.NET Core Memory Dump Analysis Accelerated

Revised Version

Dmitry Vostokov
Software Diagnostics Services

Prerequisites

WinDbg Commands

We use these boxes to introduce some WinDbg commands used in practice exercises

Basic .NET Core programming and debugging

Training Goals

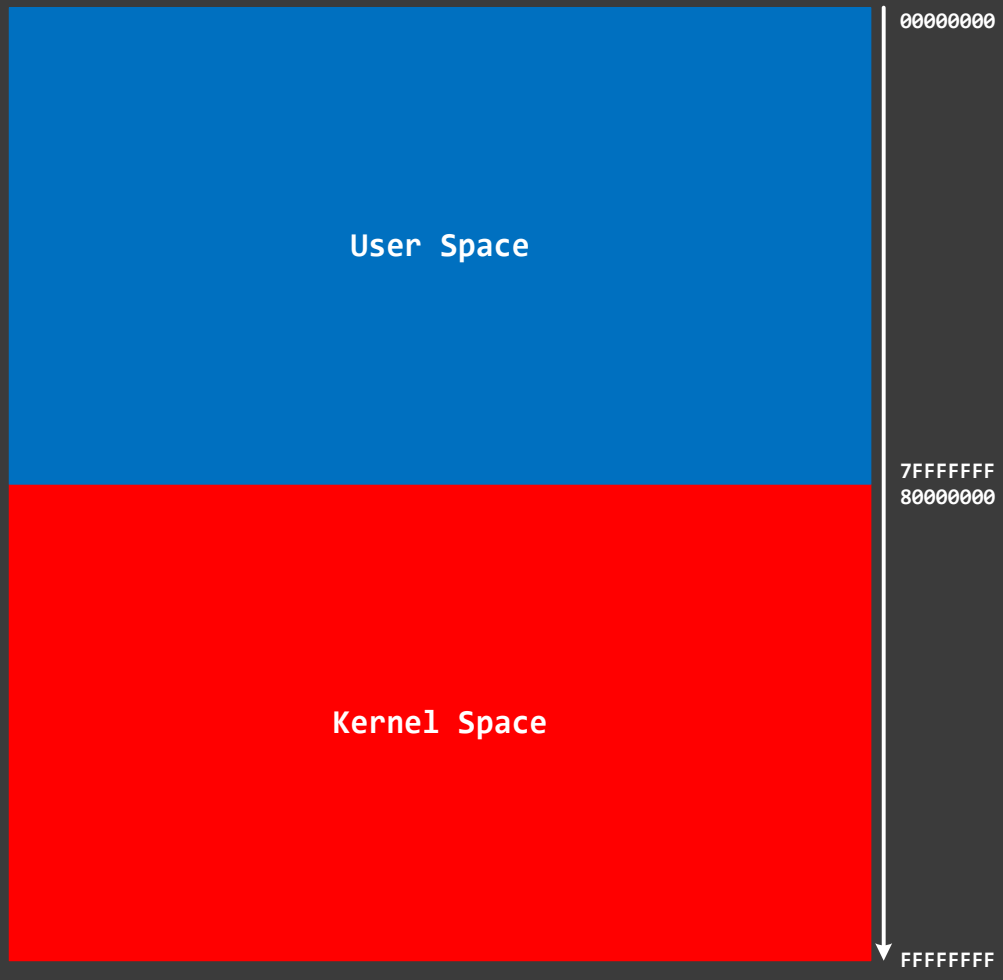
- Review fundamentals
- Learn how to analyze process dumps
- Learn necessary commands in context
- Cover CoreCLR x64

