Prerequisites

- Debugging in Visual Studio

or

- Basic crash dump analysis
Why WinDbg/WinDbg Classic?

- Easy to install (WinDbg)
- Production debugging
- Redistributable (WinDbg Classic)
- Kernel mode debugging
- Time Travel Debugging (WinDbg)
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](http://www.debugging.tv) (more than 40 episodes)
- PDB Symbols: episodes 0x01 – 0x04
- Kernel debugging setup: episode 0x25
Schedule Summary

Day 1
- Debugging Fundamentals and x64 Disassembly Review (1 hour)
- User Mode Debugging (30 minutes)

Day 2/3
- User Mode Debugging (3 hours)

Day 4
- Kernel Debugging (1 hour 30 minutes)

Day 5
- Managed Debugging (30 minutes)
- Time-Travel Debugging (30 minutes)
- Rust Debugging (30 minutes)
Part 1: Fundamentals
Memory Space³
Execution Mode³
Code³
Live Debugging Technique

- Breakpoints
- Inspection
- Tracing
Pattern$^3$
Debugging Paradigm³
Debugging Paradigm

Time Travel Debugging

Live

Dumps

Logs
Memory Spacetime
Debugging Paradigm

Idea: Kaluza-Klein Theory of a microscopic 5th dimension
Pattern Mapping

- Software Incident
- Elementary Diagnostics
- Software Diagnostics
- Memory Analysis
- Debugging
- Debugging Implementation
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 catalog

- Software Trace and Log Analysis catalog
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)
Full Debugging Patterns Catalog

Python Debugging for AI, Machine Learning, and Cloud Computing
A Pattern-Oriented Approach
Dmitry Vostokov
<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>09F00000</td>
<td>App</td>
</tr>
<tr>
<td>0F530000</td>
<td>0F5F0000</td>
<td>ntdll</td>
</tr>
<tr>
<td>638D0000</td>
<td>63B2D0000</td>
<td>CoreUIComponents</td>
</tr>
<tr>
<td>63BB0000</td>
<td>64350000</td>
<td>CoreMessaging</td>
</tr>
<tr>
<td>642D0000</td>
<td>64435000</td>
<td>twinapi_appcore</td>
</tr>
<tr>
<td>6BE0000</td>
<td>6C0B4000</td>
<td>comctl32</td>
</tr>
<tr>
<td>6D7E0000</td>
<td>6D930000</td>
<td>dwapi</td>
</tr>
<tr>
<td>6D930000</td>
<td>6DA960000</td>
<td>uitheme</td>
</tr>
<tr>
<td>6DA96000</td>
<td>6F530000</td>
<td>WinTypes</td>
</tr>
<tr>
<td>6DAC0000</td>
<td>6DCE8000</td>
<td>iertutil</td>
</tr>
<tr>
<td>6DCF0000</td>
<td>6D803000</td>
<td>urlTypes</td>
</tr>
<tr>
<td>6D82F000</td>
<td>6E530000</td>
<td>propsys</td>
</tr>
<tr>
<td>6E08C000</td>
<td>6F530000</td>
<td>winspool</td>
</tr>
<tr>
<td>6E0F6000</td>
<td>6F530000</td>
<td>oleacc</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>mpr</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>bcrypt</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>IPHLPAPI</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>CRYPTBASE</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>spicil1</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>gdi32</td>
</tr>
<tr>
<td>6E1A8000</td>
<td>6F530000</td>
<td>msvcrct</td>
</tr>
<tr>
<td>6F4D0000</td>
<td>6F530000</td>
<td>user32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>KERNELBASE</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>kernel32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>clbcatq</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>msctf</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>shell32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>bcryptPrimitives</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>ucrtbase</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>rpctr4</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>win32u</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>cfgmgr32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>fliLib</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>powrprof</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>shlwapi</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>oleaut32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>profapi</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>sechost</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>gdi32full</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>imm32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>comdlg32</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>windows_storage</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>msvcp_win</td>
</tr>
<tr>
<td>6F530000</td>
<td>6F530000</td>
<td>ntdll</td>
</tr>
</tbody>
</table>
Thread Stack Trace

