Windows Debugging³
Accelerated

Version 2.0

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

- Debugging in Visual Studio

or

- Basic crash dump analysis
Why WinDbg?

- Production debugging
- Redistributable
- Kernel mode debugging
Training Goals

- Review fundamentals
- Learn live debugging techniques
- See how software diagnostics is used during debugging
Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples
Course Idea

Debugging TV

- www.debugging.tv (more than 40 episodes)
- PDB Symbols: episodes 0x01 – 0x04
- Kernel debugging setup: episode 0x25
Schedule Summary

Day 1
- Debugging Fundamentals (30 minutes)
- User Mode Debugging (2 hour and 30 minutes)

Day 2
- User Mode Debugging (30 minutes)
- Kernel Mode Debugging (1 hour and 30 minutes)
- Managed Debugging (1 hour)
Part 1: Fundamentals
Memory Space³
Execution Mode³
Code³
Debugging Technique

Breakpoints

Tracing

Inspection
Pattern\(^3\)
Elementary Diagnostics

- Functional
  - Use-case Deviation

- Non-functional
  - Crash
  - Hang (includes delays)
  - Counter Value (includes resource leaks, CPU spikes)
  - Error Message
Analysis Patterns

- Memory Analysis catalogue
- Software Trace and Log Analysis catalogue
Pattern-Oriented Diagnostic Analysis

**Diagnostic Pattern**: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

**Diagnostic Problem**: a set of indicators (symptoms, signs) describing a problem.

**Diagnostic Analysis Pattern**: a common recurrent analysis technique and method of diagnostic pattern identification in a specific context.

**Diagnostics Pattern Language**: common names of diagnostic and diagnostic analysis patterns. The same language for any operating system: Windows, Mac OS X, Linux, ...

**Checklist**: [http://www.dumpanalysis.org/windows-memory-analysis-checklist](http://www.dumpanalysis.org/windows-memory-analysis-checklist)
Unified Debugging Patterns

- Analysis (software diagnostics)
- Architecture/Design of debugging
- Implementation of debugging
- Usage/presentation of debugging (for example, Watch dialog)
0x00> lm

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00007ff7`73080000</td>
<td>00007ff7`7330f000</td>
<td>App</td>
</tr>
<tr>
<td>00007ff8`44990000</td>
<td>00007ff8`44a27000</td>
<td>MyDLL</td>
</tr>
<tr>
<td>00007ff8`46a9d0000</td>
<td>00007ff8`46d39000</td>
<td>comctl32</td>
</tr>
<tr>
<td>00007ff8`56d70000</td>
<td>00007ff8`56d9e000</td>
<td>oleacc</td>
</tr>
<tr>
<td>00007ff8`56de0000</td>
<td>00007ff8`56f7a000</td>
<td>GdiPlus</td>
</tr>
<tr>
<td>00007ff8`58d00000</td>
<td>00007ff8`58e54000</td>
<td>winspool</td>
</tr>
<tr>
<td>00007ff8`5cc680000</td>
<td>00007ff8`5cfed000</td>
<td>CoreUIComponents</td>
</tr>
<tr>
<td>00007ff8`5cc7f0000</td>
<td>00007ff8`5ce2c000</td>
<td>ninput</td>
</tr>
<tr>
<td>00007ff8`5ce870000</td>
<td>00007ff8`5ce9b000</td>
<td>WinTypes</td>
</tr>
<tr>
<td>00007ff8`5cf1a0000</td>
<td>00007ff8`5cf1f000</td>
<td>propsys</td>
</tr>
<tr>
<td>00007ff8`5cf1f0000</td>
<td>00007ff8`5cf26000</td>
<td>winmm</td>
</tr>
<tr>
<td>00007ff8`5cf26000</td>
<td>00007ff8`5cf3a000</td>
<td>winmm</td>
</tr>
<tr>
<td>00007ff8`5cf8d0000</td>
<td>00007ff8`5cf94000</td>
<td>dwmapi</td>
</tr>
<tr>
<td>00007ff8`601c0000</td>
<td>00007ff8`601f1000</td>
<td>ntmsata</td>
</tr>
<tr>
<td>00007ff8`601f1000</td>
<td>00007ff8`60332000</td>
<td>IPHLPAPI</td>
</tr>
<tr>
<td>00007ff8`60f00000</td>
<td>00007ff8`60f0d000</td>
<td>bcyrypt</td>
</tr>
<tr>
<td>00007ff8`60f0d000</td>
<td>00007ff8`60f5f000</td>
<td>powrprof</td>
</tr>
<tr>
<td>00007ff8`60f5f000</td>
<td>00007ff8`60f5f000</td>
<td>fltLib</td>
</tr>
<tr>
<td>00007ff8`61120000</td>
<td>00007ff8`6112a000</td>
<td>kernel_appcore</td>
</tr>
<tr>
<td>00007ff8`6112a000</td>
<td>00007ff8`61134000</td>
<td>profapi1</td>
</tr>
<tr>
<td>00007ff8`61134000</td>
<td>00007ff8`6118f000</td>
<td>ucrbase</td>
</tr>
<tr>
<td>00007ff8`6118f000</td>
<td>00007ff8`61193000</td>
<td>cfgmgr32</td>
</tr>
<tr>
<td>00007ff8`61193000</td>
<td>00007ff8`6119f000</td>
<td>gdi32full</td>
</tr>
<tr>
<td>00007ff8`6119f000</td>
<td>00007ff8`611a1000</td>
<td>win32u</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>windows_storage</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>bcyryptPrimitives</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>msvcp_win</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>KERNELBASE</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>sehost</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>shell32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>oleaut32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>advapi32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>imm32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>user32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>msctf</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>gdi32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>msvcrtest</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>SHCore</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>rpcrt4</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>shlwapi</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>combination</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>ole32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>kernel32</td>
</tr>
<tr>
<td>00007ff8`611a1000</td>
<td>00007ff8`611a1000</td>
<td>nttdll</td>
</tr>
</tbody>
</table>