Training Principles

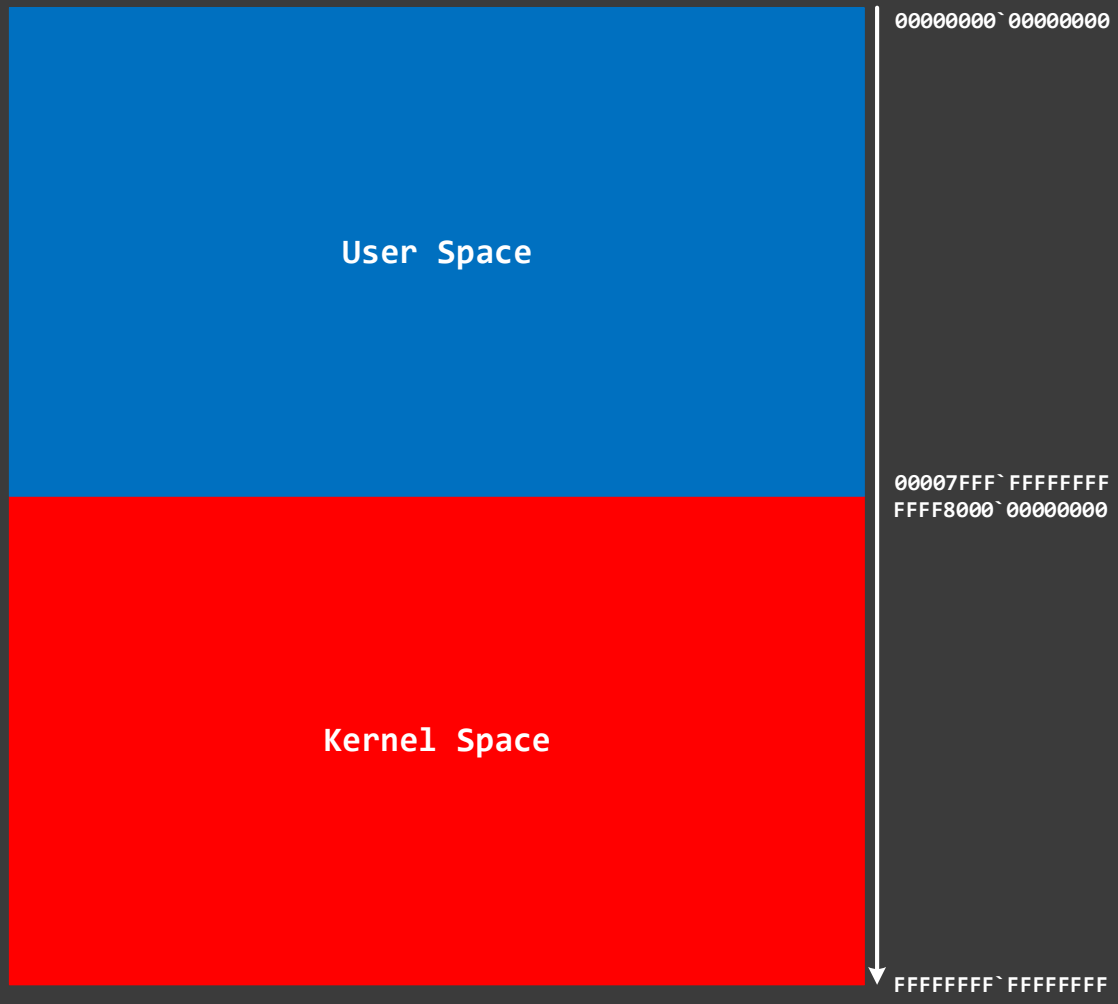
- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content

