.

Here is the result of the change to the sample *boot.ini* file after it has been modified by steps one through eight:

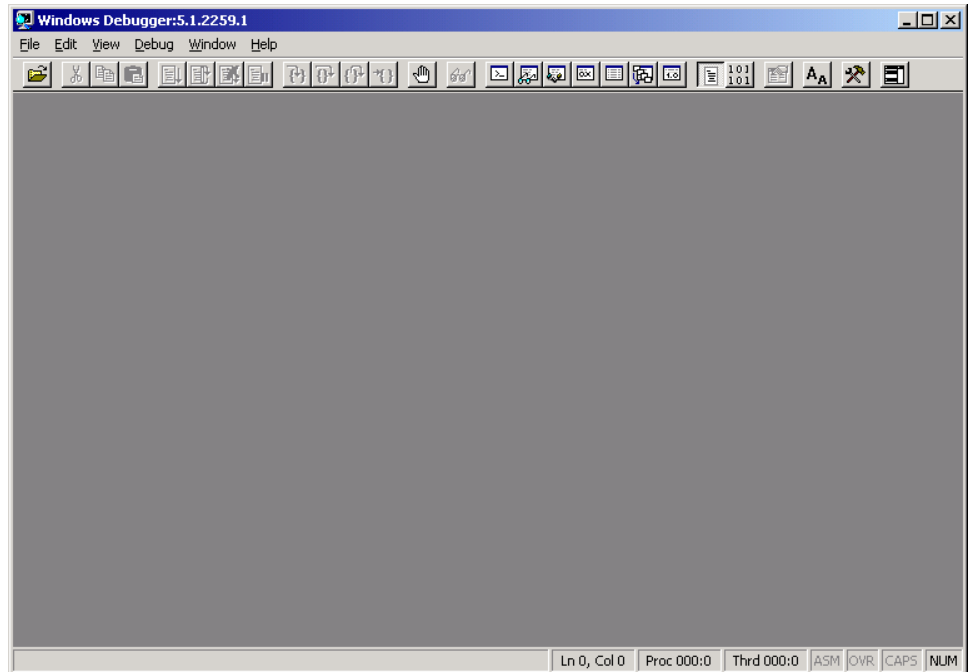
```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional" /fastdetect
/debugport=Com1
```

Note: In the actual file, **there must be no line break** between “/fastdetect” and “/debugport=Com1”. The line break in the preceding is an artifact of the printing requirements for this document.

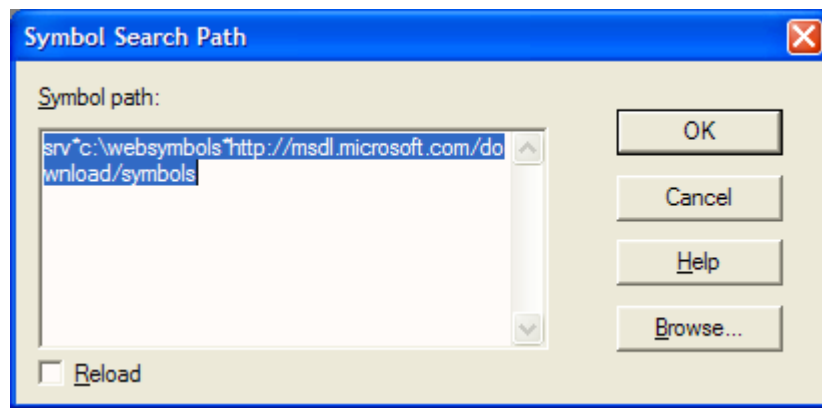
9. Save the *boot.ini* file and quit notepad.
10. Shut down and restart Windows on the target computer. You’ll notice that a boot option screen appears, offering you at least two choices, one that includes the notation “debugger enabled” and one that does not. Select the one that says “debugger enabled”.

Part 2: Starting a Debug Session With WinDbg on the Host System

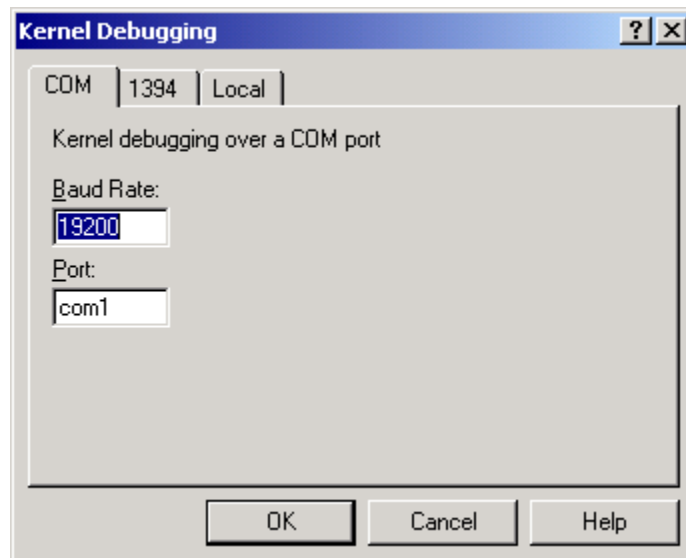
11. Select **Start | Programs | Microsoft Debugging Tools | WinDbg**.



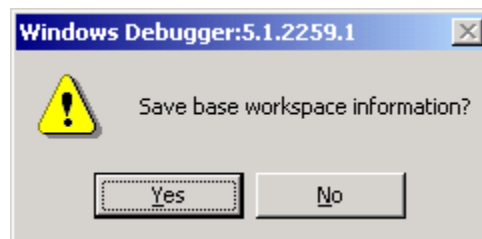
12. Select **File | Symbols File Path...** or press **Ctrl+S**.
13. Enter “**srv*c:\websymbols*http://msdl.microsoft.com/download/symbols**” and then select OK. (Your instructor may supply a different symbol path.)



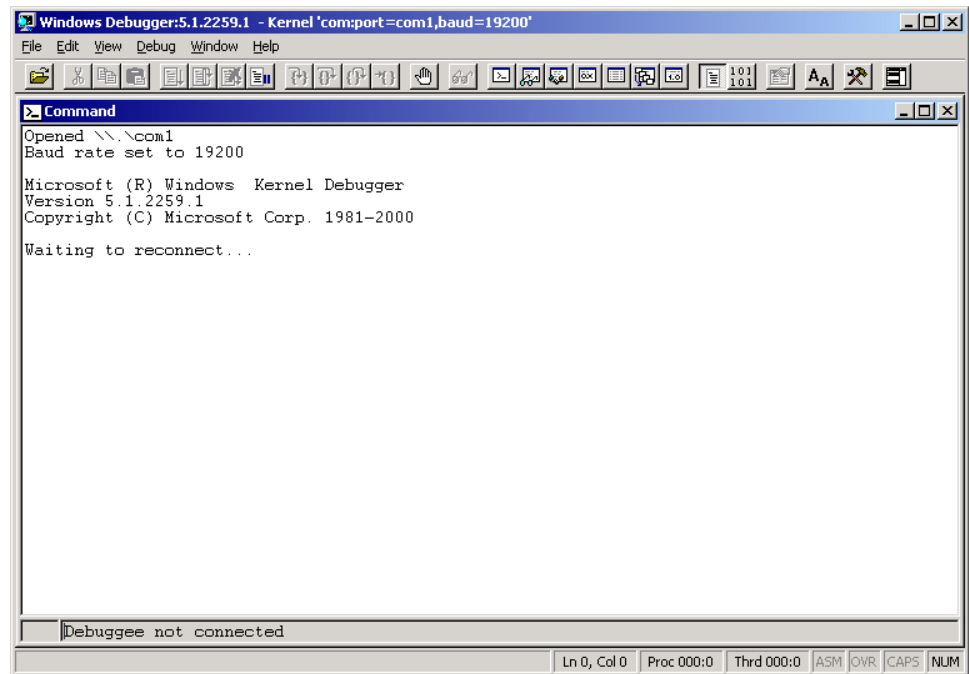
14. Select **File | Kernel Debug...** or press **Ctrl+K**
15. Enter *19200* for Baud Rate and *COM1* for Port



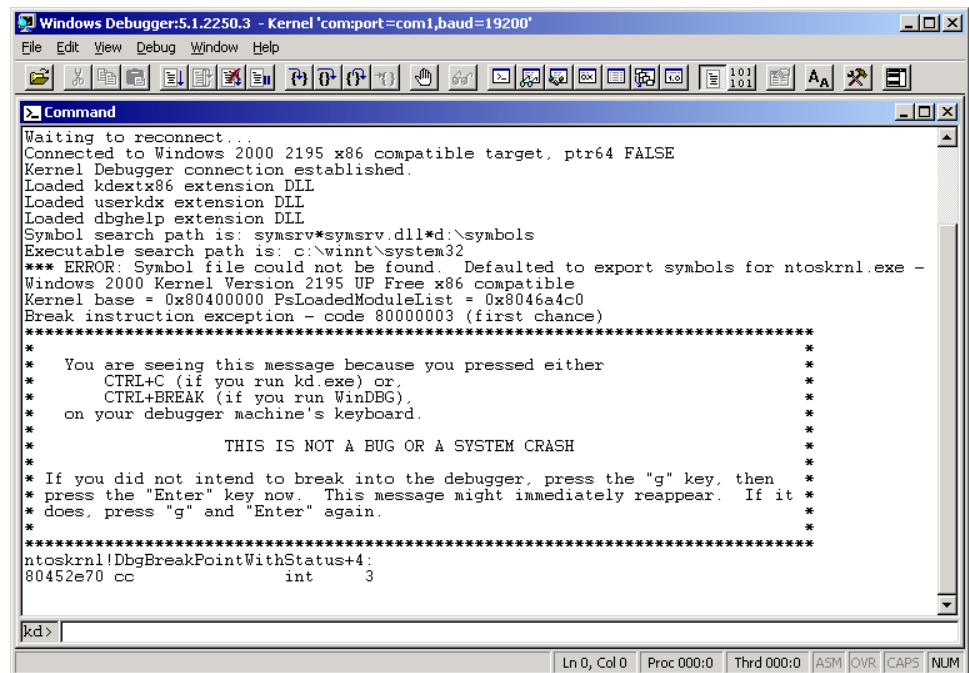
16. Select OK
17. You will be prompted to save base workspace information. If you are certain the symbol path is correct, select Yes; otherwise, select No.



The Windows Debugger will look something like this:



18. Select **Debug | Break** or press **Ctrl+Break**



The debugger's output should look similar to the above.

19. Observe that the target system is now "frozen," completely unresponsive to any sort of input.
20. Type **!sym noisy** and press <enter>. This will allow you to see the symbols when they are loaded.
21. Type **.reload** and press enter.

The screenshot shows the Windows Debugger interface. The title bar reads "Windows Debugger:5.1.2250.3 - Kernel 'comport=com1,baud=19200'". The menu bar includes File, Edit, View, Debug, Window, and Help. The Command window contains the following text:

```

*****
*
*   You are seeing this message because you pressed either
*   CTRL+C (if you run kd.exe) or,
*   CTRL+BREAK (if you run WinDBG),
*   on your debugger machine's keyboard.
*
*   THIS IS NOT A BUG OR A SYSTEM CRASH
*
*   If you did not intend to break into the debugger, press the "g" key, then
*   press the "Enter" key now. This message might immediately reappear. If it
*   does, press "g" and "Enter" again.
*
*****
ntoskrnl!RtlpBreakWithStatusInstruction:
80452e70 cc          int     3
kd> !sym noisy
Noisy mode on.
kd> .reload
Connected to Windows 2000 2195 x86 compatible target, ptr64 FALSE
SYMSRV: d:\symsrv\ntoskrnl.dbg\384d9b17190900\ntoskrnl.dbg - OK
DBGHELP: FindDebugInfoFileEx-> Looking for d:\symsrv\ntoskrnl.dbg\384d9b17190900\ntoskrnl.d
DBGHELP: FindExecutableImageEx-> Looking for D:\debug\ntoskrnl.exe... no file
DBGHELP: LocatePDB-> Looking for d:\symsrv\ntoskrnl.dbg\384d9b17190900\ntoskrnl.pdb... file
SYMSRV: d:\symsrv\ntoskrnl.pdb\38237d2054\ntoskrnl.pdb - OK
DBGHELP: LocatePDB-> Looking for d:\symsrv\ntoskrnl.pdb\38237d2054\ntoskrnl.pdb... OK
Loading Kernel Symbols
.....

```

The status bar at the bottom shows "Ln 0, Col 0", "Proc 000:0", "Thrd 000:0", and tabs for ASM, OVR, CAPS, and NUM.

22. Use the **kb** (stack backtrace) command to display the current stack. Inspect the sequence of procedure calls.
23. Use the **!thread** extension to display information about the current thread. Use the **!process** extension to display information about the process that owns that thread.
24. What was happening on the target system when you broke into it?
25. Use the debugger's **s (step)** command to execute a few instructions. Notice the stack changing. What is happening on the target system?
26. Use the debugger's **g** (continue from breakpoint) command, or select **Debug | Go**, to allow the target to continue executing, then select **Debug | Break** or press **Ctrl+Break** to break into the target again. Use the **kb** and **!thread** commands to examine the current thread. Has anything changed?
27. Use some of the other debugger commands to examine the state of the target system.
28. When finished, use the debugger's **g** (continue from breakpoint) command, or select **Debug | Go**, to allow the target to continue executing, then select **File | Exit**, or use **Alt-F4**, to exit WinDbg.

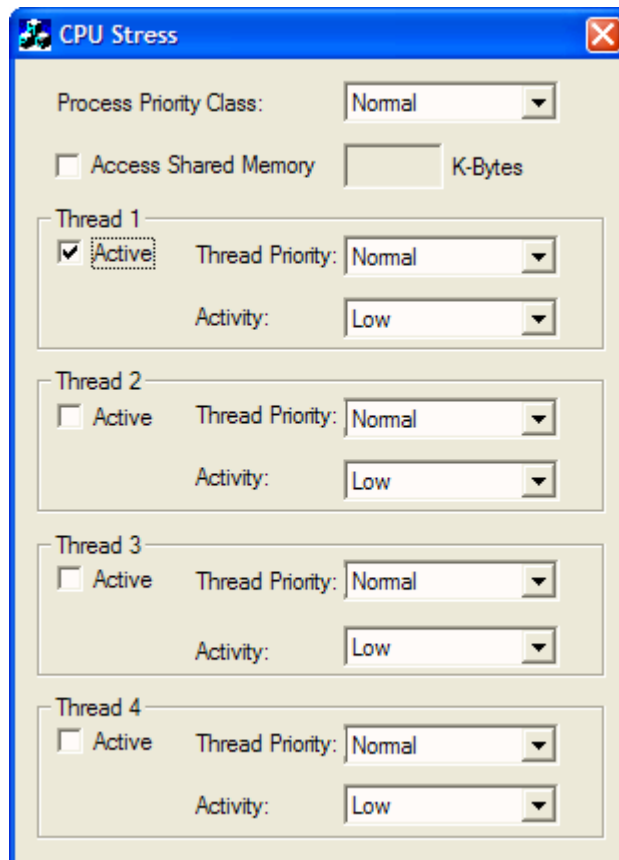
Exercise 2: Debugging a “System Hang” Caused by a High-Priority Thread

In this lab you will use a WinDbg kernel debugging session to determine the cause of a system hang.

This lab assumes that you have completed Exercise 1 in this module.

Part 1: Creating the Problem Scenario

1. Ensure that your target system was booted with debugger enabled. If not, reboot it and select the “debugger enabled” option from the boot menu.
2. On the target system, start the Resource Kit utility *cpustres.exe*. Select **Start | Run**, type “cpustres” into the text box and select OK or press <enter>. The following dialog box should appear:



3. If the target system is a multiprocessor system, check as many of the “Thread 1 – active”, “Thread 2 – active”, etc., boxes, as there are CPUs.
4. Set the “Activity” level of Thread 1 to “Maximum.”
5. On the target system, start Task Manager by right-clicking in an empty area of the Task Bar, and selecting Task Manager (or use Ctrl-Alt-Delete). Within Task Manager, select the Processes tab. Locate the process running CpuStres.Exe. Note that its CPU usage is close to 100%.