User Space

Kernel Space
Thread Stack Trace

FunctionA()
{
    ...
    FunctionB();
    ...
}

FunctionB()
{
    ...
    FunctionC();
    ...
}

FunctionC()
{
    ...
    FunctionD();
    ...
}

© 2018 Software Diagnostics Services
Thread Stack Trace (no PDB)

FunctionA()
{
    ...
    FunctionB();
    ...
}
FunctionB()
{
    ...
    FunctionC();
    ...
}
FunctionC()
{
    ...
    FunctionD();
    ...
}

Symbol file Module.pdb
FunctionA 22000 - 23000
FunctionB 32000 - 33000
FunctionC 43000 - 44000
FunctionD 54000 - 55000

No symbols for Module

© 2018 Software Diagnostics Services
void main()
{
    foo();
    crash();
}

void foo()
{
    char sz[256] = "Some String";
    bar();
}

void bar()
{
    do();
}

void crash()
{
    WER();
}
First vs. Second Chance

- **First chance exceptions**
  WinDbg is notified of an exception, you can ignore it

- **Second chance exceptions**
  If the exception wasn’t handled (for example, by a catch block) WinDbg is notified again

- **Relation to crash dumps**
Part 2: Practice Exercises
Links

- **Applications:**
  Not available in preview version

- **Exercise Transcripts:**
  Not available in preview version
Warning

Because this is live debugging, due to differences in actual systems and also ASLR (Address Space Layout Randomization), when you launch applications actual addresses in WinDbg command output may be different from shown in exercise transcripts.
Exercise 0

- **Goal**: Install Debugging Tools for Windows and learn how to set up symbols correctly

- **Memory Analysis Patterns**: Incorrect Stack Trace

- \AWD3\Exercise-0-Download-Setup-WinDbg.pdf
User Mode Debugging

Exercises D1-D8
Exercise D1

- **Goal:** Learn how code generation parameters can influence process execution behavior

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace

- **Debugging Implementation Patterns:** Scope, Variable Value, Type Structure, Code Breakpoint

- \AWD3\Exercise-D1.pdf

© 2018 Software Diagnostics Services
Exercise D2

- **Goal:** Learn how to use hardware breakpoints to catch data corruption

- **Elementary Diagnostics Patterns:** Counter Value

- **Memory Analysis Patterns:** Unloaded Module, Memory Leak (process heap), Abnormal Value (*from trace analysis patterns*)

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Scope, Variable Value, Data Breakpoint