Part 1: Fundamentals

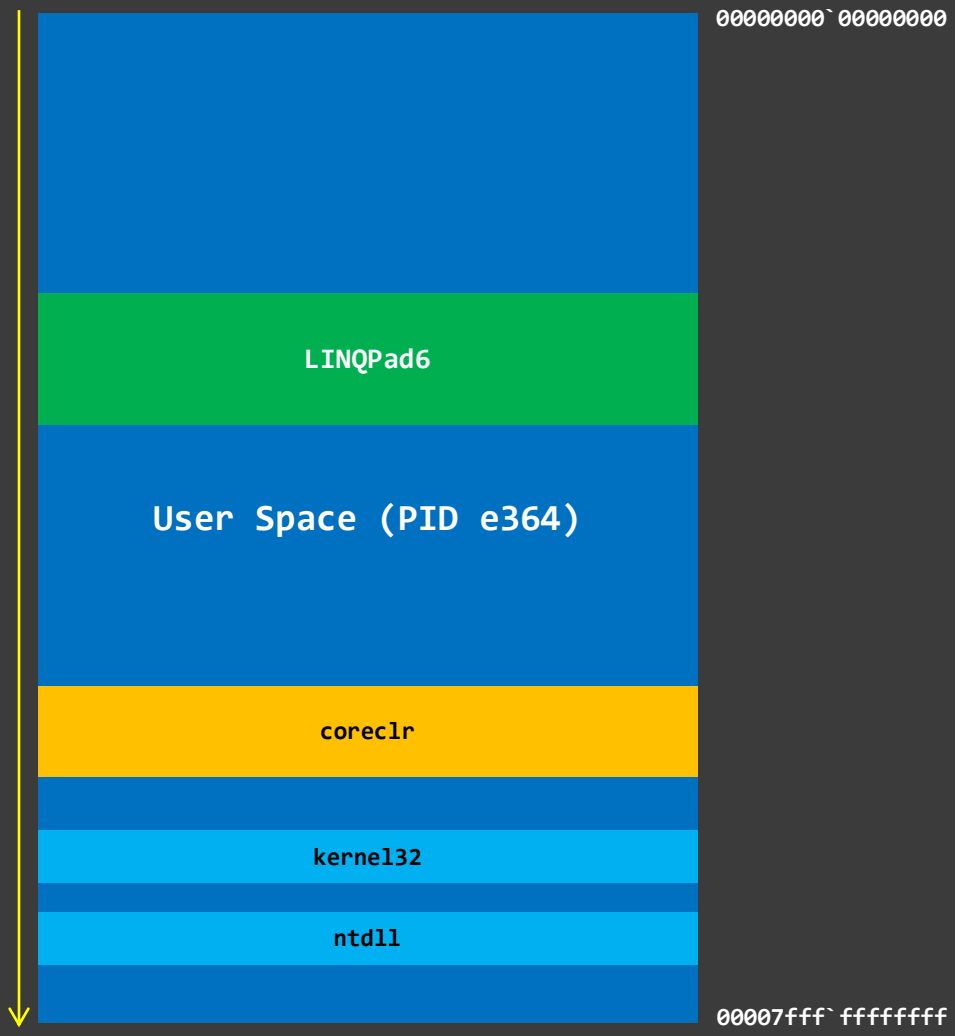
Memory Space (x86)



Memory Space (x64)



User/Managed Space



WinDbg Commands

lmv command lists all loaded modules (EXE and DLLs)

Types/Assemblies/Modules

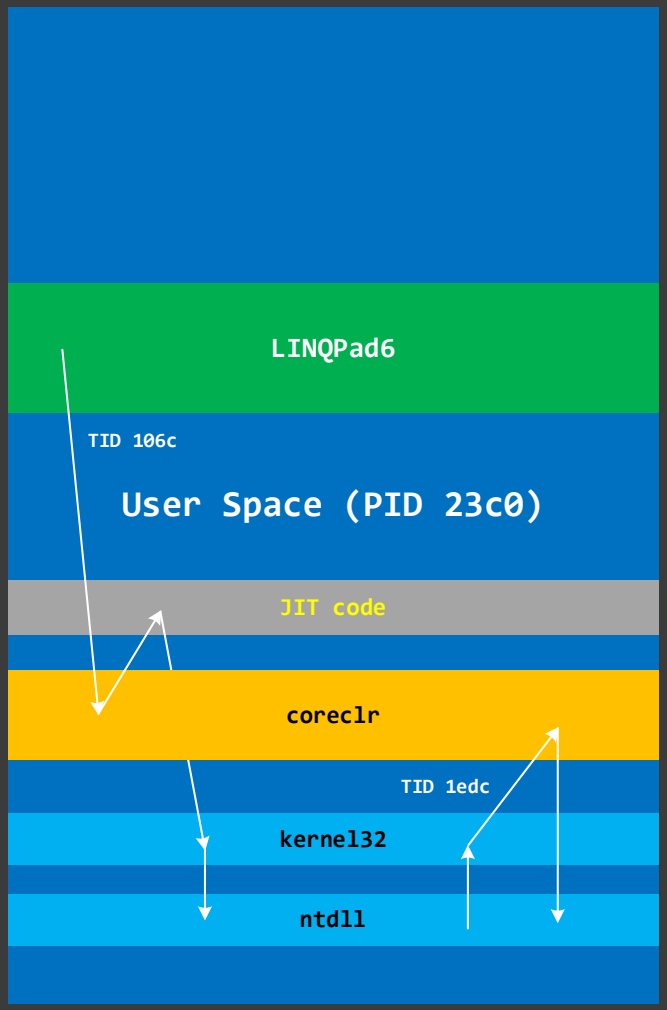
WinDbg Commands

lmv command lists all loaded modules (EXE and DLLs)