User Stack for TID 102

Return address Module!FunctionC+130

Return address Module!FunctionB+220

Return address Module!FunctionA+110

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

Module!FunctionA

Resumes from address Module!FunctionA+110
Saves return address Module!FunctionA+110

Module!FunctionB

Resumes from address Module!FunctionB+220
Saves return address Module!FunctionB+220

Module!FunctionC

Resumes from address Module!FunctionC+130
Saves return address Module!FunctionC+130

Module!FunctionD

© 2024 Software Diagnostics Services
Thread Stack Trace (no PDB)

User Stack for TID 102

Return address Module+43130

Return address Module+32220

Return address Module+22110

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

© 2024 Software Diagnostics Services
Thread Raw Stack Data

```c
void main()
{
    foo();
    crash();
}

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

void bar()
{
    do();
}

void crash()
{
    WER();
}
```

0:000> k
module!crash+30
module!main+10
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: x64 Disassembly
**x64 CPU Registers**

- **RAX → EAX → AX → {AH, AL}**
- **ALU:** RAX, RDX
- **Counter:** RCX
- **Memory copy:** RSI (src), RDI (dst)
- **Stack:** RSP
- **Next instruction:** RIP
- **New:** R8 – R15, Rx(D|W|B)

RAX 64-bit

EAX 32-bit
Instructions and Registers

- **Opcodes** DST, SRC

- **Examples:**

  - `mov rax, 10h`
    - `RAX ← 0x10`
  - `mov r13, rdx`
    - `R13 ← RDX`
  - `add r10, 10h`
    - `R10 ← R10 + 0x10`
  - `imul edx, ecx`
    - `EDX ← EDX * ECX`
  - `call rdx`
    - `RDX already contains the address of func (&func)`
    - `PUSH RIP; &func → RIP`
  - `sub rsp, 30h`
    - `RSP ← RSP-0x30`
    - `make room for local variables`
Memory and Stack Addressing

Stack grows

Lower addresses

Higher addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSP-0x20</td>
<td>[RSP-0x20]</td>
</tr>
<tr>
<td>RSP-0x18</td>
<td>[RSP-0x18]</td>
</tr>
<tr>
<td>RSP-0x10</td>
<td>[RSP-0x10]</td>
</tr>
<tr>
<td>RSP-0x8</td>
<td>[RSP-0x8]</td>
</tr>
<tr>
<td>RSP</td>
<td>[RSP]</td>
</tr>
<tr>
<td>RSP+0x8</td>
<td>[RSP+0x8]</td>
</tr>
<tr>
<td>RSP+0x10</td>
<td>[RSP+0x10]</td>
</tr>
<tr>
<td>RSP+0x18</td>
<td>[RSP+0x18]</td>
</tr>
<tr>
<td>RSP+0x20</td>
<td>[RSP+0x20]</td>
</tr>
</tbody>
</table>
Memory Cell Sizes

- RSP → BYTE PTR [RSP]
- RSP → DWORD PTR [RSP]
- RSP → QWORD PTR [RSP]
Memory Load Instructions

- **Opcode DST, PTR [SRC+Offset]**
- **Opcode DST**

**Examples:**

```plaintext
mov  rax, qword ptr [rsp+10h] ; RAX ←
     ; 64-bit value at address RSP+0x10
mov  ecx, dword ptr [20]      ; ECX ←
     ; 32-bit value at address 0x20
pop  rdi                      ; RDI ← value at address RSP
     ; RSP ← RSP + 8
lea  r8, [rsp+20h]            ; R8 ← address RSP+0x20
```