- \AWD3\Exercise-D2.pdf
Exercise D3

- **Goal:** Learn how to navigate parameters, static and local variables, and data structures

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace, Stack Overflow, String Parameter, Module Variable

- **Debugging Implementation Patterns:** Break-in, Scope, Variable Value, Type Structure

- AWD3\Exercise-D3.pdf
Exercise D4

- **Goal:** Learn how to use conditional breakpoints to log behavior

- **Elementary Diagnostics Patterns:** Use-case Deviation

- **Memory Analysis Patterns:** -

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Breakpoint Action

- \AWD3\Exercise-D4.pdf
Exercise D5

- **Goal:** Learn how to debug multiple processes and their deadlock

- **Elementary Diagnostics Patterns:** Crash, Hang

- **Memory Analysis Patterns:** Exception Stack Trace, NULL Data Pointer, Execution Residue, Hidden Exception, Handled Exception, Deadlock

- **Debugging Implementation Patterns:** Break-in

- \AWD3\Exercise-D5.pdf
Expected Behavior

Thread A
- Acquires Mutex A
- Waits for Mutex B
- Acquires Mutex B
- Releases Mutex B
- Releases Mutex A
- Acquires Mutex A

Thread B
- Acquires Mutex B
- Releases Mutex B
- Waits for Mutex A
- Acquires Mutex A
- Releases Mutex A
Deadlock

Thread A

- Acquires Mutex A
- Waits for Mutex B

Thread B

- Acquires Mutex B
- NewFeature()
- Waits for Mutex A
Exercise D6

- **Goal:** Learn how to recognize when we need kernel level debugging

- **Elementary Diagnostics Patterns:** Hang, Counter Value

- **Memory Analysis Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Data Breakpoint, Code Trace

© 2018 Software Diagnostics Services
Exercise D7

- **Goal:** Learn how to manipulate threads to debug race conditions

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace, NULL Code Pointer

- **Debugging Implementation Patterns:** Frozen Thread

- \AWD3\Exercise-D7.pdf
Exercise D8

- **Goal:** Learn how to inspect heap for signs of corruption

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Dynamic Memory Corruption (process heap), Module Variable, Exception Stack Trace

- **Debugging Implementation Patterns:** Break-in

- \AWD3\Exercise-D8.pdf
Kernel Mode Debugging

Exercises KD6, KD9, KD10
Exercise K0

- **Goal:** Setup VMware kernel debugging environment

- \AWD3\Exercise-K0-Kernel-Debugging-Setup.pdf
Space Review (x86)

User Space

Kernel Space

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>ffffffff</td>
<td>driver</td>
</tr>
<tr>
<td>80000000</td>
<td>ffffffff</td>
<td>hal</td>
</tr>
<tr>
<td>7fffffff</td>
<td>ffffffff</td>
<td>ntdll</td>
</tr>
<tr>
<td>80200000</td>
<td>8020a000</td>
<td>BATTC</td>
</tr>
<tr>
<td>8020a000</td>
<td>8020c900</td>
<td>compbatt</td>
</tr>
<tr>
<td>8020d000</td>
<td>80215000</td>
<td>msisadrv</td>
</tr>
<tr>
<td>80215000</td>
<td>8021e000</td>
<td>WMILIB</td>
</tr>
<tr>
<td>8021e000</td>
<td>8022b000</td>
<td>WDFLDR</td>
</tr>
<tr>
<td>8022b000</td>
<td>80266000</td>
<td>CLFS</td>
</tr>
<tr>
<td>80266000</td>
<td>8026e000</td>
<td>BOOTVID</td>
</tr>
<tr>
<td>[...]</td>
<td>81800000</td>
<td>nt</td>
</tr>
<tr>
<td>81ba1000</td>
<td>81bd5000</td>
<td>hal</td>
</tr>
<tr>
<td>[...]</td>
<td>87eb3000</td>
<td>ndiswan</td>
</tr>
<tr>
<td>87ed6000</td>
<td>87ee1000</td>
<td>ndistapi</td>
</tr>
<tr>
<td>87ef8000</td>
<td>87f03000</td>
<td>rsl2tp</td>
</tr>
<tr>
<td>87f03000</td>
<td>87f03000</td>
<td>TDI</td>
</tr>
<tr>
<td>[...]</td>
<td>937b4000</td>
<td>srv</td>
</tr>
<tr>
<td>9446d000</td>
<td>94480000</td>
<td>dump_LSI_SCSI</td>
</tr>
<tr>
<td>96ca1000</td>
<td>96cc9000</td>
<td>fastfat</td>
</tr>
</tbody>
</table>
### Space Review (x64)