!IP2MD command shows type method and module address

!DumpModule command shows module name

Process Threads



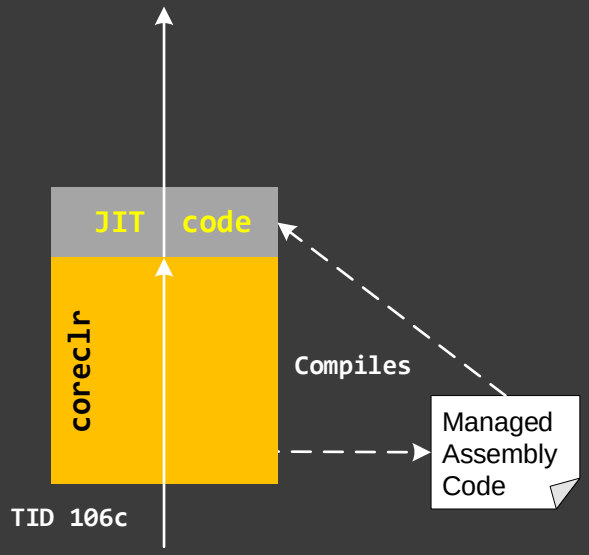
WinDbg Commands

~<n>s command switches between threads

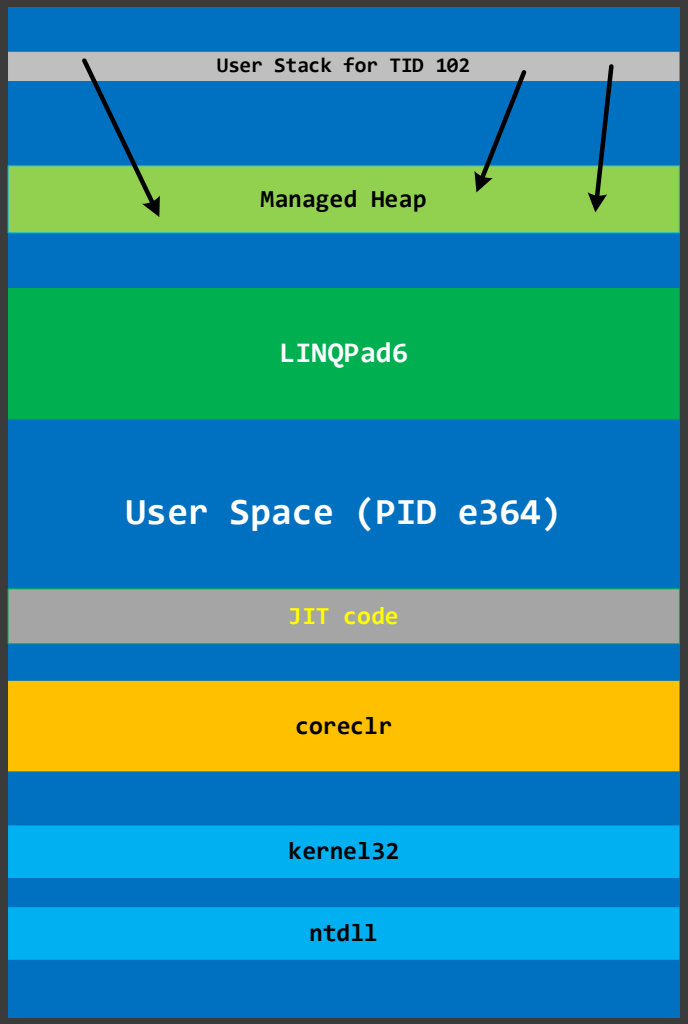
k command shows unmanaged stack trace

!Threads command shows managed threads

!CLRStack command shows managed stack trace



Thread Stack Raw Data



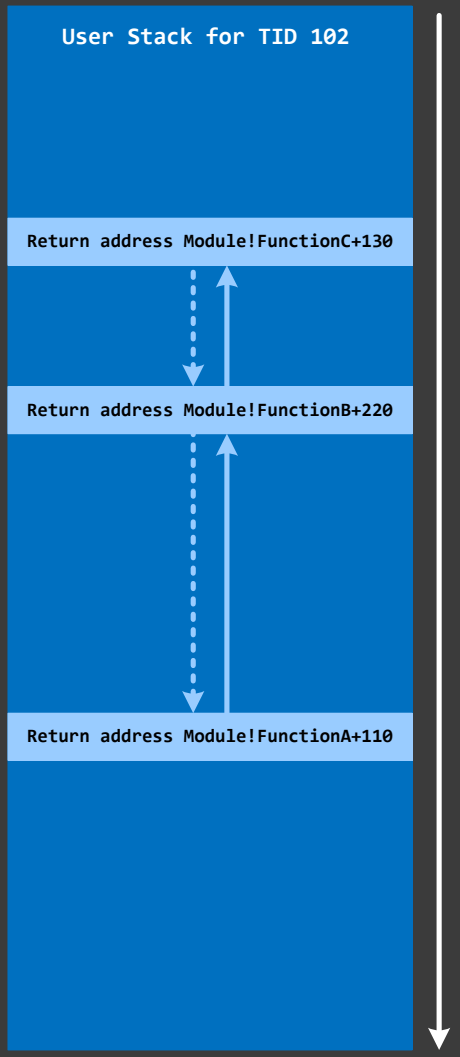