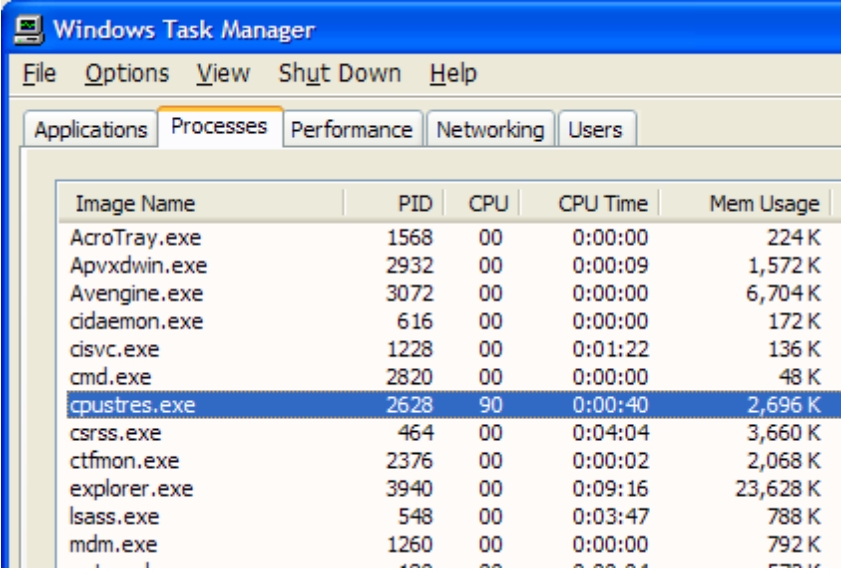
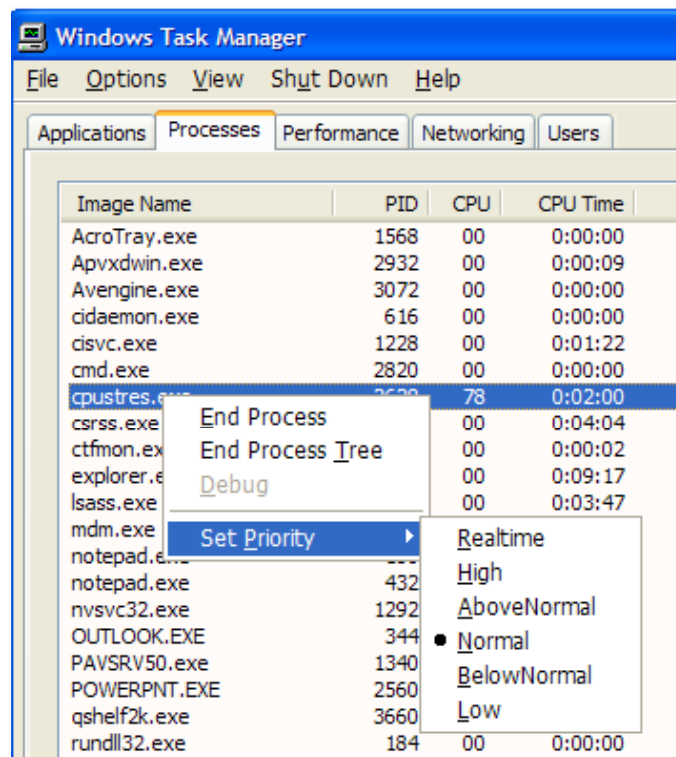


Image Name	PID	CPU	CPU Time	Mem Usage
AcroTray.exe	1568	00	0:00:00	224 K
Apvxdwin.exe	2932	00	0:00:09	1,572 K
Avengine.exe	3072	00	0:00:00	6,704 K
cidaemon.exe	616	00	0:00:00	172 K
cisvc.exe	1228	00	0:01:22	136 K
cmd.exe	2820	00	0:00:00	48 K
cpustres.exe	2628	90	0:00:40	2,696 K
csrss.exe	464	00	0:04:04	3,660 K
ctfmon.exe	2376	00	0:00:02	2,068 K
explorer.exe	3940	00	0:09:16	23,628 K
lsass.exe	548	00	0:03:47	788 K
mdm.exe	1260	00	0:00:00	792 K

6. In the dialog box of *CpuStres*, try setting the thread's "activity level" to various values, and observe the effect on the CPU time used by the process. You could try enabling additional threads as well.
7. In Task Manager's Processes tab, Right-click on the CpuStres.Exe process and select "Set Priority."



8. In the selection box, click on "Realtime." This changes the priority class of the process, setting it above that of many critical threads in the operating system.
9. Note that the target system is now completely unresponsive to keyboard input.

Part 2: Analyzing the Problem with WinDbg

10. On the host system, begin a debugging session using WinDbg, as described in Exercise 1, steps 8 through 15 in this module.
11. Use the debugger commands you have learned to examine the state of the target system.
 - a. Can you identify the “problem” thread? Do you see things in the debugger’s display of the thread that are consistent with the problems you observe on the target system?
 - b. Can you identify the process to which the “problem” thread belongs? What program is the process running?
 - c. What other threads are trying to execute on the target system? What routines are they executing?
 - d. Is there anything you can do with the debugger to allow the target system to resume normal operation?
 - e. Assuming for the moment that the problem was not apparent – are there any debugger commands that might allow you to collect more information for further analysis, without continuing to tie up the system with a kernel debugging session?
 - f. What could you have done if the target system had not been booted with “debugger enabled”?

Exercise 3: Debugging a “System Hang” at High IRQL

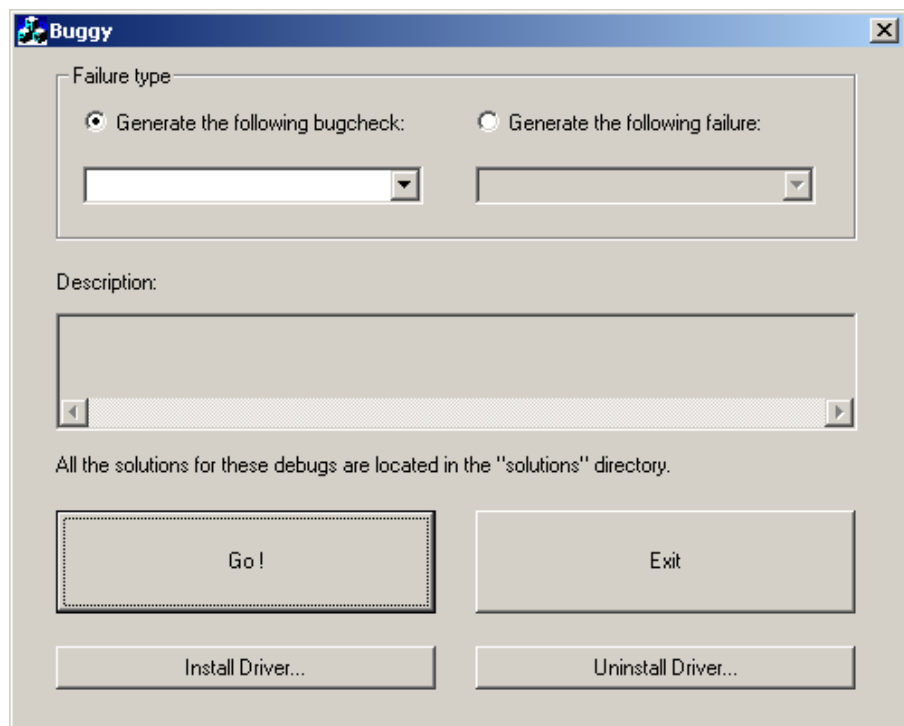
In this lab you will use a WinDbg kernel debugging session to determine the cause of a system hang, this one created by a driver spending excessive time at high IRQL (Interrupt Request Level). This will appear similar to the previous case, but the problem scenario is quite different.

This lab assumes that you have completed Exercise 1 in this module.

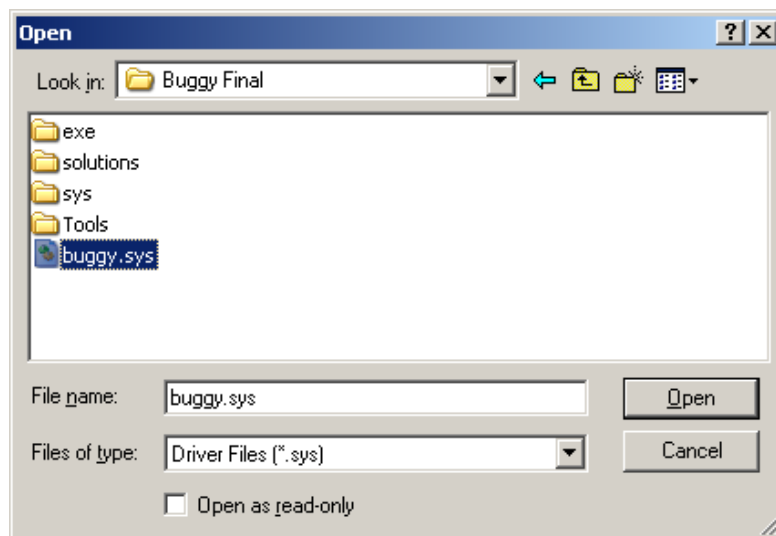
Part 1: Installing the Buggy application

In this section, you will install the *Buggy* application and driver. The *buggy.sys* driver contains examples of many common types of driver problems. The *buggy.exe* application allows you to load the *buggy.sys* driver and cause it to execute the code to exhibit these problems. In later labs, we'll interactively debug the code in *buggy.sys* in a live kernel debugging session using WinDbg; here, we'll just use WinDbg to identify the failing component.

1. Locate **Buggy.exe** on the CD in **labs\Buggy** and launch this application.



2. Click on the “**Install Driver...**” button.



3. Select **buggy.sys** and click **Open**.

Part 2: Creating the Problem Scenario

Once the driver is installed, the *Buggy* application offers the ability to generate various bugchecks and failures. Once the bugcheck or failure has been selected, clicking on the “Go !” button will immediately generate the problem. So, take note that your system will become unusable once this button has been pressed!

4. Using the “Failure type” radio buttons, select “**Generate the following failure:**”
5. Using the right-hand drop-down menu, select “Loop at high IRQL”
6. Click on the “Go!” button.
7. Observe that the target system is now completely unresponsive to mouse movement, keyboard input, etc.

Part 3: Analyzing the Problem with WinDbg

8. On the host system, begin a debugging session using WinDbg, as described in Exercise 1, steps 8 through 15 in this module.
9. Use the debugger commands you have learned to examine the state of the target system.
 - a. Can you identify the “problem” thread? Do you see things in the debugger’s display of the thread that are consistent with the problems you observe on the target system? How is it different from the “CpuStres” case?
 - b. Can you identify the process to which the “problem” thread belongs? What program is the process running?
 - c. What other threads are trying to execute on the target system? What routines are they executing? How is this different from the “CpuStres” case?
 - d. Is there anything you can do with the debugger to allow the target system to resume normal operation?

Module 7 Labs: Understanding Disassembled Code

Lab Objectives



Lab 7 – Understanding Disassembled Code

Exercise 1 – Mapping Assembly to C / C++

Exercise 2 – Reading Assembly Language (Part 1)

Exercise 3 – Reading Assembly Language (Part 2)

Estimated Time to Complete this Lab: 30 Minutes

Exercise 1: Mapping Assembly to C / C++

Match the C/C++ code to the corresponding Assembly code.

Given the three lettered boxes (A-C) on the left that contain C/C++ code, and the three numbered boxes (1-3) to the right that contain Assembly code, fill in the following blanks to correctly match a C/C++ box to the Assembly code box that contains the corresponding code.

A → _____

B → _____

C → _____

A	1
<pre>void main(void) { int x = 0; int y = 1; int z = 3; x = y + z; }</pre>	<pre>00401034 mov ecx,dword ptr [ebp-4] 00401037 cmp ecx,dword ptr [ebp-0Ch] 0040103A jne 0040103E 0040103C jmp 00401042 0040103E xor eax,eax 00401040 jmp 00401044 00401042 jmp 0040103E 00401044 mov esp,ebp 00401046 pop ebp 00401047 ret</pre>
B	2
<pre>if (x == z) goto End; End: return 0;</pre>	<pre>0040AE4E mov edx,1 0040AE53 test edx,edx 0040AE55 je 0040AE6C 0040AE57 mov eax,dword ptr [ebp-4] 0040AE5A shl eax,1 0040AE5C mov dword ptr [ebp-4],eax 0040AE5F cmp dword ptr [ebp-4],4E20h 0040AE66 jnl 0040AE6A 0040AE68 jmp 0040AE6C 0040AE6A jmp 0040AE4E 0040AE6C xor eax,eax 0040AE6E jmp 0040AE72 0040AE70 jmp 0040AE6C 0040AE72 mov esp,ebp 0040AE74 pop ebp 0040AE75 ret</pre>
C	3
<pre>while (true) { x*=2; if (x >= 20000) break; } return 0;</pre>	<pre>00401010 push ebp 00401011 mov ebp,esp 00401013 sub esp,0Ch 00401016 mov dword ptr [ebp-4],0 0040101D mov dword ptr [ebp-8],1 00401024 mov dword ptr [ebp-0Ch],3 0040102B mov eax,dword ptr [ebp-8] 0040102E add eax,dword ptr [ebp-0Ch] 00401031 mov dword ptr [ebp-4],eax 00401034 mov esp,ebp 00401036 pop ebp 00401037 ret</pre>

Exercise 2: Reading Assembly Language (Part 1)

Determine what the Assembly code is doing and then answer the questions below.

Note:

The CDQ instruction is a way to convert a DWORD to a QUAD word and preserve the sign. The result of this conversion is stored in EDX:EAX. In the example below, CDQ is acting on EAX (which has a value of 0xA), and when widened EDX will contain 0x0 and EAX will contain 0xA. (So, you should basically ignore the CDQ for this example).

```
0040AE20  push      ebp
0040AE21  mov       ebp,esp
0040AE23  sub       esp,10h
0040AE26  mov       dword ptr [ebp-4],0Ah
0040AE2D  mov       dword ptr [ebp-0Ch],3
0040AE34  mov       eax,dword ptr [ebp-4]
0040AE37  cdq       (Ignore me... see above for why)
0040AE38  idiv      eax,dword ptr [ebp-0Ch]
0040AE3B  mov       dword ptr [ebp-8],eax
0040AE3E  mov       eax,dword ptr [ebp-0Ch]
0040AE41  imul      eax,dword ptr [ebp-8]
0040AE45  mov       ecx,dword ptr [ebp-4]
0040AE48  sub       ecx,eax
0040AE4A  mov       dword ptr [ebp-10h],ecx
0040AE4D  mov       esp,ebp
0040AE4F  pop       ebp
0040AE50  ret
```

What is the Value stored in ebp-4? _____

What is the Value stored in ebp-8? _____

What is the Value stored in ebp-10? _____

What is the Value stored in ebp-0xc _____

What is this program doing?

Exercise 3: Reading Assembly Language (Part 2)

Determine what the Assembly code is doing and then answer the questions below.

Note:

JGE is Jump if Greater or Equal.

```
0040AE20  push      ebp
0040AE21  mov       ebp,esp
0040AE23  sub       esp,8
0040AE26  mov       dword ptr [ebp-4],5
0040AE2D  mov       dword ptr [ebp-8],1
0040AE34  jmp       0040AE3F
0040AE36  mov       eax,dword ptr [ebp-8]
0040AE39  add       eax,1
0040AE3C  mov       dword ptr [ebp-8],eax
0040AE3F  cmp       dword ptr [ebp-8],6
0040AE43  jge       0040AE51
0040AE45  mov       ecx,dword ptr [ebp-4]
0040AE48  imul      ecx,dword ptr [ebp-8]
0040AE4C  mov       dword ptr [ebp-4],ecx
0040AE4F  jmp       0040AE36
0040AE51  mov       esp,ebp
0040AE53  pop       ebp
0040AE54  ret
```

1. What is the Value stored in ebp-4? _____
2. What is the Value stored in ebp-8? _____
3. What is this program doing?

Module 8 Labs: Call Stacks

Lab Objectives



Lab 8 – Call Stacks

Exercise 1 – Reading a Call Stack

Exercise 2 – Identifying Calling Conventions

Estimated Time to Complete this Lab: 30 Minutes

Exercise 1: Reading a Call Stack

Given the following call stack, determine the return address, the locals, and the two variables passed to `devidem!Devidem()`.

```
ChildEBP
0012ff64 devidem!Devidem+0x18
0012ff80 devidem!main+0x28
0012ffc0 devidem!mainCRTStartup+0xb4
0012fff0 KERNEL32!BaseProcessStart+0x3d

0:000> dd esp-10
0012ff50 00402f44 002f0000 40000069 00000800
0012ff60 00000002 0012ff80 00401028 00000006
0012ff70 00000003 00000000 00000003 00000006
0012ff80 0012ffc0 00401144 00000001 002f0d70
0012ff90 002f0db8 00000000 00000000 7ffdf000
0012ffa0 80100000 be4a7d00 0012ff94 00000000
0012ffb0 0012ffe0 00402764 00407118 00000000
0012ffc0 0012fff0 77e87903 00000000 00000000
```

_____ Return address pushed on stack during call to `devidem!Devidem()`

_____ Return address pushed on stack during call to `devidem!main()`

_____ Value of the first argument passed to `devidem!Devidem()`

_____ Value of the second argument passed to `devidem!Devidem()`

_____ Value of the local variable for `devidem!Devidem()`

Exercise 2: Identifying Calling Conventions

Determine what calling conventions are being used below.