#### Kernel Space
- **driver**

#### User Space
- **MyDLL**
- **ntdll**
- **nt**
- **hal**

### Memory Map

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffffd414`7bc00000</td>
<td>ffffd414`7bf90000</td>
<td>win32kfull</td>
</tr>
<tr>
<td>ffffd414`7bf90000</td>
<td>ffffd414`7c1c4000</td>
<td>win32kbase</td>
</tr>
<tr>
<td>ffffd414`7c1d0000</td>
<td>ffffd414`7c20f000</td>
<td>cdd</td>
</tr>
<tr>
<td>ffffd414`7c730000</td>
<td>ffffd414`7c7ac000</td>
<td>win32k</td>
</tr>
<tr>
<td>fffffff802`1cc05000</td>
<td>fffffff802`1d5f2000</td>
<td>nt</td>
</tr>
<tr>
<td>fffffff802`1d800000</td>
<td>fffffff802`1d80b000</td>
<td>kd</td>
</tr>
<tr>
<td>fffffff803`9f200000</td>
<td>fffffff803`9f264000</td>
<td>CLFS</td>
</tr>
<tr>
<td>fffffff803`9f270000</td>
<td>fffffff803`9f294000</td>
<td>tm</td>
</tr>
<tr>
<td>fffffff803`9f2a0000</td>
<td>fffffff803`9f2b7000</td>
<td>PSHED</td>
</tr>
<tr>
<td>fffffff803`9f2c0000</td>
<td>fffffff803`9f2cb000</td>
<td>BOOTVID</td>
</tr>
</tbody>
</table>

[...]
Space Review

Complete stack traces (x64 + x86)
Context Switch

We always see the current process space

User Space

current process A
(Notepad)

Kernel Space

User Space

current process B
(AppD.exe)

Kernel Space

Context switch
Common Commands

- `.logopen <file>`
  Opens a log file to save all subsequent output

- **View commands**
  Dump everything or selected processes and threads (context changes automatically)

- **Switch commands**
  Switch to a specific process or thread
View Commands

- **!process 0 0**
  Lists all processes

- **!process <address> 3f**
  Lists process information including CPU times, environment, modules and its thread stack traces

- **!thread <address> 1f**
  Shows thread information and stack trace
### Switch Commands

- **.process /r /p <address>**
  Switches to a specified process. Its context becomes current. Reloads symbol files for user space. Now we can use commands like `!cs`

  ```
  0: kd> .process /r /p fffffff044d8b30
  Implicit process is now fffffff044d8b30
  Loading User Symbols
  ..................................  
  ```

- **.thread <address>**
  Switches to a specified thread. Assumes the current process context. Now we can use commands like `k*`

- **.thread /r /p <address>**
  The same as the previous command but makes the thread process context current and reloads symbol files for user space:

  ```
  0: kd> .thread /r /p fffffff051b7060
  Implicit thread is now fffffff051b7060
  Implicit process is now fffffff044d8b30
  Loading User Symbols
  .................................
  ```
Exercise KD6

- **Goal:** Learn how to use kernel level debugging to catch corruption caused by a driver or other process

- **Elementary Diagnostics Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Memory Analysis Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Debugging Implementation Patterns:** Code Breakpoint, Data Breakpoint, Code Trace

- `\AWD3\Exercise-KD6.pdf`
Exercise KD9

- **Goal:** Learn how to debug a 32-bit process under x64 Windows

- **Elementary Diagnostics Patterns:** Hang

- **Memory Analysis Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Debugging Implementation Patterns:** Break-in

- \AWD3\Exercise-KD9.pdf
Exercise KD10

- **Goal:** Learn how to debug handle leaks in user and kernel mode debugging

- **Elementary Diagnostics Patterns:** Counter Value

- **Memory Analysis Patterns:** Handle Leak, Historical Information

- **Debugging Implementation Patterns:** Break-in, Usage Trace

- AWD3\Exercise-KD10.pdf

© 2018 Software Diagnostics Services
Managed Debugging

Exercise MD11
Modeling with LINQPad

http://www.linqpad.net/

Download AnyCPU Build (defaults to x64)
User / Managed Space

00000000 00000000

00007fff ffffffff

LINQPad

User Space (PID e364)