WinDbg Commands

Get stack range:
!teb

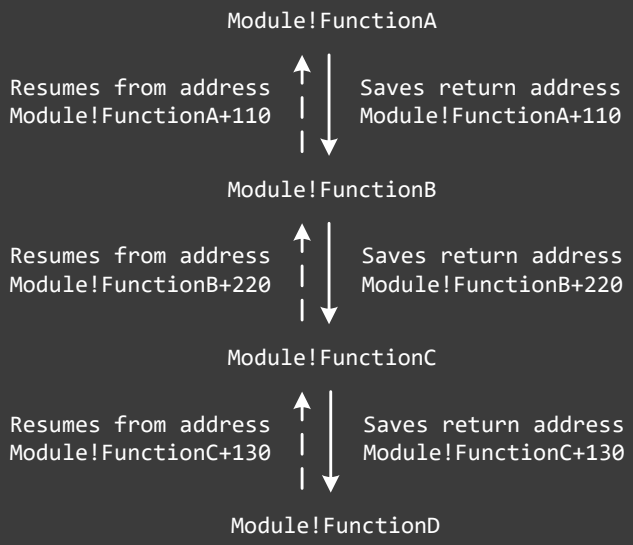
Dump raw data:
dc / dps / dpp / dpa / dpu

Dump managed references:
!DumpStackObjects

Thread Stack Trace



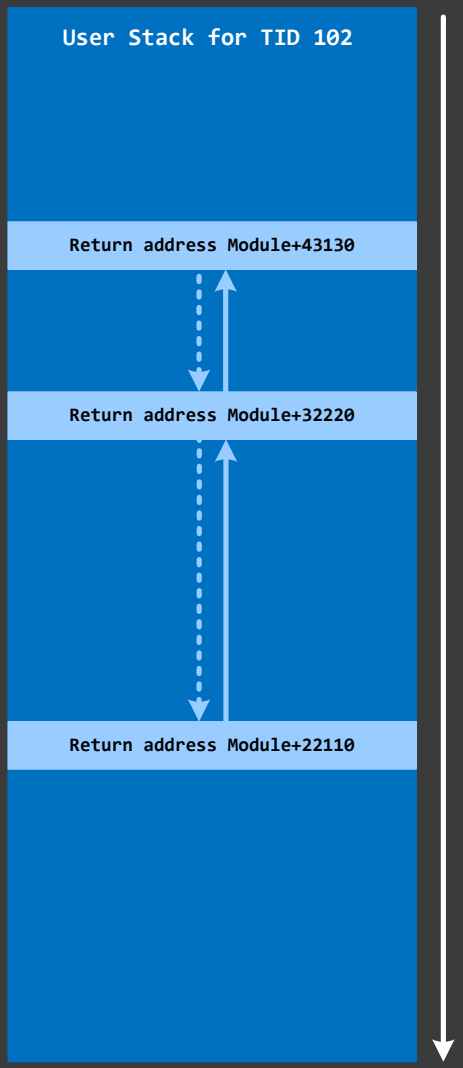

```
FunctionA()
{
  ...
  FunctionB();
  ...
}

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

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



WinDbg Commands