_____ CDECL

_____ FASTCALL

_____ STDCALL

A	<pre> 0040103e push ebp 0040103f mov ebp,esp 00401041 sub esp,0xc 00401044 mov [ebp-0xc],edx 00401047 mov [ebp-0x8],ecx 0040104a mov dword ptr [ebp-0x4],0x0 00401051 mov eax,[ebp-0x8] 00401054 add eax,[ebp-0xc] 00401057 mov [ebp-0x4],eax 0040105a mov eax,[ebp-0x4] 0040105d mov esp,ebp 0040105f pop ebp 00401060 ret </pre>
B	<pre> 00401040 push ebp 00401041 mov ebp,esp 00401043 push ecx 00401044 mov dword ptr [ebp-0x4],0x0 0040104b mov eax,[ebp+0x8] 0040104e add eax,[ebp+0xc] 00401051 mov [ebp-0x4],eax 00401054 mov eax,[ebp-0x4] 00401057 mov esp,ebp 00401059 pop ebp 0040105a ret 0x8 </pre>
C	<pre> 00401043 push ebp 00401044 mov ebp,esp 00401046 push ecx 00401047 mov dword ptr [ebp-0x4],0x0 0040104e mov eax,[ebp+0x8] 00401051 add eax,[ebp+0xc] 00401054 mov [ebp-0x4],eax 00401057 mov eax,[ebp-0x4] 0040105a mov esp,ebp 0040105c pop ebp 0040105d ret </pre>

Module 9 Labs: Crash Dump Analysis II

Lab Objectives



Lab 9 – Crash Dump Analysis II

Exercise 1 – Using WinDbg to Analyze a Kernel-Mode Dump File (Part 3)

Estimated Time to Complete this Lab: 90 minutes

Exercise 1: Using WinDbg to Analyze the Kernel-Mode Dump (Part 3)

In this exercise, you will use WinDbg to load a couple of sample memory.dmp files for analysis. The files are located in the `c:\dumps` directory.

Continuing the analysis of the *memory.dmp* file

1. Load *memory.dmp* by repeating steps 1, 2, and 3 from Lab 3.

```

Dump C:\Documents\DebugFest Taiwan\cdrom\labs\dumps\dumps\MEMORY1.DMP - WinDbg:6.0.0012.0
File Edit View Debug Window Help
Symbol search path is: c:\win2ksym
Microsoft (R) Windows Debugger Version 6.0.0012.0
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\Documents\DebugFest Taiwan\cdrom\labs\dumps\dumps\MEMORY1.DMP]
Kernel Summary Dump File: Only kernel address space is available

WARNING: C:\cases\FS memory corruption\Symbols-92 is not accessible, ignoring
Symbol search path is: c:\win2ksym;SRV*\\symbols\symbols
Executable search path is:
Windows 2000 Kernel Version 2195 UP Free x86 compatible
Product: Server
Kernel base = 0x80400000 PsLoadedModuleList = 0x8046a4c0
Debug session time: Wed Aug 16 16:05:41 2000
System Uptime: 0 days 0:17:15.291
Loading Kernel Symbols
Loading unloaded module list
No unloaded module list present
Loading User Symbols

*****
*                               *
*               Bugcheck Analysis               *
*                               *
*****

Use !analyze -v to get detailed debugging information.

BugCheck 1E, {c0000005, 804676c8, 1, 0}

Probably caused by : ntoskrnl.exe ( nt!ExFreePoolWithTag+342 )
Ln 176, Col 1 Sys 0:C:\Docu Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

2. The first thing we need to look at when looking at a memory dump file is the crash dump analysis performed by Windbg itself. Type “`!analyze -v`” to see the interpretation made by the debugger:

```

kd> !analyze -v
*****
*                               *
*               Bugcheck Analysis               *
*                               *
*****

KMODE_EXCEPTION_NOT_HANDLED (1e)
This is a very common bugcheck. Usually the exception address pinpoints
the driver/function that caused the problem. Always note this address
as well as the link date of the driver/image that contains this address.
Some common problems are exception code 0x80000003. This means a hard
coded breakpoint or assertion was hit, but this system was booted
/NODEBUG. This is not supposed to happen as developers should never have
hardcoded breakpoints in retail code, but ...
If this happens, make sure a debugger gets connected, and the
system is booted /DEBUG. This will let us see why this breakpoint is
happening.
An exception code of 0x80000002 (STATUS_DATATYPE_MISALIGNMENT) indicates
that an unaligned data reference was encountered. The trap frame will
supply additional information.
Arguments:
Arg1: c0000005, The exception code that was not handled
Arg2: 804676c8, The address that the exception occurred at
Arg3: 00000001, Parameter 0 of the exception
Arg4: 00000000, Parameter 1 of the exception

```

Debugging Details:

EXCEPTION_CODE: c0000005

FAULTING_IP:

nt!ExFreePoolWithTag+342

804676c8 890a mov [edx],ecx

EXCEPTION_PARAMETER1: 00000001

EXCEPTION_PARAMETER2: 00000000

WRITE_ADDRESS: 00000000

DEFAULT_BUCKET_ID: DRIVER_FAULT

BUGCHECK_STR: 0x1E_W

TRAP_FRAME: f71efb38 -- (.trap ffffffff71efb38)

ErrCode = 00000002

eax=e1e40fc0 ebx=00000000 ecx=00000000 edx=00000000 esi=e1e40ea0 edi=81438428

eip=804676c8 esp=f71efbac ebp=f71efbd0 iopl=0 nv up ei pl zr na po nc

cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010246

nt!ExFreePoolWithTag+342:

804676c8 890a mov [edx],ecx ds:0023:00000000=????????

Resetting default context

LAST_CONTROL_TRANSFER: from 804672a2 to 804676c8

STACK_TEXT:

```

f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 80495b70 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall2+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall2+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
0054fddc 77d4fa80 0054fdfc 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStub+0x5e
0054fe44 77d4f9d7 00000000 000fc328 001112f0 RPCRT4!OSF_SCALL::DispatchHelper+0xa4
0054fe58 77d4f779 00000000 00000000 00000001 RPCRT4!OSF_SCALL::DispatchRPCCall+0x115
0054fe90 77d4f4d8 000fc310 00000003 00000000 RPCRT4!OSF_SCALL::ProcessReceivedPDU+0x43
0054feb0 77d4f0bc 000fc310 0000002c 00015f90 RPCRT4!OSF_SCALL::BeginRpcCall+0xd0
0054ff10 77d4f033 00000000 000fc310 0000002c
PCRT4!OSF_SCONNECTION::ProcessReceiveComplete+0x235
0054ff20 77d5bb62 00088860 0000000c 00000000 RPCRT4!ProcessConnectionServerReceivedEvent+0x1b
0054ff74 77d5ba15 77d4b7bf 00088860 00070178 RPCRT4!LOADABLE_TRANSPORT::ProcessIOEvents+0x9d
0054ff78 77d4b7bf 00088860 00070178 00070640 RPCRT4!ProcessIOEventsWrapper+0x9
0054ffa8 77d4b771 0008b120 0054ffec 77e92ca8 RPCRT4!BaseCachedThreadRoutine+0x4f
0054ffb4 77e92ca8 0008d2c0 00070178 00070640 RPCRT4!ThreadStartRoutine+0x18
0054ffec 00000000 77d4b759 0008d2c0 00000000 KERNEL32!BaseThreadStart+0x52

```

FOLLOWUP_IP:

nt!ExFreePoolWithTag+342

804676c8 890a mov [edx],ecx

FOLLOWUP_NAME: Pool_Corruption

SYMBOL_NAME: nt!ExFreePoolWithTag+342

MODULE_NAME: nt

```

IMAGE_NAME:  ntoskrnl.exe

DEBUG_FLR_IMAGE_TIMESTAMP:  384d9b17

STACK_COMMAND:  .trap ffffffff71efb38 ; kb

BUCKET_ID:  0x1E_W_nt!ExFreePoolWithTag+342

Followup: Pool_Corruption
-----

```

3. This bug check is “Stop 0x1E” (Unhandled Exception). This tells us the system encountered an exception and none of the exception handlers handled it. So it gets passed on up to the debugger if one is available; if not, like in this case, it crashes the system with a Stop 0x1E.
4. Next let's take a look at the stack trace. Type “**kb** <enter>”.

```

DEBUG_FLR_IMAGE_TIMESTAMP:  384d9b17

STACK_COMMAND:  .trap ffffffff71efb38 ; kb

BUCKET_ID:  0x1E_W_nt!ExFreePoolWithTag+342

Followup: Pool_Corruption
-----

kd> kb
ChildEBP RetAddr  Args to Child
f71efac8 804624cb f71efae4 00000000 f71efb38 nt!KiDispatchException+0x30e
f71efb30 8046247d 81288784 f71efb84 804b0b48 nt!CommonDispatchException+0x4d
f71efb30 804676c8 81288784 f71efb84 804b0b48 nt!KiUnexpectedInterruptTail+0x1f4
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 80495b70 nt!ExFreePool+0xb
f71efbe8 80495b70 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 77f8c7c7 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa68 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RcCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall12+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall12+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100

```

5. Next we need to find the trap frame. The trap frame will actually show us the state of the CPU when the exception occurred. The current system context is the exception handling code. To do this we type “**.trap <address> <enter>**”, where **<address>** is the address of the trap frame. From the debugger documentation's description of bugcheck 0x1E, we know that the third parameter of KiDispatchException is the trap frame. The third parameter passed to KiDispatchException in the above stack trace is 0xF71EFB38, so that is the address we will supply to the **.trap** command:

```

Dump C:\Documents\DebugFest Taiwan\cdrom\labs\dumps\dumps\MEMORY1.DMP - WinDbg5.0.0012.0 - [Command]
File Edit View Debug Window Help

Followup: Pool_Corruption

kd> kb
ChildEBP RetAddr  Args to Child
f71efac8 804624cb f71efae4 00000000 f71efb38 nt!KiDispatchException+0x30e
f71efb30 8046247d 81288784 f71efb84 804b0b48 nt!CommonDispatchException+0x4d
f71efb30 804676c8 81288784 f71efb84 804b0b48 nt!KiUnexpectedInterruptTail+0x1f4
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 8049b570 nt!ExFreePool+0xb
f71efbe8 8049b570 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RcCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall12+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall12+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
kd> .trap f71efb38
ErrCode = 00000002
eax=e1e40fc0 ebx=00000000 ecx=00000000 edx=00000000 esi=e1e40ea0 edi=81438428
eip=804676c8 esp=f71efbac ebp=f71efbd0 iopl=0         nv up ei pl zr na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010246
nt!ExFreePoolWithTag+342:
804676c8 890a          mov     [edx],ecx          ds:0023:00000000=????????

```

6. The screen shot above shows us the result of supplying the trap frame address to the debugger. The values represent the registers and the last instruction processed at the point the exception occurred. We can see that we are trying to move the contents of ecx to the memory location referenced in edx. We can also see that value of edx is 0x00000000 or NULL, which is invalid. This caused the exception. So we need to determine how edx became zero!
7. Type **"kb <enter>"** to take a look at the stack trace. Notice that the same **kb** command now displays the stack trace in the context of the trap frame, because we have issued the **.trap <address>** command. (If you want to return back to the previous context, just issue a **.thread** command.)

```

Dump C:\Documents\DebugFest Taiwan\cdrom\labs\dumps\dumps\MEMORY1.DMP - WinDbg5.0.0012.0 - [Command]
File Edit View Debug Window Help

0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
kd> .trap f71efb38
ErrCode = 00000002
eax=e1e40fc0 ebx=00000000 ecx=00000000 edx=00000000 esi=e1e40ea0 edi=81438428
eip=804676c8 esp=f71efbac ebp=f71efbd0 iopl=0         nv up ei pl zr na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010246
nt!ExFreePoolWithTag+342:
804676c8 890a          mov     [edx],ecx          ds:0023:00000000=????????
kd> kb
*** Stack trace for last set context - .thread/.cxr resets it
ChildEBP RetAddr  Args to Child
f71efbd0 804672a2 00000001 00000000 8049b588 nt!ExFreePoolWithTag+0x342
f71efbdc 8049b588 e1e40ea8 e1bbb80c 8049b570 nt!ExFreePool+0xb
f71efbe8 8049b570 e1bbb810 e1bbb7f8 81437740 nt!SepTokenDeleteMethod+0x1b
f71efc04 8044c3b3 e1bbb810 e1bbb810 81272020 nt!ObpRemoveObjectRoutine+0xd6
f71efc28 80451ac4 00000002 81272020 f71efd3c nt!ObfDereferenceObject+0x149
f71efc40 80451e5d 81272020 00000000 00000000 nt!PsImpersonateClient+0x191
f71efc78 804983da 81272020 00000000 f71efd64 nt!PsAssignImpersonationToken+0x133
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!NtSetInformationThread+0x3a9
f71efd4c 80461691 ffffffff 00000005 0054fa7c nt!KiSystemService+0xc4
0054fa64 010074c2 ffffffff 00000005 0054fa7c ntdll!NtSetInformationThread+0xb
0054fa80 0100687b 0054fa9c 0054fac0 00160014 services!ScReleasePrivilege+0x18
0054fa98 77d45178 00000015 02020202 00000001 services!RcCloseServiceHandle+0x81
0054fab0 77da1586 010067e0 0054fac4 00000001 RPCRT4!Invoke+0x30
0054fd10 77da1937 00000000 00000000 0054fdfc RPCRT4!NdrStubCall12+0x63d
0054fd2c 77d453e2 0054fdfc 0008a288 0054fdfc RPCRT4!NdrServerCall12+0x17
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!DispatchToStubInC+0x84
0054fdb0 77d45215 00000000 00000000 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x100
0054fd64 77d452ef 010065fd 0054fdfc 0054fe40 RPCRT4!RPC_INTERFACE::DispatchToStub+0x5e
0054fd44 77d4f9d7 00000000 000fc328 001112f0 RPCRT4!OSF_SCALL::DispatchHelper+0xa4
0054fe58 77d4f779 00000000 00000000 00000001 RPCRT4!OSF_SCALL::DispatchRPCCall+0x115

```

8. Note that WinDbg's **!analyze -v** command actually performed much of this analysis. The command finds the trap frame address, issued the **.trap** command, and displayed the registers, faulting instruction, and stack with that **.trap** command in effect. It also told us how to recall that context at any time:

```
STACK_COMMAND: .trap ffffffff71efb38 ; kb
```

9. We can see in this case we are trying to free some memory, as we are in a routine called `ExFreePoolWithTag`. We might infer from the arguments to this routine that the address we are trying to free is `0x00000001`, which would be invalid.
10. We can also see that this routine was in turn called from `ExFreePool`, and the first argument (the address of pool to be freed) was valid in that routine. So it appears that we either lost the correct value during a context switch or someone else wrote over some of our memory.
11. Examine this crash dump in more detail, and try to answer the following questions:
 - a. Is the preceding analysis correct?
 - b. What debugger commands have we seen that are associated with the memory pools?
 - c. What do those commands tell you about the address passed as the first argument to `ExFreePool`?
 - d. What is the code in `ExFreePoolWithTag` doing that caused it to try to reference address 0?
12. Load up the additional *memory.dmp* files and see if you can isolate the point of failure.

Hint: Start out by using **!analyze -v**. Refer to the debugger help file for information on each particular stop message and parameters. Use the DDK (Driver Developer's Kit) documentation to aid in the understanding of many of the system routines you find on the stack.

Module 10 Labs: Kernel Debugging II

Lab Objectives



Lab 10 – Kernel Debugging II

Exercise 1 – Using Some Basic Debugger Commands in Kernel Mode

Exercise 2 – Live Debugging Using “Buggy.sys”

Exercise 3 – Finding a Resource Deadlock

Exercise 4 – Using PoolMon to find a Kernel-Mode Memory Leak

Exercise 5 – Remote Debugging with WinDbg

Estimated Time to Complete this Lab: 90 minutes

Exercise 1: Using Some Basic Debugger Commands in Kernel Mode

In this exercise we will experiment with some of the more advanced debugger commands we've learned in a live kernel debugging scenario.

1. Ensure that your target system was booted with debugging enabled.
2. Select **Start | Programs | Microsoft Debugging Tools | WinDbg**.
3. Select **File | Kernel Debugging**.
4. Enter *19200* for the baud rate and *COM1* for the port.
5. Press CTRL+Break to break into the target.
6. Type **“.reload <enter>”** to reload the symbols.