© 2024 Software Diagnostics Services
Memory Store Instructions

- **Opcode** `PTR [DST+Offset], SRC`
- **Opcode** `DST | SRC`

**Examples:**

```assembly
mov qword ptr [rbp-20h], rcx ; 64-bit value at address RBP-0x20 ← RCX
mov byte ptr [0], 1 ; 8-bit value at address 0 ← 1
push rsi ; RSP ← RSP - 8
inc dword ptr [rcx] ; 32-bit value at address RCX ← 1 + 32-bit value at address RCX
```
Flow Instructions

- Opcode DST

- Opcode PTR [DST]

Examples:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>00007ff6`9ef2f008</td>
<td>RIP ← 0x7ff69ef2f008; (goto 0x7ff69ef2f008)</td>
</tr>
<tr>
<td>jmp</td>
<td>qword ptr [rax+10h]</td>
<td>RIP ← value at address RAX+0x10</td>
</tr>
<tr>
<td>call</td>
<td>00007ff6`9ef21400</td>
<td>RSP ← RSP – 8</td>
</tr>
<tr>
<td></td>
<td>00007ff6`9ef21057:</td>
<td>value at address RSP ← 0x7ff69ef21057</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RIP ← 0x7ff69ef21400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(goto 0x7ff69ef21400)</td>
</tr>
</tbody>
</table>
Windows API Parameters

- x86: Right to left **PUSH**

  Args to Child are parameters

- x64: Left to right **RCX, RDX, R8, R9, stack**

  Args to Child are **not** parameters

---

**WinDbg Commands**

```
0:000> kv
  # Child-SP   RetAddr     : Args to Child     : Call Site
  ...
```
Part 3: Practice Exercises
Links

- **Applications:**
  Download links are in exercises UD0 and KD0.

- **Exercise Transcripts:**
  Included in this book.
Warning

Because of live debugging, due to differences in actual systems and ASLR (Address Space Layout Randomization), when you launch applications, actual addresses and even the number and order of threads in WinDbg command output may differ from those shown in exercise transcripts.
Exercise UDO

- **Goal:** Download and verify your WinDbg installation
- **Memory Analysis Patterns:** Stack Trace; Incorrect Stack Trace
- \AWD4\Exercise-UD0-Download-WinDbg.pdf
User Mode Debugging

Exercises UD1 – UD8
Exercise UD1

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

- **Elementary Diagnostics Patterns:** Error Message or Crash

- **Memory Analysis Patterns:** Message Box or Exception Stack Trace; Constant Subtrace

- **Debugging Implementation Patterns:** Break-in; Scope; Variable Value; Type Structure; Code Breakpoint

- AWD4\Exercise-UD1.pdf
Exercise UD2

- **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); Corrupt Structure; Abnormal Value (*from trace analysis patterns*)

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

- AWD4\Exercise-UD2.pdf
Exercise UD3

- **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 (User Mode); String Parameter; Module Variable

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

- \AWD4\Exercise-UD3.pdf
Exercise UD4

- **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

- \AWD4\Exercise-UD4.pdf
Exercise UD5

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

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

- **Memory Analysis Patterns:** Exception Stack Trace; Constant Subtrace; NULL Pointer (Data); Main Thread; Execution Residue (Unmanaged Space, User); Hidden Exception (User Space); Handled Exception (User Space); Wait Chain (Mutex Objects); Deadlock (Objects, User Space)

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

- AWD4\Exercise-UD5.pdf

© 2024 Software Diagnostics Services
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
- Releases Mutex A
Deadlock

Thread A
- Acquires Mutex A
- Waits for Mutex B

Thread B
- Acquires Mutex B
- NewFeature()
- Waits for Mutex A
Exercise UD6

- **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

- \AWD4\Exercise-UD6.pdf
Exercise UD7

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

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace; NULL Pointer (Data)

- **Debugging Implementation Patterns:** Frozen Thread

- \AWD4\Exercise-UD7.pdf
Exercise UD8

- **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

- \AWD4\Exercise-UD8.pdf
Kernel Mode Debugging

Exercises KD6, KD9, KD10
Exercise KD0

- **Goal:** Set up Hyper-V or VMware kernel debugging environment

- \AWD4\Exercise-KD0-Kernel-Debugging-Setup.pdf
### Space Review (x86)