```
0:000> k
Module!FunctionD
Module!FunctionC+130
Module!FunctionB+220
Module!FunctionA+110
```

Thread Stack Trace (no PDB)

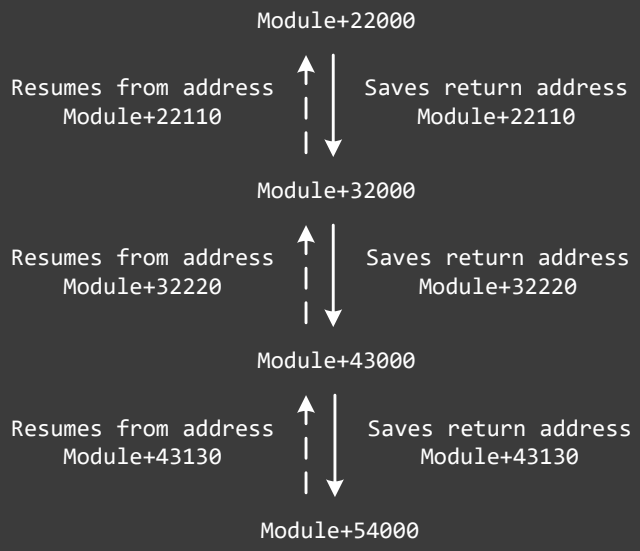



```
FunctionA()
{
  ...
  FunctionB();
  ...
}

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

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

Symbol file Module.pdb

FunctionA 22000 - 23000
FunctionB 32000 - 33000
FunctionC 43000 – 44000
FunctionD 54000 - 55000

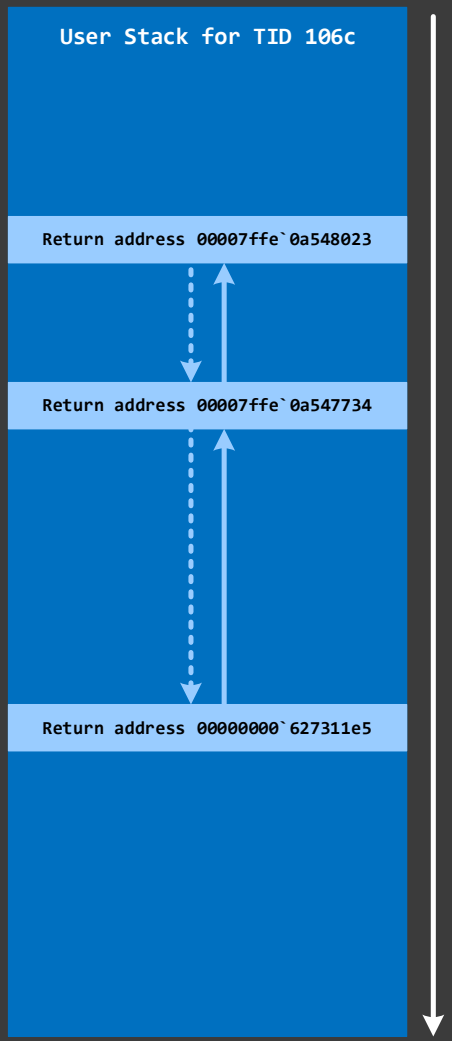
No symbols for Module



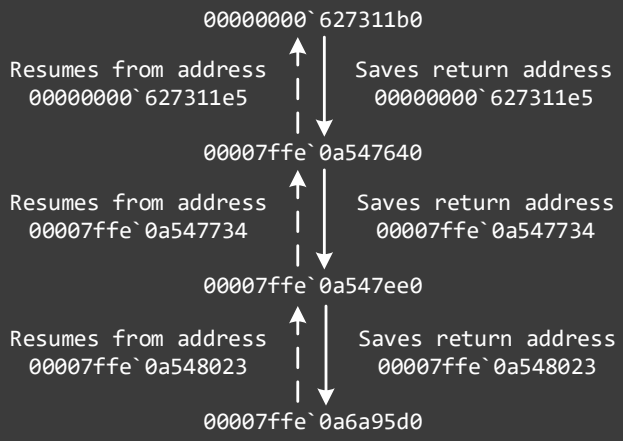
WinDbg Commands

```
0:000> k
Module+0
Module+43130
Module+32220
Module+22110
```