```

Windows Debugger:5.1.2250.3 - Kernel 'com:port=com1,baud=19200'
File Edit View Debug Window Help
[Icons]
Command
Executable search path is: c:\winnt\system32
Windows 2000 Kernel Version 2195 UP Free x86 compatible
Kernel base = 0x80400000 PsLoadedModuleList = 0x8046a4c0
Break instruction exception - code 80000003 (first chance)
*****
*
*   You are seeing this message because you pressed either
*       CTRL+C (if you run kd.exe) or,
*       CTRL+BREAK (if you run WinDBG),
*   on your debugger machine's keyboard.
*
*               THIS IS NOT A BUG OR A SYSTEM CRASH
*
*   If you did not intend to break into the debugger, press the "g" key, then
*   press the "Enter" key now. This message might immediately reappear. If it
*   does, press "g" and "Enter" again.
*
*****
ntoskrnl!RtlpBreakWithStatusInstruction:
80452e70 cc          int     3
kd> .reload
Connected to Windows 2000 2195 x86 compatible target, ptr64 FALSE
Loading Kernel Symbols
.....Unable to read image header for fdc.sys at f
Loading User Symbols
PFEB is NULL
kd>
Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

7. Now let's take a look around. Type **“!process”** to view the active process at the time we broke in:

```

Windows Debugger:5.1.2250.3 - Kernel 'comport=com1,baud=19200'
File Edit View Debug Window Help
Command
Loading User Symbols
PPEB is NULL
kd> !process
PROCESS 8046bb60 SessionId: 0 Cid: 0000 Peb: 00000000 ParentCid: 0000
DirBase: 00030000 ObjectTable: 81437b88 TableSize: 148
Image: Idle
VadRoot 0 Clone 0 Private 0 Modified 0 Locked 0
DeviceMap 0
Process Lock Owned by Thread 0
Token e10007f0
ElapsedTime 23:51:40.0602
UserTime 0:00:00.0000
KernelTime 4:48:07.0478
QuotaPoolUsage[PagedPool] 0
QuotaPoolUsage[NonPagedPool] 0
Working Set Sizes (now,min,max) (4, 50, 450) (16KB, 200KB, 1800KB)
PeakWorkingSetSize 4
VirtualSize 0 Mb
PeakVirtualSize 0 Mb
PeakFaultCount 1
MemoryPriority BACKGROUND
BasePriority 0
CommitCharge 0

THREAD 8046bdf0 Cid 0.0 Teb: 00000000 Win32Thread: 00000000 RUNNING
kd>
Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

Note: The Idle Process is running on the computer used in these screenshots. Your experience may be different.

8. Now type “**!process 0 0**”. This will give you a list of all of the processes on the system.
9. Find the Client ID (“CID”) for the explorer process on your target computer.
10. Now type “**!process <CID> 7** <enter>”. This will dump out the EPROCESS structure for the explorer process along with all of the linked ETHREAD structures.

```

Windows Debugger:5.1.2250.3 - Kernel 'comport=com1,baud=19200'
File Edit View Debug Window Help
Command
f5a/4938 f5a/497c 810e908c 810e9020 f5a/49dc +0x//f8/e//
07e4fea8 00000000 00000000 00000000 00000000 +0xf5a7497c

THREAD 811a0400 Cid 4a8.1f0 Teb: 7ffa8000 Win32Thread: e1d23a88 WAIT: (WrUserReq
811f0160 SynchronizationEvent
Not impersonating
Owning Process 8119f020
WaitTime (seconds) 1781317
Context Switch Count 729 LargeStack
UserTime 0:00:00.0170
KernelTime 0:00:00.0150
Start Address 0x77e92c50
Win32 Start Address 0x76e32ab8
Stack Init f5fb4000 Current f5fb3cc8 Base f5fb4000 Limit f5fb1000 Call 0
Priority 10 BasePriority 8 PriorityDecrement 0 DecrementCount 0
Kernel stack not resident.

ChildEBP RetAddr Args to Child
f5fb3ce0 8042d61c 00000000 e1d23a88 00000001 ntoskrnl!KiSwapThread+0xc5
f5fb3d08 a00159cb 811f0160 0000000d 00000001 ntoskrnl!KeWaitForSingleObject+0x1a1
f5fb3d44 a0014f6c 000020ff 00000000 00000001 win32k!xxxSleepThread+0x183
f5fb3d54 a0014f53 0804feec 80461691 00000000 win32k!xxxWaitMessage+0xe
f5fb3d5c 80461691 00000000 00000000 00000000 win32k!NtUserWaitMessage+0xb
f5fb3d5c 77e14b53 00000000 00000000 00000000 ntoskrnl!KiSystemService+0xc4
f5fb3cd0 f5fb3d08 811a046c 811a0400 811f0160 +0x77e14b53
0804ff2c 00000000 00000000 00000000 00000000 +0xf5fb3d08
kd>
Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

11. Find a thread with an IRP List. Dump out that thread by typing “**!thread <address>** <enter>”.


```

Windows Debugger:5.1.2250.3 - Kernel 'com:port=com1,baud=19200'
File Edit View Debug Window Help
[Icons]
Command
kd> !thread 810de560
THREAD 810de560 Cid 4a8.548 Teb: 7ffad000 Win32Thread: 00000000 WAIT: (WrEventPairLow) U:
81162220 Unknown
810de648 NotificationTimer
IRP List:
8114c4a8: (0006.00b8) Flags: 00000800 Mdl: 00000000
Not impersonating
Owning Process 8119f020
WaitTime (seconds) 1757051
Context Switch Count 61
UserTime 0:00:00.0000
KernelTime 0:00:00.0000
Start Address 0x77e92c50
Win32 Start Address 0x77d4b759
Stack Init f5e67000 Current f5e66c90 Base f5e67000 Limit f5e64000 Call 0
Priority 8 BasePriority 8 PriorityDecrement 0 DecrementCount 0
Kernel stack not resident.

ChildEBP RetAddr Args to Child
f5e66ca8 8042e949 f5e66d64 00000000 804b219f ntoskrnl!KiSwapThread+0xc5
f5e66ccc 804b2288 f5e66d00 00000001 f5e66d00 ntoskrnl!KeRemoveQueue+0x195
f5e66d48 80461691 000000fc 0126ff0c 0126fefe ntoskrnl!NtRemoveIoCompletion+0xf1
f5e66d48 77f8b520 000000fc 0126ff0c 0126fefe ntoskrnl!KiSystemService+0xc4
f5e66c98 f5e66ccc 810de5e4 810de560 81162220 +0x77f8b520
0126fee4 00000000 00000000 00000000 00000000 +0xf5e66ccc

kd>
Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

12. Next let's take a look at the IRP (I/O Request Packet). Type "**!irp <address>**" <enter>."

```

Windows Debugger:5.1.2250.3 - Kernel 'com:port=com1,baud=19200'
File Edit View Debug Window Help
[Icons]
Command
Context Switch Count 61
UserTime 0:00:00.0000
KernelTime 0:00:00.0000
Start Address 0x77e92c50
Win32 Start Address 0x77d4b759
Stack Init f5e67000 Current f5e66c90 Base f5e67000 Limit f5e64000 Call 0
Priority 8 BasePriority 8 PriorityDecrement 0 DecrementCount 0
Kernel stack not resident.

ChildEBP RetAddr Args to Child
f5e66ca8 8042e949 f5e66d64 00000000 804b219f ntoskrnl!KiSwapThread+0xc5
f5e66ccc 804b2288 f5e66d00 00000001 f5e66d00 ntoskrnl!KeRemoveQueue+0x195
f5e66d48 80461691 000000fc 0126ff0c 0126fefe ntoskrnl!NtRemoveIoCompletion+0xf1
f5e66d48 77f8b520 000000fc 0126ff0c 0126fefe ntoskrnl!KiSystemService+0xc4
f5e66c98 f5e66ccc 810de5e4 810de560 81162220 +0x77f8b520
0126fee4 00000000 00000000 00000000 00000000 +0xf5e66ccc

kd> !irp 8114c4a8
Irp is active with 2 stacks 2 is current (= 0x8114c53c)
No Mdl Thread 810de560: Irp stack trace.
cmd flg cl Device File Completion-Context
[ 0, 0] 0 0 00000000 00000000 00000000-00000000

Args: 00000000 00000000 00000000 00000000
>[ d, 0] 0 1 8131d430 810d4a8 00000000-00000000 pending
\FileSystem\Npfs
Args: 00000000 00000000 00110008 00000000

kd>
Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```

13. Next, we will look at the device objects associated with this IRP. Type "**!devobj <address>**" <enter>" to look at the device object.

Windows Debugger:5.1.2250.3 - Kernel 'com:port=com1,baud=19200'

File Edit View Debug Window Help

Command

Kernel stack not resident.

```
ChildEBP RetAddr  Args to Child
f5e66ca8 8042e949 f5e66d64 00000000 804b219f ntoskrnl!KiSwapThread+0xc5
f5e66ccc 804b2288 f5e66d00 00000001 f5e66d00 ntoskrnl!KeRemoveQueue+0x195
f5e66d48 80461691 000000fc 0126ff0c 0126fefe ntoskrnl!NtRemoveIoCompletion+0xf1
f5e66d48 77f8b520 000000fc 0126ff0c 0126fefe ntoskrnl!KiSystemService+0xc4
f5e66c98 f5e66ccc 810de5e4 810de560 81162220 +0x77f8b520
0126fee4 00000000 00000000 00000000 00000000 +0xf5e66ccc
```

kd> !irp 8114c4a8

Irp is active with 2 stacks 2 is current (= 0x8114c53c)

No Mdl Thread 810de560: Irp stack trace.

cmd	flg	cl	Device	File	Completion-Context
[0, 0]	0	0	00000000	00000000	00000000-00000000

Args: 00000000 00000000 00000000 00000000

>[d, 0] 0 1 8131d430 810dfda8 00000000-00000000 pending

\FileSystem\Npfs

Args: 00000000 00000000 00110008 00000000

kd> !devobj 8131d430

Device object (8131d430) is for:

NamedPipe \FileSystem\Npfs DriverObject 8131d770

Current Irp 00000000 RefCount 159 Type 00000011 Flags 00000240

DevExt 8131d4e8 DevObjExt 8131d590

ExtensionFlags (0000000000)

Device queue is not busy.

kd>

Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

14. Type “!vm <enter>” to take a look at the virtual memory usage.

Windows Debugger:5.1.2250.3 - Kernel 'com:port=com1,baud=19200'

File Edit View Debug Window Help

Command

Process	Driver Commit	Committed pages	Commit limit
00e4 services.exe	793 (3172 Kb)	19733 (78932 Kb)	77790 (311160 Kb)
00c8 winlogon.exe			
04a8 explorer.exe			
0390 inetinfo.exe			
00f0 lsass.exe			
02ac llssrv.exe			
031c winmgmt.exe			
01c0 SPOOLSV.EXE			
0464 SMSAPM32.exe			
029c svchost.exe			
0200 msdtc.exe			
00b4 csrss.exe			
049c launch32.exe			
0098 smss.exe			
0204 SMSMon32.exe			
05cc mslexec.exe			
01a4 svchost.exe			
0578 svchost.exe			
0148 mda.exe			
02f8 mstask.exe			
0374 dfssvc.exe			
02e0 regsvc.exe			
0008 System			

kd>

Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

15. Type “!drivers <enter>” to get a list of loaded drivers.

Windows Debugger:5.1.2250.3 - Kernel 'comport=com1,baud=19200'

File Edit View Debug Window Help

Command

f75d0000	2a0 (0 kb)	480 (1 kb)	Null.SYS	Sat Sep 25 11:34:58 1999
f75d1000	720 (1 kb)	540 (1 kb)	Beep.SYS	Wed Oct 20 15:18:59 1999
f74d4000	2800 (10 kb)	aa0 (2 kb)	vga.sys	Sat Sep 25 11:37:40 1999
f75d2000	4a0 (1 kb)	800 (2 kb)	mnndd.SYS	Sat Sep 25 11:37:40 1999
f73a0000	4140 (16 kb)	e20 (3 kb)	Msfs.SYS	Tue Oct 26 16:21:32 1999
f7100000	7940 (30 kb)	1380 (4 kb)	Npfs.SYS	Sat Oct 09 16:58:07 1999
f7514000	1380 (4 kb)	7e0 (1 kb)	rasacd.sys	Sat Sep 25 11:41:23 1999
f6eae000	43500 (269 kb)	7020 (28 kb)	tcpip.sys	Mon Nov 29 23:38:42 1999
f7110000	7400 (29 kb)	1000 (4 kb)	msgpc.sys	Mon Nov 29 23:37:21 1999
f72b8000	63e0 (24 kb)	12a0 (4 kb)	wanarp.sys	Sat Oct 30 15:36:06 1999
f6e89000	21980 (134 kb)	2840 (10 kb)	netbt.sys	Mon Nov 29 23:37:39 1999
f7120000	6f20 (27 kb)	fa0 (3 kb)	netbios.sys	Tue Oct 12 12:34:19 1999
f6e67000	1de80 (119 kb)	3220 (12 kb)	rdbss.sys	Tue Nov 30 00:52:29 1999
f6dcf000	52ca0 (331 kb)	a180 (40 kb)	mrxsmbs.sys	Tue Nov 30 00:52:10 1999
f75d3000	740 (1 kb)	560 (1 kb)	dump_WMILIB.SYS	Sat Sep 25 11:36:47 1999
f6dba000	11b80 (70 kb)	2c00 (11 kb)	dump_atapi.sys	Sat Dec 04 12:19:32 1999
a0000000	177d60 (1503 kb)	2d5a0 (181 kb)	win32k.sys	Tue Nov 30 00:51:03 1999
f6d6f000	1f360 (124 kb)	2c00 (11 kb)	atiraged.dll	Tue Nov 30 01:31:17 1999
f643d000	170c0 (92 kb)	25a0 (9 kb)	afd.sys	Mon Nov 29 23:12:04 1999
f758c000	d40 (3 kb)	860 (2 kb)	ParVdm.SYS	Mon Sep 27 20:28:16 1999
f633a000	fc40 (63 kb)	2120 (8 kb)	wdmaud.sys	Wed Oct 27 11:40:45 1999
f64ff000	9520 (37 kb)	1f40 (7 kb)	sysaudio.sys	Mon Oct 25 12:28:14 1999
f60aa000	35520 (213 kb)	59e0 (22 kb)	srv.sys	Mon Nov 29 23:38:21 1999
f619e000	d820 (54 kb)	1280 (4 kb)	Cdfs.SYS	Mon Oct 25 12:23:52 1999
f605d000	21360 (132 kb)	2a00 (10 kb)	Fastfat.SYS	Mon Oct 25 12:20:50 1999
f60fe000	2160 (8 kb)	ac0 (2 kb)	spud.sys	Fri Nov 19 15:36:27 1999
f5f1f000	11f20 (71 kb)	2ac0 (10 kb)	ipsec.sys	Mon Nov 29 23:08:54 1999
TOTAL:	6d9880 (7014 kb)	12b420 (1197 kb)	(0 kb)	(0 kb)

kd>

Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

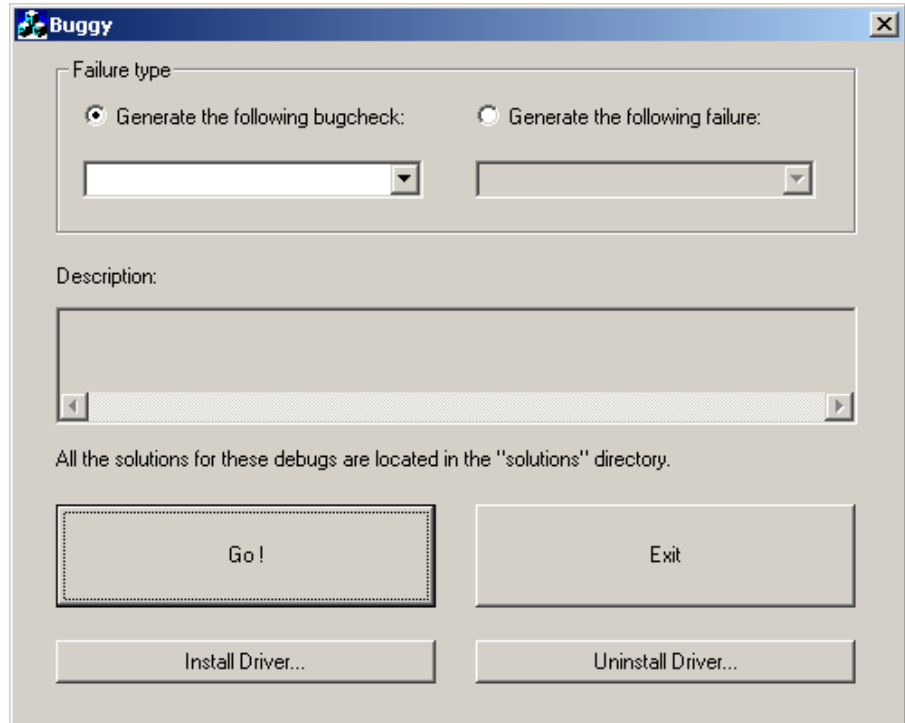
Exercise 2: Live Debugging using “Buggy.Sys”