CLR

kernel32

ntdll
Process Threads

- `.load <a path to SOS>`: Loads sos WinDbg extension
- `~<n>\s`: Switches between threads
- `k`: Shows unmanaged stack trace
- `!Threads`: Shows managed threads
- `!CLRStack`: Shows managed stack trace

User Space (PID e364)

- LINQPad
- JIT code
- CLR
- kernel32
- ntdll
### Stack Trace Example

<table>
<thead>
<tr>
<th>#</th>
<th>Child-SP</th>
<th>RetAddr</th>
<th>Call Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>000000000007b4fdbc8</td>
<td>00007ffa`fa5633f8</td>
<td>win32u!NtUserWaitMessage+0x14</td>
</tr>
<tr>
<td>01</td>
<td>000000000007b4fdd00</td>
<td>00007ffa`fa4ff452</td>
<td>System_Windows_Forms_ni+0x2d33f8</td>
</tr>
<tr>
<td>02</td>
<td>000000000007b4fdd00</td>
<td>00007ffa`fa4febd2</td>
<td>System_Windows_Forms_ni+0x26f452</td>
</tr>
<tr>
<td>03</td>
<td>000000000007b4fdd70</td>
<td>00007ffa`fa4fe9df</td>
<td>System_Windows_Forms_ni+0x26e9df</td>
</tr>
<tr>
<td>04</td>
<td>000000000007b4fde10</td>
<td>00007ffa`a2c6dbf4</td>
<td>System_Windows_Forms_ni+0x26e9df</td>
</tr>
<tr>
<td>05</td>
<td>000000000007b4fde70</td>
<td>00007ffa`a2c66899</td>
<td>0x00007ffa`a2c6dfb4</td>
</tr>
<tr>
<td>06</td>
<td>000000000007b4fe020</td>
<td>00007ffa`a2c62390</td>
<td>0x00007ffa`a2c66899</td>
</tr>
<tr>
<td>07</td>
<td>000000000007b4fe770</td>
<td>00007ffa`a2c62102</td>
<td>0x00007ffa`a2c62390</td>
</tr>
<tr>
<td>08</td>
<td>000000000007b4fe850</td>
<td>00007ffa`a2c610a5</td>
<td>0x00007ffa`a2c62102</td>
</tr>
<tr>
<td>09</td>
<td>000000000007b4fe890</td>
<td>00007ff`021f6bb3</td>
<td>0x00007ffa`a2c610a5</td>
</tr>
<tr>
<td>0a</td>
<td>000000000007b4feaf0</td>
<td>00007ff`021f6a70</td>
<td>clr!CallDescrWorkerInternal+0x83</td>
</tr>
<tr>
<td>0b</td>
<td>000000000007b4feb30</td>
<td>00007ff`021f735d</td>
<td>clr!CallDescrWorkerWithHandler+0x4e</td>
</tr>
<tr>
<td>0c</td>
<td>000000000007b4feb70</td>
<td>00007ff`0224ec1c</td>
<td>clr!MethodDescCallSite::CallTargetWorker+0x8</td>
</tr>
<tr>
<td>0d</td>
<td>000000000007b4fecc70</td>
<td>00007ff`0224ee06</td>
<td>clr!RunMain+0x1e7</td>
</tr>
<tr>
<td>0e</td>
<td>000000000007b4fee50</td>
<td>00007ff`0224ecfb</td>
<td>clr!Assembly::ExecuteMainMethod+0xb6</td>
</tr>
<tr>
<td>0f</td>
<td>000000000007b4ff140</td>
<td>00007ff`0224eaf4</td>
<td>clr!SystemDomain::ExecuteMainMethod+0x57c</td>
</tr>
<tr>
<td>10</td>
<td>000000000007b4ff750</td>
<td>00007ff`0224ea72</td>
<td>clr!ExecuteEXE+0x3f</td>
</tr>
<tr>
<td>11</td>
<td>000000000007b4ff7c0</td>
<td>00007ff`0224e3f4</td>
<td>clr! CorExeMainInternal+0xb2</td>
</tr>
<tr>
<td>12</td>
<td>000000000007b4ff850</td>
<td>00007ff`27ef7b2d</td>
<td>clr! CorExeMain+0x14</td>
</tr>
<tr>
<td>13</td>
<td>000000000007b4ff890</td>
<td>00007ff`27f9a4cc</td>
<td>mscoreei! CorExeMain+0x12</td>
</tr>
<tr>
<td>14</td>
<td>000000000007b4ff8f0</td>
<td>00007ff`2bd3034</td>
<td>mscoreei! CorExeMain_Exported+0x6c</td>
</tr>
<tr>
<td>15</td>
<td>000000000007b4ff920</td>
<td>00007ff`2dcb1431</td>
<td>kernel32! BaseThreadInitThunk+0x14</td>
</tr>
<tr>
<td>16</td>
<td>000000000007b4ff950</td>
<td>0000000000000000</td>
<td>ntdll!RtlUserThreadStart+0x21</td>
</tr>
</tbody>
</table>
Examining JIT Code