Thread Stack Trace (JIT Code)



C# Code	JIT Code
`System.Windows.UIElement.RaiseEvent(...)`	`00000000`627311b0`
`{ ... System.Windows.UIElement.RaiseEventImpl(...); ...`	`00000000`627311e5`
`}`	
`System.Windows.UIElement.RaiseEventImpl(...)`	`00007ffe`0a547640`
`{ ... System.Windows.EventRoute.InvokeHandlersImpl(...); ...`	`00007ffe`0a547734`
`}`	
`System.Windows.EventRoute.InvokeHandlersImpl(...)`	`00007ffe`0a547ee0`
`{ ... ApplicationA.MainWindow.Button_Click(...); ...`	`00007ffe`0a548023`
`}`	
`ApplicationA.MainWindow.Button_Click(...)`	`00007ffe`0a6a95d0`
`{ ... *p = 1; ...`	`00007ffe`0a6a9609` PresentationCore+0x4011e5
`}`	



WinDbg Commands

```
0:000> k
...
0x00007ffe`0a6a9609
0x00007ffe`0a548023
0x00007ffe`0a547734
PresentationCore+0x4011e5
...
```

Example: Unmanaged Stack Trace

```
0:000> kL
 # Child-SP          RetAddr           Call Site
00 00000029`6e3add78 00007ffa`7eb90458 win32u!NtUserWaitMessage+0x14
01 00000029`6e3add80 00007ffa`d7244d61 0x00007ffa`7eb90458
02 00000029`6e3ade40 00007ffa`d7247486 System_Windows_Forms!...+0x581
03 00000029`6e3adf50 00007ffa`d7247026 System_Windows_Forms!...+0x416
04 00000029`6e3ae000 00007ffa`d6ef212b System_Windows_Forms!...+0x46
05 00000029`6e3ae060 00007ffa`7cb021e4 System_Windows_Forms!...+0x3b
06 00000029`6e3ae0a0 00007ffa`7cae2534 0x00007ffa`7cb021e4
07 00000029`6e3ae350 00007ffa`7c70a16d 0x00007ffa`7cae2534
08 00000029`6e3ae680 00007ffa`7c7015c2 0x00007ffa`7c70a16d
09 00000029`6e3ae6e0 00007ffa`dc226ce3 0x00007ffa`7c7015c2
0a 00000029`6e3ae7b0 00007ffa`dc1bcc42 coreclr!CallDescrWorkerInternal+0x83
0b (Inline Function) --------`-------- coreclr!CallDescrWorkerWithHandler+0x57
0c 00000029`6e3ae7f0 00007ffa`dc1c3c09 coreclr!MethodDescCallSite::...+0x196
0d (Inline Function) --------`-------- coreclr!MethodDescCallSite::Call+0xb
0e 00000029`6e3ae930 00007ffa`dc1c4097 coreclr!RunMain+0x1f5
0f 00000029`6e3aeb10 00007ffa`dc1c4841 coreclr!Assembly::ExecuteMainMethod+0x1cb
10 00000029`6e3aeea0 00007ffa`dc0e21c1 coreclr!CorHost2::ExecuteAssembly+0x221
11 00000029`6e3af030 00007ffb`2ef54e2d coreclr!coreclr_execute_assembly+0x101
12 00000029`6e3af0d0 00007ffb`2ef62e27 hostpolicy!coreclr_t::execute_assembly+0x2d
13 00000029`6e3af120 00007ffb`2ef62a36 hostpolicy!run_app_for_context+0x387
14 00000029`6e3af280 00007ffb`2ef64262 hostpolicy!run_app+0x46
15 00000029`6e3af2d0 00007ffb`36ff3a7e hostpolicy!corehost_main+0x132
16 00000029`6e3af480 00007ffb`36ff72d8 hostfxr!execute_app+0x1de
17 (Inline Function) --------`-------- hostfxr!?A0xd51c85dd::read_config...+0x10a
18 00000029`6e3af570 00007ffb`36ff5b5b hostfxr!fx_muxer_t::handle_exec_...+0x214
19 00000029`6e3af660 00007ffb`36ff2109 hostfxr!fx_muxer_t::execute+0x39b
1a 00000029`6e3af7a0 00007ff6`ea712361 hostfxr!hostfxr_main_startupinfo+0x89
1b 00000029`6e3af8a0 00007ff6`ea712758 LINQPad6!exe_start+0x651
1c 00000029`6e3afad0 00007ff6`ea714608 LINQPad6!wmain+0x88
1d (Inline Function) --------`-------- LINQPad6!invoke_main+0x22
1e 00000029`6e3afb00 00007ffb`58387034 LINQPad6!__scrt_common_main_seh+0x10c
1f 00000029`6e3afb40 00007ffb`5a002651 kernel32!BaseThreadInitThunk+0x14
20 00000029`6e3afb70 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