Part 1: Installing the Buggy application

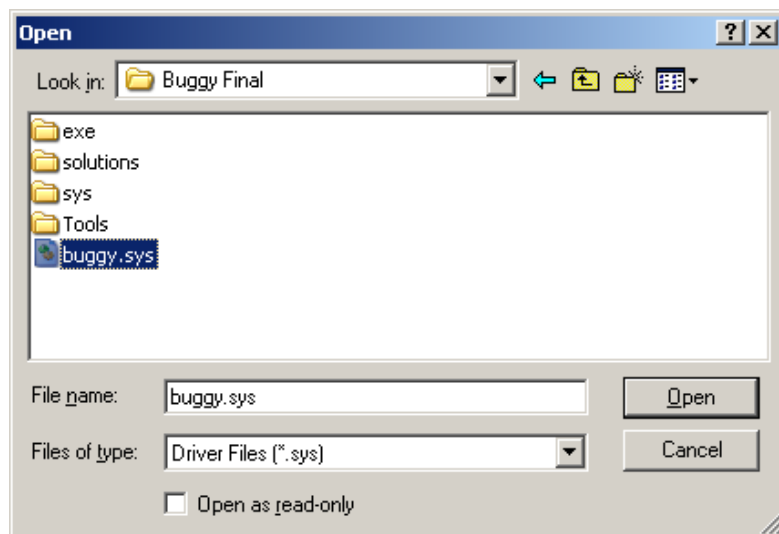
In this section, you will install the buggy application and driver.

This way, you will be able to generate various bugchecks and failures and debug them lively using WinDbg.

1. Locate **Buggy.exe** on the CD in **labs\Buggy** and launch this application.



2. Click on the “**Install Driver...**” button.



3. Select **buggy.sys** and click **Open**.

Part 2: Using the Buggy application

The application offers the possibility to generate various bugchecks and failures.

You can select the Failure type using the radio buttons. The choices are:

- **“Generate the following bugcheck:”**
Selecting this option allows you to choose a stop screen you want to generate. You can generate blue screens associated with the following codes: 0x0A, 0x4E, 0x7E, 0x35, 0x50, 0xD1.
- **“Generate the following failure:”**
Selecting this option allows you to generate the following problems: Deadlocks, Pool corruptions, Pool leaks, CPU used at 100%.

Once the bugcheck or failure has been select, pressing the **“Go !”** button will generate the problem. So, take note that your system will become unusable once this button has been pressed!

Part 3: Choosing which problem you want to debug

The problems that are generated by Buggy are very different and require more or less skills to debug. The solution for all the cases are on the CD in the **labs\buggy\solutions** directory. Please try to debug the problems using the knowledge you have gained during this training and have a look at the solution later.

Here is a table summarizing the problems generated by the application and the driver. A difficulty level is associated with each of them:

Bugcheck code	Description	Difficulty
0x0A	IRQL_NOT_LESS_OR_EQUAL	★
0x4E	PFN_LIST_CORRUPT	★★
0x7E	SYSTEM_THREAD_EXCEPTION_NOT_HANDLED	★★★
0x35	NO_MORE_IRP_STACK_LOCATIONS	★
0x50	PAGE_FAULT_IN_NONPAGED_AREA	★★
0xD1	DRIVER_IRQL_NOT_LESS_OR_EQUAL	★

Failure type	Description	Difficulty
Paged Pool Corruption	Memory corruption in the paged pool.	★
Non Paged Pool Corruption	Memory corruption in the non paged pool.	★
Paged Pool Leak	Memory leak coming from paged pool	★★
Non Paged Pool Leak	Memory leak coming from non paged pool.	★★
Deadlock	Deadlock with dispatcher synchronization objects	★★★★

Deadlock 2	Deadlock involving other synchronization objects	★ ★ ★
100% CPU	CPU fully used by a thread	★ ★

★	Easy
★ ★	Medium
★ ★ ★	Difficult
★ ★ ★ ★	Very difficult

Please first choose a problem with a difficulty level associated with your confidence and experience level.

You might begin with an easy problem first.

Step 4 – Debugging the problem

Once you have chosen the problem you want to troubleshoot, setup the kernel debugger as you have learned in the previous labs.

You will need the symbols file for `buggy.sys` and you might use the sources of this driver as well.

The symbol file `Buggy.pdb` is located in **Labs\Buggy** on your CD.

The sources of the driver are located in **Labs\Buggy\sys** and the sources of the application are located in **Labs\Buggy\exe**.

Once the debugger is attached and setup, you can go ahead and generate the problem of your choice on the target computer. You can try debugging the problem in two different ways:

1. Have the debugger connected to the target system, and stimulate the problem. Instead of producing a blue screen and a crash dump, the system will break into the debugger. You will be able to use the debugger to examine the **Buggy** driver's code at the point of failure.
2. Examine the source code for **buggy.sys** and determine which driver routines are involved in the bugcheck or failure scenario you are exercising. Before using the **buggy** application to stimulate the problem, break into the target system with the debugger and set a breakpoint on one of those routines. When you stimulate the problem, the system should encounter the breakpoint and the debugger should get control. You can then use the debugger's **s (step)**, **t (trace)**, and other commands to examine the behavior of the code in detail.

Don't forget to use the WinDbg help file for this exercise – it contains valuable information!

Have some extra time?

If you have some extra time at the end of this lab period, try enabling Driver Verifier for the driver `buggy.sys`, and see if you can get Driver Verifier to catch either the "memory corruption in paged pool" or "memory corruption in nonpaged pool" problem with this driver!

Exercise 3: Finding a Resource Deadlock

In this exercise, you will use WinDbg to load a sample dump file named `deadlock.dmp`. The file is located in the `c:\dumps` directory. In this lab, use the debugger commands we have covered so far and try to locate the resource deadlock.

Note: This file is build 2195sp1.

Loading the *deadlock.dmp* file with WinDbg

1. Select **Start | Program | Microsoft Debugging Tools | WinDbg**.
2. Select **File | Open Crash Dumps**.
3. Select **deadlock.dmp**.

Hint: Start with `!locks`. Determine which threads own which resources. Then determine what they are waiting on...

Exercise 4: Using PoolMon to find a Kernel-Mode Memory Leak

In this exercise, you will use a utility called *memleak.exe*. You can find this tool in the Labs share in the directory memtools.

Like the previous lab, in this lab we will not be giving step by step instructions.

1. Expand the memleak tool to your local system by typing “**memleak_x86 -c**”.
2. After expanding all of the files, run the executable image *memleak_x86.exe* to install the kernel mode portion of this utility. A reboot will be necessary.
3. Using *memleak.exe*, create a non-paged pool leak of size 1000 bytes. Read the *memleak.txt* file for the correct syntax (**memleak 1 1000**).
4. Start perfmon, and log non-paged pool bytes every 15 seconds.
5. Now continue to make more allocations and see if you can see them in perfmon.
6. Now run poolmon and see if you can determine what tag the driver is using.

Exercise 5: Remote Debugging with WinDbg

Starting a Remote Debugging Session With WinDbg

1. Click the **Start | Programs | Microsoft Debugging Tools | WinDbg**.
2. Select **File | Kernel Debug...** or press **Ctrl+K**
3. Enter *19200* for Baud Rate and *COM1* for Port.
4. Select OK, then select YES to save base workspace information.
5. Select **Debug | Break** or press **Ctrl+Break**
6. Type **.reload** and press <enter>.
7. Type **.server npipe:pipe=debug** and press <enter>.

The screenshot shows the Windows Debugger (WinDbg) window titled "Windows Debugger:5.1.2250.3 - Server 'npipe:pipe=debug' - Kernel 'comport=COM1,baud=19200'". The interface includes a menu bar (File, Edit, View, Debug, Window, Help), a toolbar, and a Command window. The Command window displays the following text:

```

Kernel base = 0x80400000 PsLoadedModuleList = 0x8046a4c0
Break instruction exception - code 80000003 (first chance)
*****
* You are seeing this message because you pressed either *
* CTRL+C (if you run kd.exe) or, *
* CTRL+BREAK (if you run WinDBG), *
* on your debugger machine's keyboard. *
*
* THIS IS NOT A BUG OR A SYSTEM CRASH *
*
* If you did not intend to break into the debugger, press the "g" key, then *
* press the "Enter" key now. This message might immediately reappear. If it *
* does, press "g" and "Enter" again. *
*
*****
ntoskrnl!RtlpBreakWithStatusInstruction:
80452e70 cc int 3
kd> .reload
Connected to Windows 2000 2195 x86 compatible target, ptr64 FALSE
Loading Kernel Symbols
.....Unable to read image header for fdc.sys at f
Loading User Symbols
PPEB is NULL
kd> .server npipe:pipe=debug
Server started with 'npipe:pipe=debug'
kd>

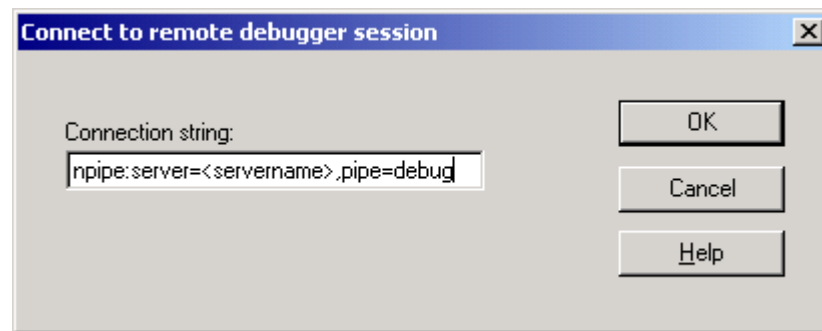
```

The status bar at the bottom shows "Ln 0, Col 0", "Proc 000:0", "Thrd 000:0", and buttons for "ASM", "OVR", "CAPS", and "NUM".

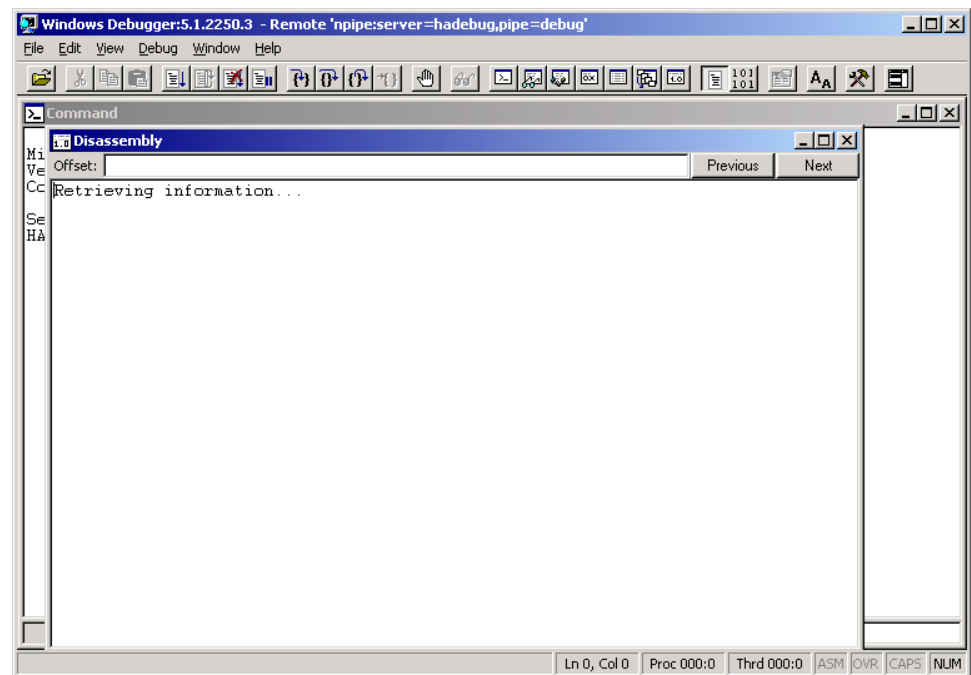
Connecting to a Remote Debugging Session With WinDbg

For this exercise, work with the group next to you and have them connect to your remote debug session while you connect to theirs from your host computers.

1. Click the **Start | Programs | Microsoft Debugging Tools | WinDbg**.
2. Select **File | Connect to Remote Sessions...** or press **Ctrl+R**.
3. Type “**npipeserver=<servername>, pipe=debug**” and select **OK**.



You should see something like this:



Once you are connected your WinDbg in Server mode will look like this:

Windows Debugger:5.1.2250.3 - Server 'npipes:pipe=debug' - Kernel 'com:port=com1,baud=19200'

File Edit View Debug Window Help

Command

```

Break instruction exception - code 80000003 (first chance)
*****
*   You are seeing this message because you pressed either
*   CTRL+C (if you run kd.exe) or,
*   CTRL+BREAK (if you run WinDBG),
*   on your debugger machine's keyboard.
*
*   THIS IS NOT A BUG OR A SYSTEM CRASH
*
*   If you did not intend to break into the debugger, press the "g" key, then
*   press the "Enter" key now. This message might immediately reappear. If it
*   does, press "g" and "Enter" again.
*
*****
ntoskrnl!RtlpBreakWithStatusInstruction:
80452e70 cc      int     3
kd> .reload
Connected to Windows 2000 2195 x86 compatible target, ptr64 FALSE
Loading Kernel Symbols
.....Unable to read image header for fdc.sys at f
.....
Loading User Symbols
PFEB is NULL
kd> .server npipes:pipe=debug
Server started with 'npipes:pipe=debug'
HADEBUG\toddwe (npipes debug) connected at Thu Aug 24 16:53:12 2000

kd>

```

Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

Your WinDbg session running in Remote mode will look like this:

Windows Debugger:5.1.2250.3 - Remote 'npipes:server=hadebug,pipe=debug'

File Edit View Debug Window Help

Command

```

Microsoft (R) Windows Kernel Debugger
Version 5.1.2250.5
Copyright (C) Microsoft Corp. 1981-2000

Server started with 'npipes:pipe=debug'
HADEBUG\toddwe (npipes debug) connected at Thu Aug 24 16:53:12 2000

kd>