#### Kernel Space

- **Driver**
- **nt**
- **hal**

#### User Space

- **ntdll**
- **nt**
- **MyDLL**
- **App**

---

Here is a table showing the start and end addresses of various modules:

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<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></td>
<td></td>
</tr>
<tr>
<td>81800000</td>
<td>81ba1000</td>
<td>nt</td>
</tr>
<tr>
<td>81ba1000</td>
<td>81bd5000</td>
<td>hal</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87eb3000</td>
<td>87ed6000</td>
<td>ndiswan</td>
</tr>
<tr>
<td>87ed6000</td>
<td>87ee1000</td>
<td>ndistapi</td>
</tr>
<tr>
<td>87ee1000</td>
<td>87ef8000</td>
<td>rasl2tp</td>
</tr>
<tr>
<td>87ef8000</td>
<td>87f03000</td>
<td>TDI</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>937b4000</td>
<td>93800000</td>
<td>srv</td>
</tr>
<tr>
<td>9446d000</td>
<td>9448d000</td>
<td>dump_LSI_SCSI</td>
</tr>
<tr>
<td>96ca1000</td>
<td>96cc9000</td>
<td>fastfat</td>
</tr>
</tbody>
</table>

---

© 2024 Software Diagnostics Services
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)

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 fffffa80044d8b30
  Implicit process is now fffffa80`044d8b30
  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 fffffa80051b7060
  Implicit thread is now fffffa80`051b7060
  Implicit process is now fffffa80`044d8b30
  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:** Hang; Counter Value

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

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

\AWD4\Exercise-KD6.pdf
Exercise KD9

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

- **Elementary Diagnostics Patterns**: Hang

- **Memory Analysis Patterns**: Virtualized Process; Debugger Bug; Execution Residue (Unmanaged Space, User); Rough Stack Trace (Unmanaged Space); Message Box; String Parameter; Near Exception

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

- \AWD4\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

- `\AWD4\Exercise-KD10.pdf`
Managed Debugging

Exercise MD11
Modeling with LINQPad

```csharp
// AppMD11.linq
// Copyright (c) 2013 - 2018 Software Diagnostics

void Main()
{
    new ClassMain().Main();
}

public class ClassMain
{
    static string cs1 = "critical section 1";
    static string cs2 = "critical section 2";

    static void DoWork()
    {
        Thread.Sleep(100);
    }

    static void thread_proc_1()
    {
        Monitor.Enter(cs1);
    }
}

http://www.linqpad.net/
```
User / Managed Space

LINQPad8

User Space (PID e364)