0:000> !IP2MD 0x00007ffa\a2c6dfb4
MethodDesc: 00007ffaa2ce42e0
Method Name: LINQPad.Program.Run(System.String, Boolean, System.String, Boolean, Boolean, Boolean, Boolean, System.String)
Class: 00007ffaa2c55730
MethodTable: 00007ffaa2ce43d8
mdToken: 00000000060005b2
Module: 00007ffaa2ae4118
IsJitted: yes
CodeAddr: 00007ffaa2c6d770
Transparency: Critical

0:000> !IP2MD System_Windows_Forms_ni+0x26e9df
MethodDesc: 00007ffafa2c9af8
Class: 00007ffafa2c55730
MethodTable: 00007ffafa2c9af8
mdToken: 0000000006005374
Module: 00007ffafa2c9af8
IsJitted: yes
CodeAddr: 00007ffafa2c6d770
Transparency: Safe critical
Exercise MD11

- **Goal:** Learn how to find problem modules, classes and methods, disassemble code, dump object references, recognize and analyze deadlocks using SOS(EX) extensions

- **Elementary Diagnostics Patterns:** Crash, Hang

- **Memory Analysis Patterns:** Exception Stack Trace, Handled Exception, NULL Data Pointer, Nested Exception, CLR Thread, JIT Code, Deadlock, Execution Residue

- **Debugging Implementation Patterns:** Break-in

- `\AWD3\Exercise-MD11.pdf`
Expected Behavior

Thread 1
- Enter cs1
- Exit cs1
- Enter cs2
- Exit cs2

Thread 2
- Enter cs2
- Waits for cs1
- Enter cs1
- Exit cs1
- Exit cs2
Deadlock

Thread #14 (owns) Object 000001e398bcd680

Thread #18 (waiting)

Thread #14 (waiting) Object 000001e398bcd6c0

Thread #18 (owns)
Postmortem Debugging

- Memory dump collection
  - .dump / .dumpcab WinDbg commands
  - External methods

- Training
  - Accelerated Windows Memory Dump Analysis
  - Advanced Windows Memory Dump Analysis
  - Accelerated .NET Memory Dump Analysis
Analysis Pattern Links

Spiking Thread
Heap Corruption
Execution Residue
Invalid Pointer
Managed Stack Trace
NULL Code Pointer
Stack Trace Collection
Deadlock (objects)
Message Box
False Function Parameters
Handle Leak
Abnormal Value
Module Variable
Handled Exception
JIT Code

CLR Thread
Exception Stack Trace
Hidden Exception
Managed Code Exception
NULL Data Pointer
Stack Trace
Stack Overflow
Virtualized Process
Memory Leak
String Parameter
Historical Information
Counter Value
Deadlock (managed)
Nested Exception
Resources

- WinDbg Help / WinDbg.org (quick links) / DumpAnalysis.org
- Debugging.TV / DebuggingTV YouTube Channel
- Practical Foundations of Windows Debugging, Disassembling, Reversing
- Windows Debugging Notebook: Essential User Space WinDbg Commands
- Software Diagnostics Library
- Pattern-Driven Software Problem Solving
- Memory Dump Analysis Anthology (Volumes 1 – 10)

© 2018 Software Diagnostics Services
Q&A

Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!