```

Ln 0, Col 0 Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

Module 11 Labs: User-Mode Debugging

Lab Objectives



Lab 11 – User-Mode Debugging

Exercise 1 – Attaching to Running Process

Exercise 2 – Using Some Basic Debugger Commands in User Mode

Exercise 3 – Analyzing a User-Mode Dump File

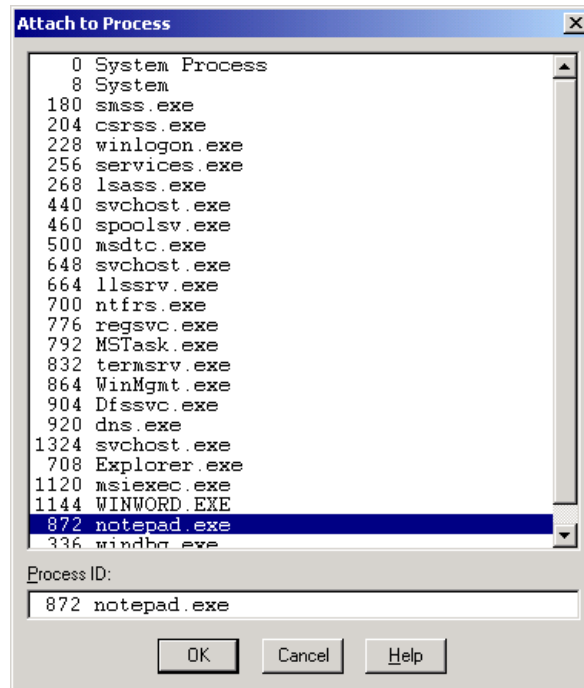
Exercise 4 – Using UMDH to Find a Memory Leak in an Application

Estimated Time to Complete this Lab: 60 minutes

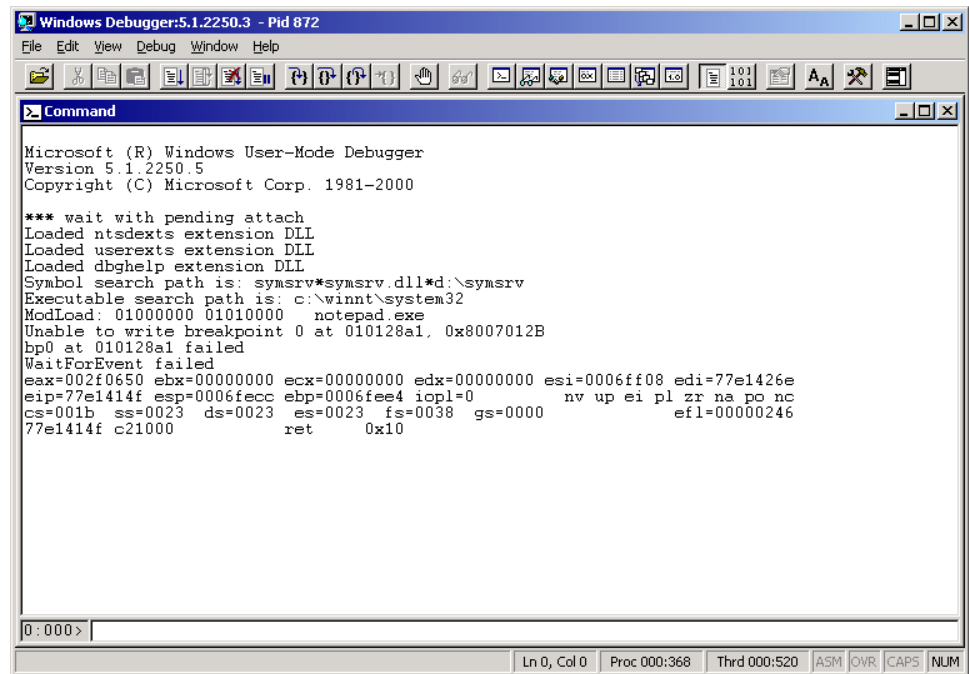
Exercise 1: Attaching to Running Process

Attach to a Running Process Using WinDbg

1. Select **Start | Run** and type “notepad” into the test box and select OK or press <enter>.
2. Select **Start | Programs | Microsoft Debugging Tools | WinDbg**.
3. Select **File | Attach to a process...**

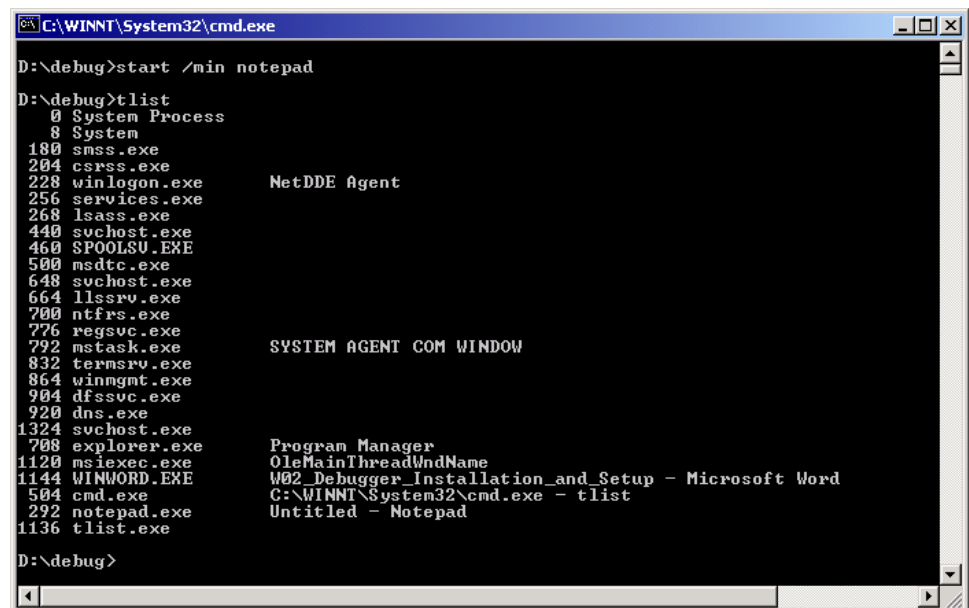


4. Scroll down and select notepad.exe from the list.
5. Select OK.

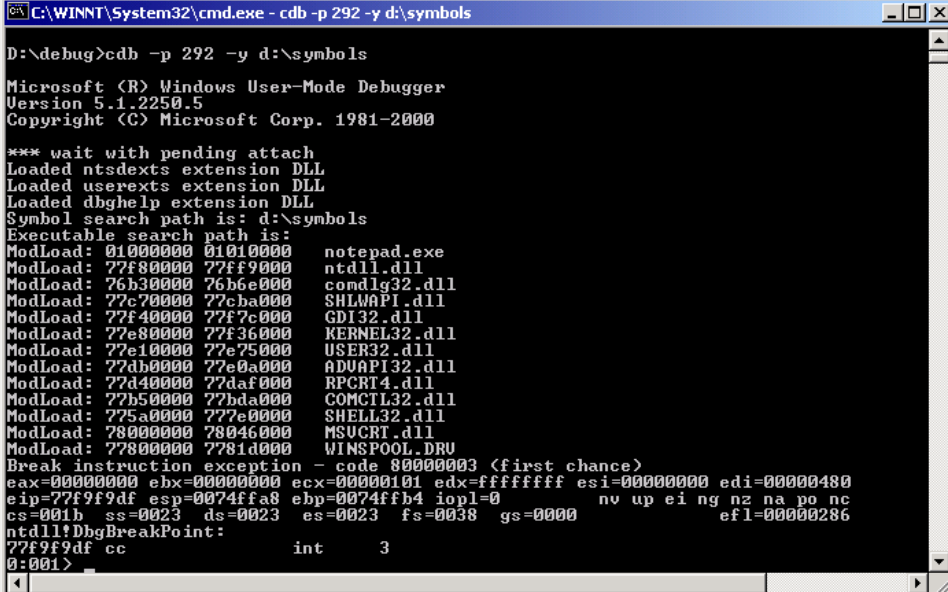


Attach to a Running Process Using CDB

1. Select **Start | Run** and type “cmd” into the test box and select OK or press <enter>.
2. Change into the “C:\Program Files\Debugging Tools for Windows” directory.
3. Type “**start /min notepad** <enter>”
4. Type “**tlist** <enter>”.



Type :“**cdb -p <PID> -y**
srv*c:\symbols\2195*<http://msdl.microsoft.com/download/symbols>.
 <enter>”.



```

C:\WINNT\System32\cmd.exe - cdb -p 292 -y d:\symbols
D:\debug>cdb -p 292 -y d:\symbols
Microsoft (R) Windows User-Mode Debugger
Version 5.1.2250.5
Copyright (C) Microsoft Corp. 1981-2000

*** wait with pending attach
Loaded ntsdexts extension DLL
Loaded userexts extension DLL
Loaded dbghelp extension DLL
Symbol search path is: d:\symbols
Executable search path is:
ModLoad: 01000000 01010000 notepad.exe
ModLoad: 77f80000 77ff9000 ntdll.dll
ModLoad: 76b30000 76b6e000 comdlg32.dll
ModLoad: 77c70000 77c7a000 SHLWAPI.dll
ModLoad: 77f40000 77f7c000 GDI32.dll
ModLoad: 77e80000 77f36000 KERNEL32.dll
ModLoad: 77e10000 77e75000 USER32.dll
ModLoad: 77db0000 77e0a000 ADVAPI32.dll
ModLoad: 77d40000 77daf000 RPCRT4.dll
ModLoad: 77b50000 77bda000 COMCTL32.dll
ModLoad: 775a0000 777e0000 SHELL32.dll
ModLoad: 78000000 78046000 MSUCRT.dll
ModLoad: 77800000 7781d000 WINSPOOL.DRV
Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=00000101 edx=ffffffff esi=00000000 edi=00000480
eip=77f9f9df esp=0074ffa8 ebp=0074ffb4 iopl=0         nv up ei ng nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000286
ntdll!DbgBreakPoint:
77f9f9df cc                int     3
0:001>

```

5. Type “q <Enter>” to quit.

Exercise 2: Using Some Basic Debugger Commands in User Mode

This lab assumes that you have completed the previous labs.

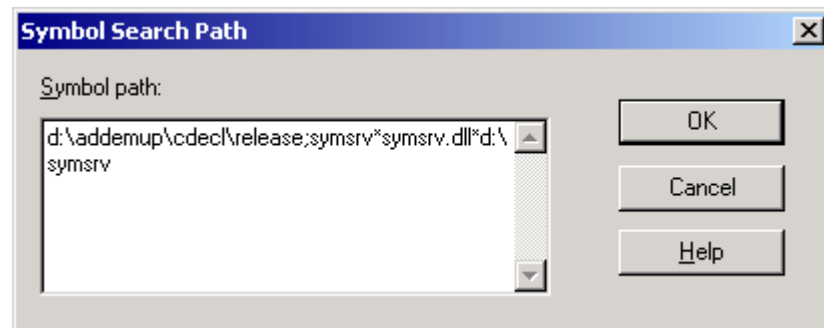
Note: Illustrations may differ from actual instructions.

Debugging a Process Using WinDbg

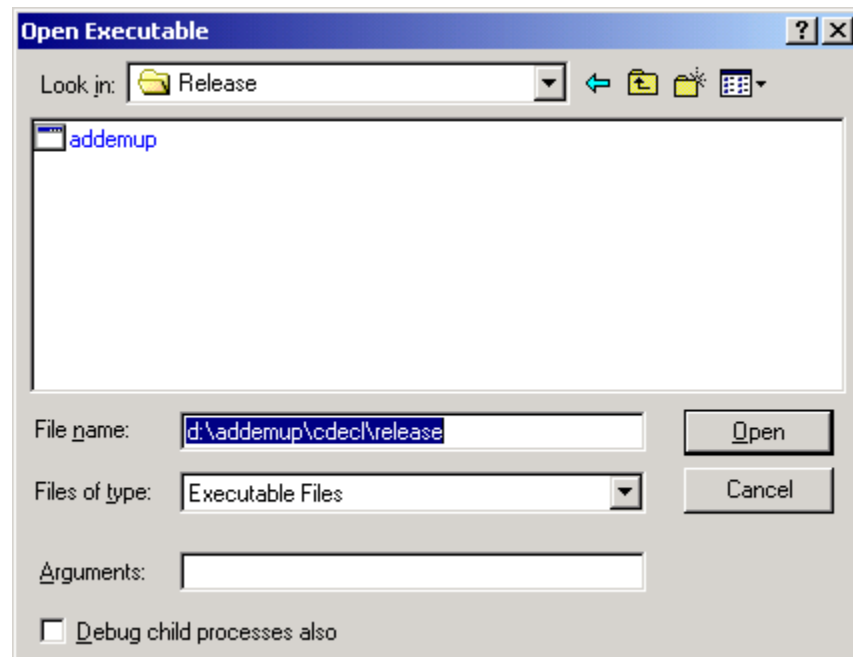
1. Map a drive to the *Labs* share on your instructor's computer.
2. Copy the Addemup folder down to your local C: drive.

Note: The Addemup folder is located on your class CD in the `\labs\files\addemup` directory.

3. Select **Start | Programs | Microsoft Debugging Tools | WinDbg**.
4. Select **File | Symbol File Path...**
5. Add "`C:\Addemup\cdecl\release;`" to the start of the symbol path and select OK.



6. Select **File | Open Executable**.
7. Type "`c:\Addemup\cdecl\release` <enter>."



8. Select the application “Addemup” and then select Open.
 9. You will see the “Save base workspace” dialog box again. Select Yes to continue:
- Now you should see some thing like this:

```

Windows Debugger:5.1.2250.3 - D:\addemup\cdecl\Release\addemup.exe
File Edit View Debug Window Help
[Icons]
Command
Microsoft (R) Windows User-Mode Debugger
Version 5.1.2250.5
Copyright (C) Microsoft Corp. 1981-2000

CommandLine: D:\addemup\cdecl\Release\addemup.exe
Loaded ntsdexts extension DLL
Loaded userexts extension DLL
Loaded dbghelp extension DLL
Symbol search path is: d:\addemup\cdecl\release;symsrv*symsrv.dll*d:\symsrv
Executable search path is: c:\winnt\system32
ModLoad: 00400000 0040c000  addemup.exe
ModLoad: 77f80000 77ff9000  ntdll.dll
ModLoad: 77e80000 77f36000  C:\WINNT\system32\KERNEL32.dll
Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00131f04 ecx=00000009 edx=00000000 esi=7ffdf000 edi=00131f70
eip=77f9f9df esp=0012f984 ebp=0012fc98 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!DbgBreakPoint:
77f9f9df cc          int     3

0:000>
Ln 0, Col 0  Proc 000:574  Thrd 000:594  ASM  OVR  CAPS  NUM

```

Note: After opening Addemup.exe, occasionally the default window of WinDbg switches to the Disassembly screen. If this occurs, click on the Window menu and select Command from the drop-down list.