coreclr

kernel32

ntdll

00000000  00000000

00007fff ffffffff
Process Threads

~<n>s
Switches between threads

k
Shows unmanaged stack trace

!Threads
Shows managed threads

!CLRStack
Shows managed stack trace
### 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>00000029<code>6e3add78 00007ffa</code>7eb90458</td>
<td>win32u!NtUserWaitMessage+0x14</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00000029<code>6e3add80 00007ffa</code>d7244d61 0x00007ffa`7eb90458</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>00000029<code>6e3ade40 00007ffa</code>d7247486 System_Windows_Forms!...+0x581</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>00000029<code>6e3adf50 00007ffa</code>d7247026 System_Windows_Forms!...+0x416</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>00000029<code>6e3ae000 00007ffa</code>d6ef212b System_Windows_Forms!...+0x46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>00000029<code>6e3ae060 00007ffa</code>7cb021e4 System_Windows_Forms!...+0x3b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>00000029<code>6e3ae0a0 00007ffa</code>7cae2534 0x00007ffa`7cb021e4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>00000029<code>6e3ae350 00007ffa</code>7c70a16d 0x00007ffa`7cae2534</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>00000029<code>6e3ae680 00007ffa</code>7c7015c2 0x00007ffa`7c70a16d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>00000029<code>6e3ae6e0 00007ffa</code>dc226ce3 0x00007ffa`7c7015c2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0a</td>
<td>00000029<code>6e3ae7b0 00007ffa</code>dc1bcd42 coreclr!CallDescrWorkerInternal+0x83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b</td>
<td>(Inline Function) --------~-------- coreclr!CallDescrWorkerWithHandler+0x57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0c</td>
<td>00000029<code>6e3ae7f0 00007ffa</code>dc1c3c09 coreclr!MethodDescCallSite::...+0x196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0d</td>
<td>(Inline Function) --------~-------- coreclr!MethodDescCallSite::Call+0xb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0e</td>
<td>00000029<code>6e3ae930 00007ffa</code>dc1c4097 coreclr!RunMain+0x1f5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0f</td>
<td>00000029<code>6e3aeb10 00007ffa</code>dc1c4841 coreclr!Assembly::ExecuteMainMethod+0x1cb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00000029<code>6e3aeaa0 00007ffa</code>dc0e21c1 coreclr!CorHost2::ExecuteAssembly+0x221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00000029<code>6e3af030 00007ffa</code>2ef54e2d coreclr!coreclr_execute_assembly+0x101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>00000029`6e3af0d0 00007ffbf2ef62e27 hostpolicy!coreclr_t::execute_assembly+0x2d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>00000029`6e3af120 00007ffbf2ef62a36 hostpolicy!run_app_for_context+0x387</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>00000029`6e3af280 00007ffbf2ef64262 hostpolicy!run_app+0x46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>00000029`6e3af2d0 00007ffbf36ff3a7e hostpolicy!corehost_main+0x132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>00000029`6e3af480 00007ffbf36ff72d8 hostfxr!execute_app+0x1de</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>(Inline Function) --------~-------- hostfxr!?A0xd51c85dd::read_config...+0x10a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>00000029`6e3af570 00007ffbf36ff5b5b hostfxr!fx_muxer_t::handle_exec...+0x214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>00000029`6e3af660 00007ffbf36ff2109 hostfxr!fx_muxer_t::execute+0x39b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1a</td>
<td>00000029`6e3af7a0 00007ffbf6ea712361 hostfxr!hostfxr_main_startupinfo+0x89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1b</td>
<td>00000029`6e3af8a0 00007ffbf6ea712758 LINQPad6!exe_start+0x651</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c</td>
<td>00000029`6e3af900 00007ffbf6ea714608 LINQPad6!wmain+0x88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1d</td>
<td>(Inline Function) --------~-------- LINQPad6!invoke_main+0x22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1e</td>
<td>00000029`6e3afb00 00007ffbf58387034 LINQPad6!_scrt_common_main_seh+0x10c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1f</td>
<td>00000029`6e3afb40 00007ffbf5a002651 kernel32!BaseThreadInitThunk+0x14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>00000029<code>6e3afb70 00000000</code>0000000 ntdll!RtlUserThreadStart+0x21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examining JIT Code

0:000> !IP2MD 0x0007fa7c7ae2534
MethodDesc: 00007fa7c7f5aa0
Method Name: LINQPad.UIProgram.Go(System.String[])
Class: 00007fa7c803a28
MethodTable: 00007fa7c7f7078
mdToken: 00000000060001A4
Module: 00007fa7c79f798
IsJitted: yes
Current CodeAddr: 00007fa7cae1b90
Version History:
  ILCodeVersion: 0000000000000000
  ReJIT ID: 0
  IL Addr: 0000000000000000
    CodeAddr: 00007fa7cae1b90 (MinOptJitted)
    NativeCodeVersion: 0000000000000000

0:000> !DumpModule 00007fa7c79f798
Name: C:\Program Files\LINQPad6\LINQPad.GUI.dll
Attributes: PFile SupportsUpdateableMethods
Assembly: 00000177c5cdece0
BaseAddress: 00000177DFDE0000
PFile: 00000177C5CDE8E0
ModuleId: 00007FFA7C7D1860
ModuleIndex: 0000000000000001
LoaderHeap: 0000000000000000
TypeDefToMethodTableMap: 00007FFA7C7A0020
TypeDefToMethodTableMap: 00007FFA7C7A1860
MethodDefToDescMap: 00007FFA7C7A4138
FieldDefToDescMap: 00007FFA7C7A4138
MemberRefToDescMap: 0000000000000000
FileReferencesMap: 00007FFA7C7B7F80
AssemblyReferencesMap: 00007FFA7C7B7F80
MetaData start address: 00000177DFE887DC (719936 bytes)
Exercise MD11

- **Goal:** Learn how to find problem modules, classes and methods, disassemble code, dump object references, recognize and analyze deadlocks

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

- **Memory Analysis Patterns:** Exception Stack Trace; NULL Pointer (Data); CLR Thread; JIT Code; Deadlock (Managed Space); Execution Residue (Managed Space); Hidden Exception (Managed Space)

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

\AWD4\Exercise-MD11.pdf

© 2024 Software Diagnostics Services
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 #15
حواية

Thread #15
الحوية

Thread #16
الحوية

Thread #16
الحوية

Thread #16
الحوية

Thread #16
الحوية
Time Travel Debugging

Exercise TD5
Exercise TD5

**Goal:** Learn how to find hidden exceptions using Time Travel Debugging

**Elementary Diagnostics Patterns:** Hang

**Memory Analysis Patterns:** Hidden Exception (User Space)

**Debugging Implementation Patterns:** Instruction Trace

\AWD4\Exercise-TD5.pdf
Rust Debugging

Exercise RD12
Exercise RD12

- **Goal:** Learn how WinDbg can be used to debug Rust applications

- **Elementary Diagnostics Patterns:** Error Message

- **Memory Analysis Patterns:** Stack Trace

- **Debugging Implementation Patterns:** Break-in; Code Breakpoint; Scope; Variable Value; Type Structure

- \AWD4\Exercise-RD12.pdf
Postmortem Debugging

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

- Training
  - Accelerated Windows Memory Dump Analysis
  - Advanced Windows Memory Dump Analysis
  - Accelerated .NET Core Memory Dump Analysis
  - Accelerated Rust Windows Memory Dump Analysis

© 2024 Software Diagnostics Services
Additional Training for Debugging

- **Accelerated Windows API for Software Diagnostics**
- **Accelerated C & C++ for Windows Diagnostics**
- **Accelerated Disassembly, Reconstruction and Reversing**
- **Memory Thinking for Rust**
Analysis Pattern Links

Spiking Thread
Debugger Bug
Hidden Exception (User Space)
Execution Residue (Managed Space)
Managed Stack Trace
NULL Pointer (Code)
Stack Overflow (User Mode)
Deadlock (Objects, User Space)
Message Box
String Parameter
Handle Leak
Abnormal Value
Module Variable
JIT Code
Unloaded Module
Main Thread
Near Exception

CLR Thread
Exception Stack Trace
Handled Exception (User Space)
Managed Code Exception
NULL Pointer (Data)
Dynamic Memory Corruption (Process Heap)
Rough Stack Trace (Unmanaged Space)
Virtualized Process (WOW64)
Memory Leak (Process Heap)
Wait Chain (Mutex Objects)
Historical Information
Counter Value
Deadlock (Managed Space)
Hidden Exception (Managed Space)
Corrupt Structure
Execution Residue (Unmanaged Space, User)
Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Practical Foundations of Windows Debugging, Disassembling, Reversing, Second Edition
- Windows Debugging Notebook: Essential User Space WinDbg Commands
- Software Diagnostics Library
- Pattern-Driven Software Problem Solving
- Memory Dump Analysis Anthology (Diagnomicon)
Q&A

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