10. Enter “LM <enter>” into the command window to get a list of loaded modules.

```

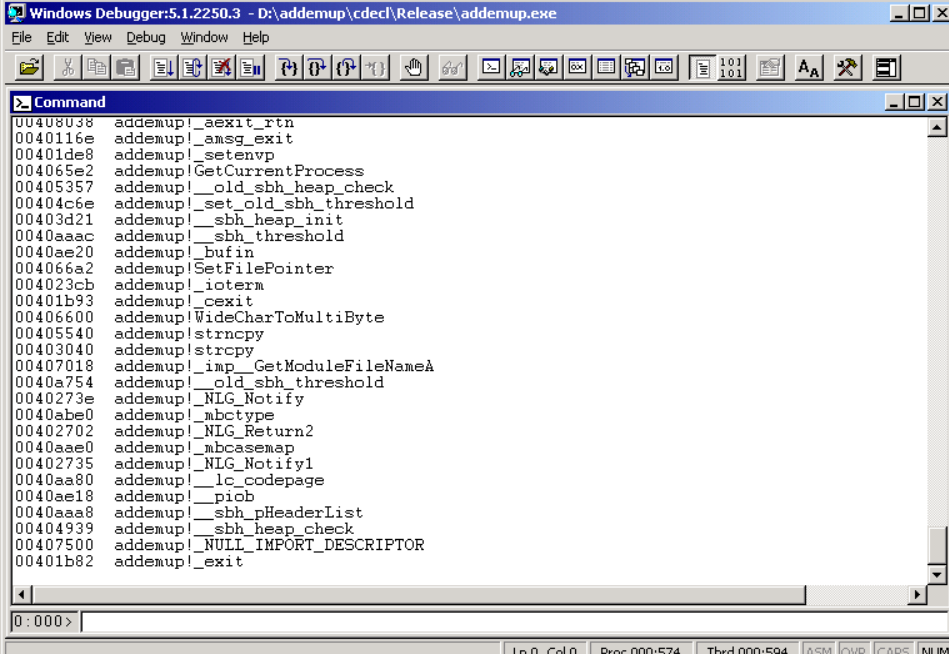
Windows Debugger:5.1.2250.3 - D:\addemup\cdecl\Release\addemup.exe
File Edit View Debug Window Help
[Icons]
Command
Microsoft (R) Windows User-Mode Debugger
Version 5.1.2250.5
Copyright (C) Microsoft Corp. 1981-2000

CommandLine: D:\addemup\cdecl\Release\addemup.exe
Loaded ntsdexts extension DLL
Loaded userexts extension DLL
Loaded dbghelp extension DLL
Symbol search path is: d:\addemup\cdecl\release;symsrv*symsrv.dll*d:\symsrv
Executable search path is: c:\winnt\system32
ModLoad: 00400000 0040c000  addemup.exe
ModLoad: 77f80000 77ff9000  ntdll.dll
ModLoad: 77e80000 77f36000  C:\WINNT\system32\KERNEL32.dll
Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00131f04 ecx=00000009 edx=00000000 esi=7ffdf000 edi=00131f70
eip=77f9f9df esp=0012f984 ebp=0012fc98 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!DbgBreakPoint:
77f9f9df cc          int     3

0:000> lm
start      end          module name
00400000 0040c000  addemup      (deferred)
77e80000 77f36000  kernel32     (deferred)
77f80000 77ff9000  ntdll        (pdb symbols)          d:\symsrv\ntdll.dbg\38175b3079000

```

11. Type “**x addemup!* <enter>**” to get a look at the functions within the addemup module.



Windows Debugger:5.1.2250.3 - D:\addemup\cdecl\Release\addemup.exe

File Edit View Debug Window Help

Command

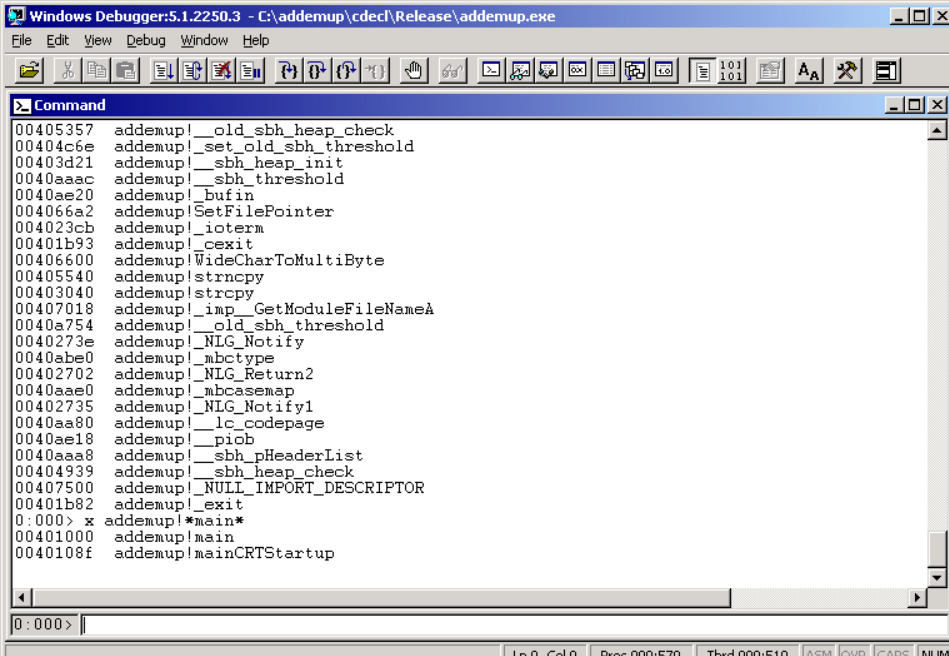
```

00408038 addemup!_aexit_rtn
0040116e addemup!_amsq_exit
00401de8 addemup!_setenvp
004065e2 addemup!GetCurrentProcess
00405357 addemup!__old_sbh_heap_check
00404c6e addemup!_set_old_sbh_threshold
00403d21 addemup!__sbh_heap_init
0040aaac addemup!__sbh_threshold
0040ae20 addemup!_bufin
004066a2 addemup!SetFilePointer
004023cb addemup!_ioterm
00401b93 addemup!_cexit
00406600 addemup!WideCharToMultiByte
00405540 addemup!strncpy
00403040 addemup!strcpy
00407018 addemup!_imp_GetModuleFileNameA
0040a754 addemup!__old_sbh_threshold
0040273e addemup!_NLG_Notify
0040abe0 addemup!_mbctype
00402702 addemup!_NLG_Return2
0040aae0 addemup!_mbcasemap
00402735 addemup!_NLG_Notify1
0040aa80 addemup!__lc_codepage
0040ae18 addemup!__piob
0040aaa8 addemup!__sbh_pHeaderList
00404939 addemup!__sbh_heap_check
00407500 addemup!_NULL_IMPORT_DESCRIPTOR
00401b82 addemup!_exit
  
```

0:000>

Ln 0, Col 0 Proc 000:574 Thrd 000:594 ASM OVR CAPS NUM

12. As you can see that can be a lot of information. This time let's search for the main() function within addemup. Type “**x addemup!*main* <enter>**.”



Windows Debugger:5.1.2250.3 - C:\addemup\cdecl\Release\addemup.exe

File Edit View Debug Window Help

Command

```

00405357 addemup!__old_sbh_heap_check
00404c6e addemup!_set_old_sbh_threshold
00403d21 addemup!__sbh_heap_init
0040aaac addemup!__sbh_threshold
0040ae20 addemup!_bufin
004066a2 addemup!SetFilePointer
004023cb addemup!_ioterm
00401b93 addemup!_cexit
00406600 addemup!WideCharToMultiByte
00405540 addemup!strncpy
00403040 addemup!strcpy
00407018 addemup!_imp_GetModuleFileNameA
0040a754 addemup!__old_sbh_threshold
0040273e addemup!_NLG_Notify
0040abe0 addemup!_mbctype
00402702 addemup!_NLG_Return2
0040aae0 addemup!_mbcasemap
00402735 addemup!_NLG_Notify1
0040aa80 addemup!__lc_codepage
0040ae18 addemup!__piob
0040aaa8 addemup!__sbh_pHeaderList
00404939 addemup!__sbh_heap_check
00407500 addemup!_NULL_IMPORT_DESCRIPTOR
00401b82 addemup!_exit
0:000> x addemup!*main*
00401000 addemup!main
0040108f addemup!mainCRTStartup
  
```

0:000>

Ln 0, Col 0 Proc 000:570 Thrd 000:510 ASM OVR CAPS NUM

13. Type “**BU addemup!main <enter>**” or “**BU 401000 <enter>**”. This sets a break point on the main() function.
14. Type “**BU addemup!addemup <enter>**.” This sets a break point on the addemup() function within the process addemup.
15. Type “**BL <enter>**” to see a list of breakpoints.

```

Windows Debugger:5.1.2250.3 - C:\addemup\cdec\Release\addemup.exe
File Edit View Debug Window Help
Loaded ntsdexts extension DLL
Loaded userexts extension DLL
Loaded dbghelp extension DLL
Symbol search path is: c:\addemup\cdec\release;symsrv*symsrv.dll*\\cokemachine\symsrv
Executable search path is: c:\winnt\system32
ModLoad: 00400000 0040c000 addemup.exe
ModLoad: 77f80000 77f90000 ntdll.dll
ModLoad: 77e80000 77f36000 C:\WINNT\system32\KERNEL32.dll
Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00131f04 ecx=00000009 edx=00000000 esi=7ffdf000 edi=00131f70
eip=77f9f9df esp=0012f984 ebp=0012fc98 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
ntdll!DbgBreakPoint:
77f9f9df cc                int     3
0:000> lm
start      end             module name
00400000 0040c000  addemup      (deferred)
77e80000 77f36000  kernel32     (deferred)
77f80000 77f90000  ntdll        (pdb symbols)          \\cokemachine\symsrv\ntdll.dbg\
0:000> x addemup!main*
00401000  addemup!main
0040108f  addemup!mainCRTStartup
0:000> bp addemup!main
0:000> bp addemup!addemup
0:000> bl
0 e 00401000 0001 (0001) 0:*** addemup!main
1 e 00401043 0001 (0001) 0:*** addemup!Addemup
0:000>
Ln 0, Col 0  Proc 000:510  Thrd 000:570  ASM OVR CAPS NUM

```

16. Type “g <enter>” to cause the process to run (“go”).
17. When the first breakpoint is hit, the debugger will stop and display the disassembly window at that spot in the code.
18. Select **Windows | Arrange All Windows**.

```

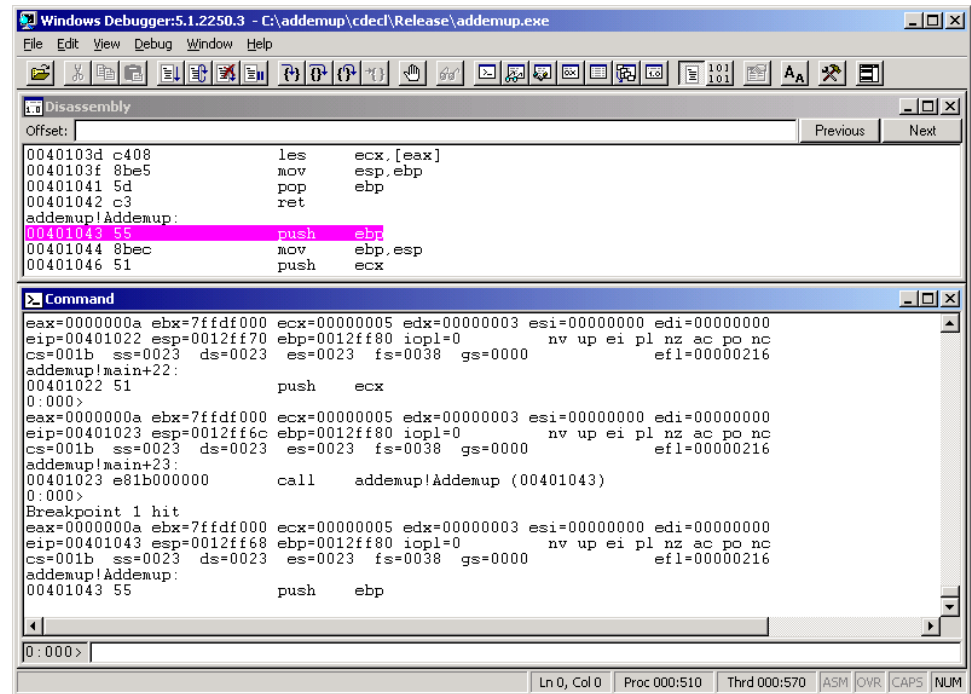
Windows Debugger:5.1.2250.3 - C:\addemup\cdec\Release\addemup.exe
File Edit View Debug Window Help
Disassembly
Offset:
00400ff8 0000      add     [eax],al
00400ffa 0000      add     [eax],al
00400ffc 0000      add     [eax],al
00400ffe 0000      add     [eax],al
addemup!main:
00401000 55      push    ebp
00401001 8bec    mov     ebp,esp
00401003 83ec0c  sub     esp,0xc
Command
00400000 0040c000  addemup      (deferred)
77e80000 77f36000  kernel32     (deferred)
77f80000 77f90000  ntdll        (pdb symbols)          \\cokemachine\symsrv\ntdll.dbg\381:
0:000> x addemup!main*
00401000  addemup!main
0040108f  addemup!mainCRTStartup
0:000> bp addemup!main
0:000> bp addemup!addemup
0:000> bl
0 e 00401000 0001 (0001) 0:*** addemup!main
1 e 00401043 0001 (0001) 0:*** addemup!Addemup
0:000> g
Breakpoint 0 hit
eax=002f0db8 ebx=7ffdf000 ecx=004080b8 edx=00000003 esi=00000000 edi=00000000
eip=00401000 esp=0012ff84 ebp=0012ffc0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000206
addemup!main:
00401000 55      push    ebp
0:000>
Ln 0, Col 0  Proc 000:510  Thrd 000:570  ASM OVR CAPS NUM

```

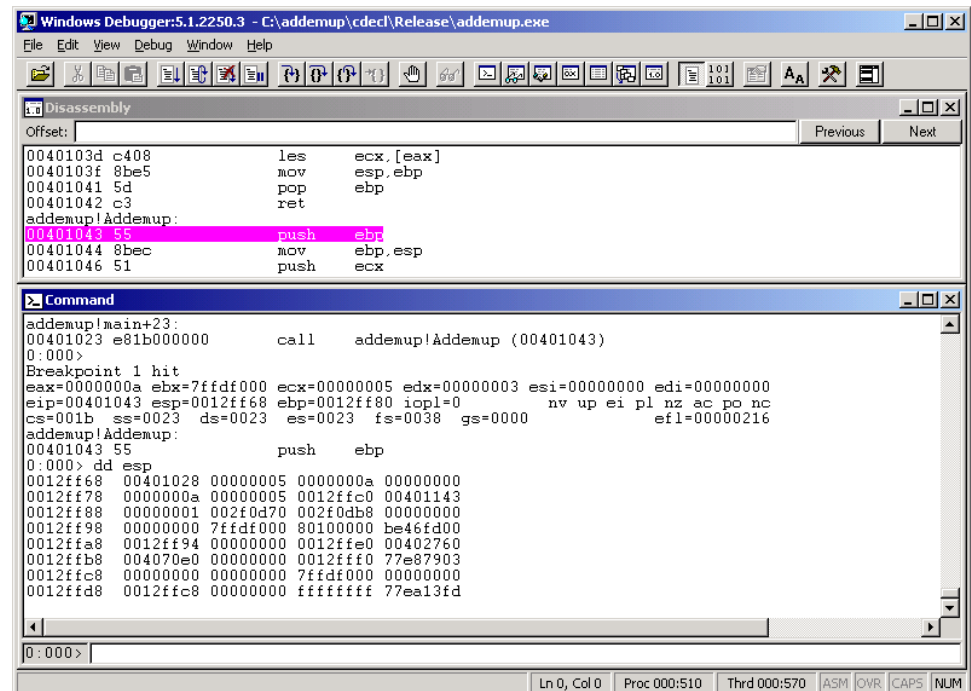
19. Type “t <enter>” to trace through the code one instruction at a time.
20. Hit the <enter> key several times and watch the debugger step through the assembly code in the top window. Hitting the <enter> key alone will repeat the last instruction.

21. Stop as soon as the following instruction is executed: “call addemup!addemup”.

The Debugger should look similar to this:

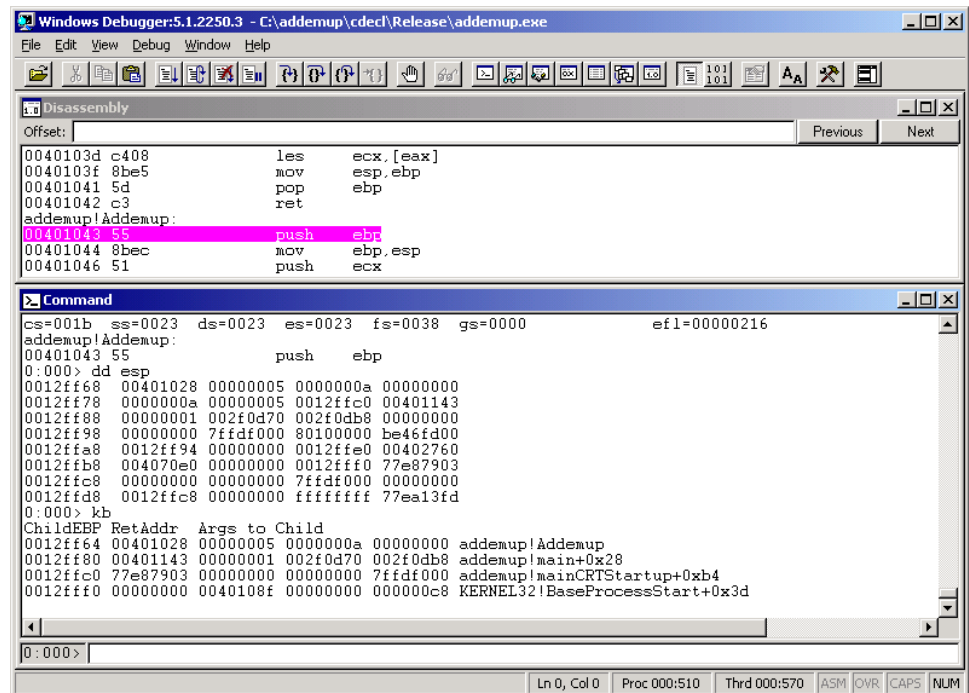


22. Type “dd esp <enter>” to look at the current stack.

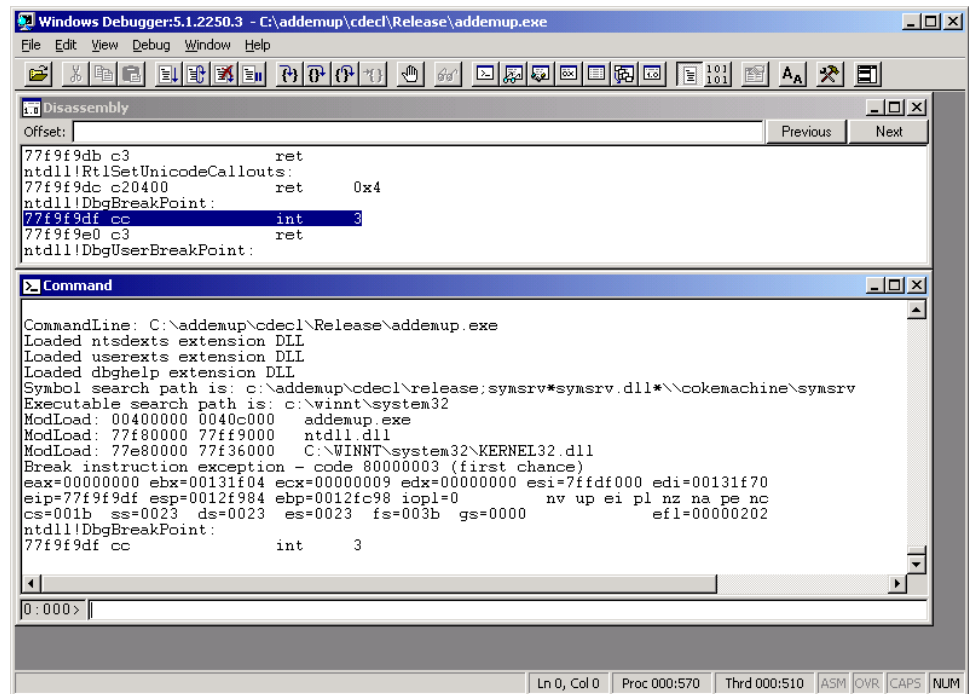


Notice the return address at the top of the stack. You can also see the two parameters being passed on the stack.

23. Type “kb <enter>” to look at the current stack trace.



24. Select **Debug | Restart**. Select **Yes** to save workspace information when the dialog box pops up. This will restart the process back to when all the modules are loaded.



25. Type **"bl <enter>"** again to look at the current break points.
26. Type **"bd 0 <enter>"** to disable break point 0 at addemup!main().
27. Type **"bl <enter>"** again to see that the break point is disabled.
28. Type **"g <enter>"** to cause the process to run ("Go").

Windows Debugger:5.1.2250.3 - C:\addemup\cdec\Release\addemup.exe

File Edit View Debug Window Help

Disassembly

Offset: Previous Next

```

0040103f 8be5    mov     esp,ebp
00401041 5d      pop     ebp
00401042 c3      ret
addemup!Addemup:
00401043 55      push    ebp
00401044 8bec    mov     ebp,esp
00401046 51      push    ecx

```

Command

```

ntdll!DbgBreakPoint:
77f9f9df cc      int     3
0:000> bl
0 e 00401000 0001 (0001) 0:*** addemup!main
1 e 00401043 0001 (0001) 0:*** addemup!Addemup
0:000> bd 0
0:000> bl
0 d 00401000 0001 (0001) 0:*** addemup!main
1 e 00401043 0001 (0001) 0:*** addemup!Addemup
0:000> g
Breakpoint 1 hit
eax=0000000a ebx=7ffdf000 ecx=00000005 edx=00000003 esi=00000000 edi=00000000
eip=00401043 esp=0012ff68 ebp=0012ff80 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000216
addemup!Addemup:
00401043 55      push    ebp

```

Ln 0, Col 0 Proc 000:570 Thrd 000:510 ASM OVR CAPS NUM

29. Type “**kb** <enter>” to look at the current stack trace.
30. Now let’s have some fun. Type “**dd esp** <enter>” to dump the stack again.
31. Now let’s change the second parameter to 0x00000005. Type “**ed 12ff70** <enter>”.
32. Type “**00000005** <enter> <enter>”. That’s seven 0’s, one 5, and two enters!
33. Type “**dd esp** <enter>” again to see the changes...

Windows Debugger:5.1.2250.3 - C:\addemup\cdec\Release\addemup.exe

File Edit View Debug Window Help

Disassembly

Offset: Previous Next

```

0040103f 8be5    mov     esp,ebp
00401041 5d      pop     ebp
00401042 c3      ret
addemup!Addemup:
00401043 55      push    ebp
00401044 8bec    mov     ebp,esp
00401046 51      push    ecx

```

Command

```

0012ffa8 0012ff94 00000000 0012ffe0 00402760
0012ffb8 004070e0 00000000 0012fff0 77e87903
0012ffc8 00000000 00000000 7ffdf000 00000000
0012ffd8 0012ffc8 00000000 ffffffff 77eal3fd
0:000> ed 12ff70
0012ff70 0000000a 00000005
0012ff74 00000000
0:000> dd esp
0012ff68 00401028 00000005 00000005 00000000
0012ff78 0000000a 00000005 0012ffc0 00401143
0012ff88 00000001 002f0d70 002f0db8 00000000
0012ff98 00000000 7ffdf000 80100000 be7dcd00
0012ffa8 0012ff94 00000000 0012ffe0 00402760
0012ffb8 004070e0 00000000 0012fff0 77e87903
0012ffc8 00000000 00000000 7ffdf000 00000000
0012ffd8 0012ffc8 00000000 ffffffff 77eal3fd

```

Ln 0, Col 0 Proc 000:510 Thrd 000:570 ASM OVR CAPS NUM

34. Now let’s step through a line at a time and see what our change did. Type “**t** <enter>”.

35. We execute “push ebp”. You can type “**dd esp** <enter>” to check – this is where we save the ebp for the previous function “ChildEBP”
36. Type “**t** <enter>” again. Note that in the assembly window the red highlighting stays on the first instruction of the function. That indicates the break point. Another confusing point is that after each **T** (trace) command, the debugger window shows the current registers and the next instruction, not the instruction that was just executed. The assembly window also highlights the next instruction to be executed in blue. So the last instruction we executed set our current stack pointer by copying the current stack pointer (register esp) into our base pointer register ebp. You can see this by comparing the current register settings for ebp and the previous one.
37. Type “**t** <enter>”. Now we push ecx to make room for our one local. Notice the stack pointer is decremented by 4 bytes.
38. Type “**t** <enter>”. Notice the instruction “*mov dword ptr [ebp-4], 0x0*”. If you remember from our discussion from the previous section. [ebp-4] is our first local variable. This line of code is zeroing out that location in memory. This is equivalent to our line of c code “int c = 0”.
39. Type “**t** <enter>”. The next instruction “*mov eax, [ebp+8]*” is loading our first parameter into eax. Remember ebp+4 =return address, ebp+8 =first parameter, and ebp+c is our second parameter. The [] mean to move the contents stored at the address referenced by ebp+8 apposed to moving the value of ebp+8 into eax...
40. Type “**t** <enter>”. The next instruction “*add eax, [ebp+c]*” is the same as *eax=eax+[ebp+c]*. So we add what is stored in the register eax to the value stored at the location referenced by [ebp+c] and put the resulting sum back into eax.
41. Type “**t** <enter>”. The next instruction “*mov [ebp-4], eax*” is moving the result from the previous addition into the location referenced by [ebp-4] or into the location of our local variable ‘c’.
42. Type “**t** <enter>”. The next thing we do is copy that value right back into eax . This satisfies the c code instruction “*return (c)*”. If we had turned on compiler optimizations, you would see this overlap...
43. Notice at this point how our return value is 0x000000a. Let’s have even more fun and change it again.
44. Type “**t** <enter>” to move the value of Local ‘c’ back into eax so that we can return it to the calling function.
45. Type “**r eax=00000010** <enter>”. You can type “**R** <enter>” to verify the value was changed.
46. Type “**t** <enter>”. The next instruction “*mov esp, ebp*” copies the saved stack pointer back to it’s previous value.
47. Type “**t** <enter>”. The next instruction “*pop ebp*” restores our base pointer to its previous value.
48. Type “**t** <enter>”. We return back to the calling function to the next instruction following the call to addemup!addemup().
49. Type “**t** <enter>” next, since this was a CDECL function the calling function needs to add 8 bytes to the stack pointer to adjust for the two arguments we passed during the previous call to addemup!addemup().

50. Type “t <enter>”. The next instruction “*mov [ebp-0xc], eax*” loads the return value into the location referenced by [ebp-0xc] which is our third local variable or ‘z’.
51. Type “t <enter>”. The next instruction moves that value into edx.
52. Type “t <enter>”. The next instruction pushes edx onto the stack. This is the second parameter being passed to printf().
53. Type “t <enter>”. The next instruction “*push 408030*” is pushing a pointer or the memory location of the “z= %i\n” string being passed to printf(). Type “*dc 408030 L2<enter>*” to dump that memory location and view its contents. The “L2” tells the debugger to only dump 2 dwords.

Windows Debugger:5.1.2250.3 - C:\addemup\cdec\Release\addemup.exe

File Edit View Debug Window Help

Disassembly

Offset: Previous Next

0040102f	55	push	ebp
00401030	f4	hlt	
00401031	52	push	edx
00401032	6830804000	push	0x408030
00401037	e822000000	call	addemup!printf (0040105e)
0040103c	83c408	add	esp,0x8
0040103f	8be5	mov	esp,ebp

Command

```

addemup!main+2e:
0040102e 8b55f4      mov     edx,[ebp-0xc]    ss:0023:0012ff74=00000010
0:000>
eax=00000010 ebx=7ffdf000 ecx=00000005 edx=00000010 esi=00000000 edi=00000000
eip=00401031 esp=0012ff74 ebp=0012ff80 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000216
addemup!main+31:
00401031 52          push    edx
0:000>
eax=00000010 ebx=7ffdf000 ecx=00000005 edx=00000010 esi=00000000 edi=00000000
eip=00401032 esp=0012ff70 ebp=0012ff80 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000216
addemup!main+32:
00401032 6830804000 push    0x408030
0:000> dc 408030 12
00408030 25203d7a 00000a69      z= %i...

```

0:000> |

Ln 0, Col 0 Proc 000:470 Thrd 000:570 ASM OVR CAPS NUM

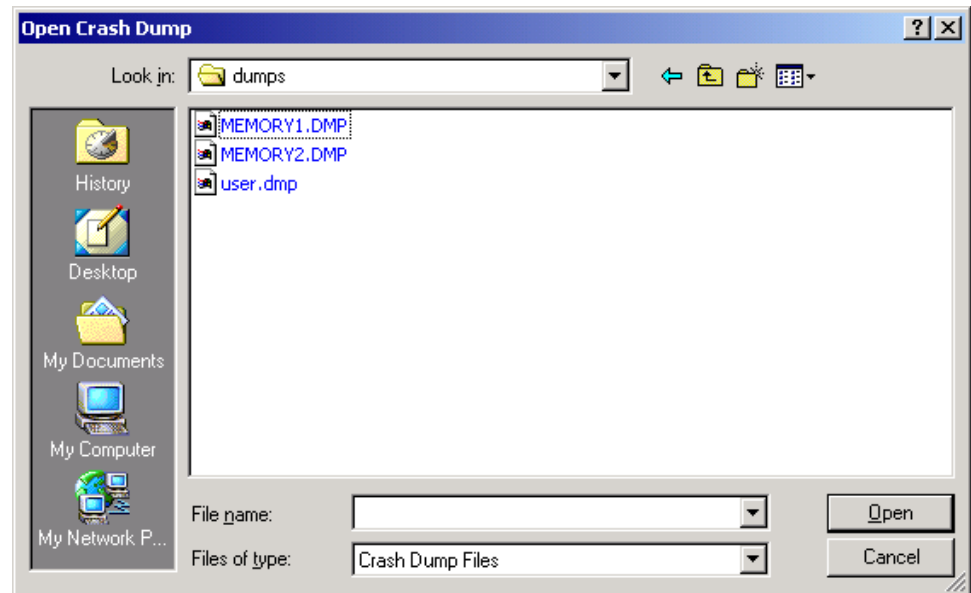
54. Type “t <enter>”. The next instruction is the actual call to printf().
55. Type “t <enter>”. Then Press <Shift-F11> to step out of printf(). Stepping through it line by line may take some time.... ☺
56. Take a look at the command window that was opened up when we started addemup. You should see the string “z= 16”
57. Type “t <enter>”. The next instruction in addemup!main, “*add esp, 0xc*”, fixes up the stack again for the parameters that we passed to printf(). What calling convention does printf() use?
58. The next couple of instructions just continue to clean up before we exit, but since we are 58 lines into the exercise we will just Type “q <enter> to quit....

Exercise 3: Analyzing a User-Mode Dump File

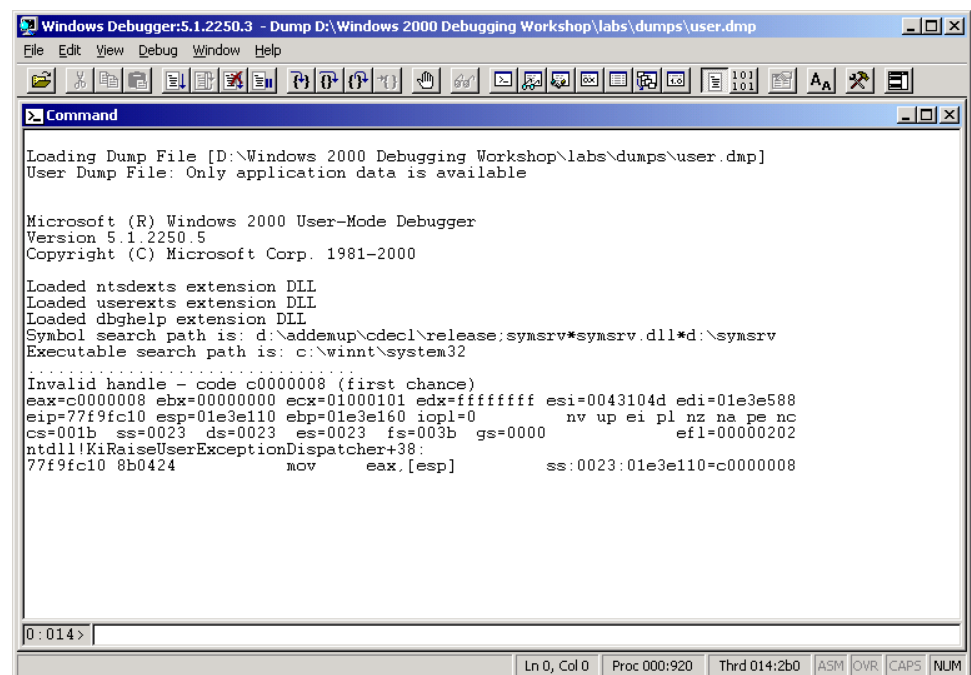
In this exercise, you will use WinDbg to load a couple of sample user.dmp files for analysis. The files are located in the `c:\dumps` directory.

Loading a user.dmp file using WinDbg

1. Select **Start | Program | Microsoft Debugging Tools | WinDbg**.
2. Select **File | Open Crash Dump**.



3. Select `user1.dmp` and then select Open.



4. We can see that the process failed with status `0xc0000008` `STATUS_INVALID_HANDLE`.

5. Type “**kb** <enter>” to get a look at the stack trace. This shows you what was going on at the point when we failed.
6. From the stack trace you can see that *inojobsv.exe* attempted to close an invalid handle. Most likely the handle had been previously freed!
7. Load up the additional *user.dmp* files and see if you can isolate the point of failure.

Exercise 4 – Using UMDH to Find a Memory Leak in an Application

In this exercise, you will use a utility called *leakyapp.exe* to generate a user mode leak. You can find this tool in the Labs share in the directory *memtools*. Like the previous lab, in this lab we will not be giving step by step instructions.

1. Expand the *umdhtools.exe* tool to your local system.
2. Start *leakyapp.exe*.
3. Use *tlist.exe* to get the PID for *leakyapp.exe*.
4. Use the *umdh.exe* utility to locate the leak.

Refer to the the workbook (Module 11, page 352) for instructions on how to use *umdh.exe*.

Hint: You need to set some global flags and reboot...