```
0:000> !IP2MD 0x00007ffa`7cae2534
MethodDesc:   00007ffa7c7f5aa0
Method Name:  LINQPad.UIProgram.Go(System.String[])
Class:        00007ffa7c803a28
MethodTable:  00007ffa7c7f7078
mdToken:      00000000060001A4
Module:       00007ffa7c79f798
IsJitted:     yes
Current CodeAddr:     00007ffa7cae1b90
Version History:
  ILCodeVersion:      0000000000000000
  ReJIT ID:           0
  IL Addr:            0000000000000000
     CodeAddr:           00007ffa7cae1b90
(MinOptJitted)
     NativeCodeVersion:  0000000000000000

0:000> !DumpModule 00007ffa7c79f798
Name: C:\Program Files\LINQPad6\LINQPad.GUI.dll
Attributes:              PEFile
SupportsUpdateableMethods
Assembly:                00000177c5cdece0
BaseAddress:             00000177DFDE0000
PEFile:                  00000177C5CDE0E0
ModuleId:                00007FFA7C7D1860
ModuleIndex:             0000000000000001
LoaderHeap:              0000000000000000
TypeDefToMethodTableMap: 00007FFA7C7A0020
TypeRefToMethodTableMap: 00007FFA7C7A1860
MethodDefToDescMap:      00007FFA7C7A4138
FieldDefToDescMap:       00007FFA7C7AF300
MemberRefToDescMap:      0000000000000000
FileReferencesMap:       00007FFA7C7B7F78
AssemblyReferencesMap:   00007FFA7C7B7F80
MetaData start address:  00000177DFE887DC (719936
bytes)
```

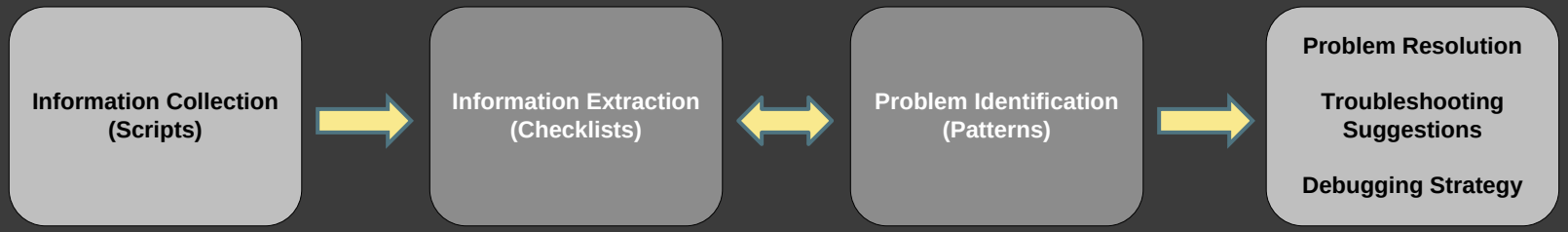
Pattern-Oriented Diagnostic Analysis

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

Part 2: Practice Exercises

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in this book

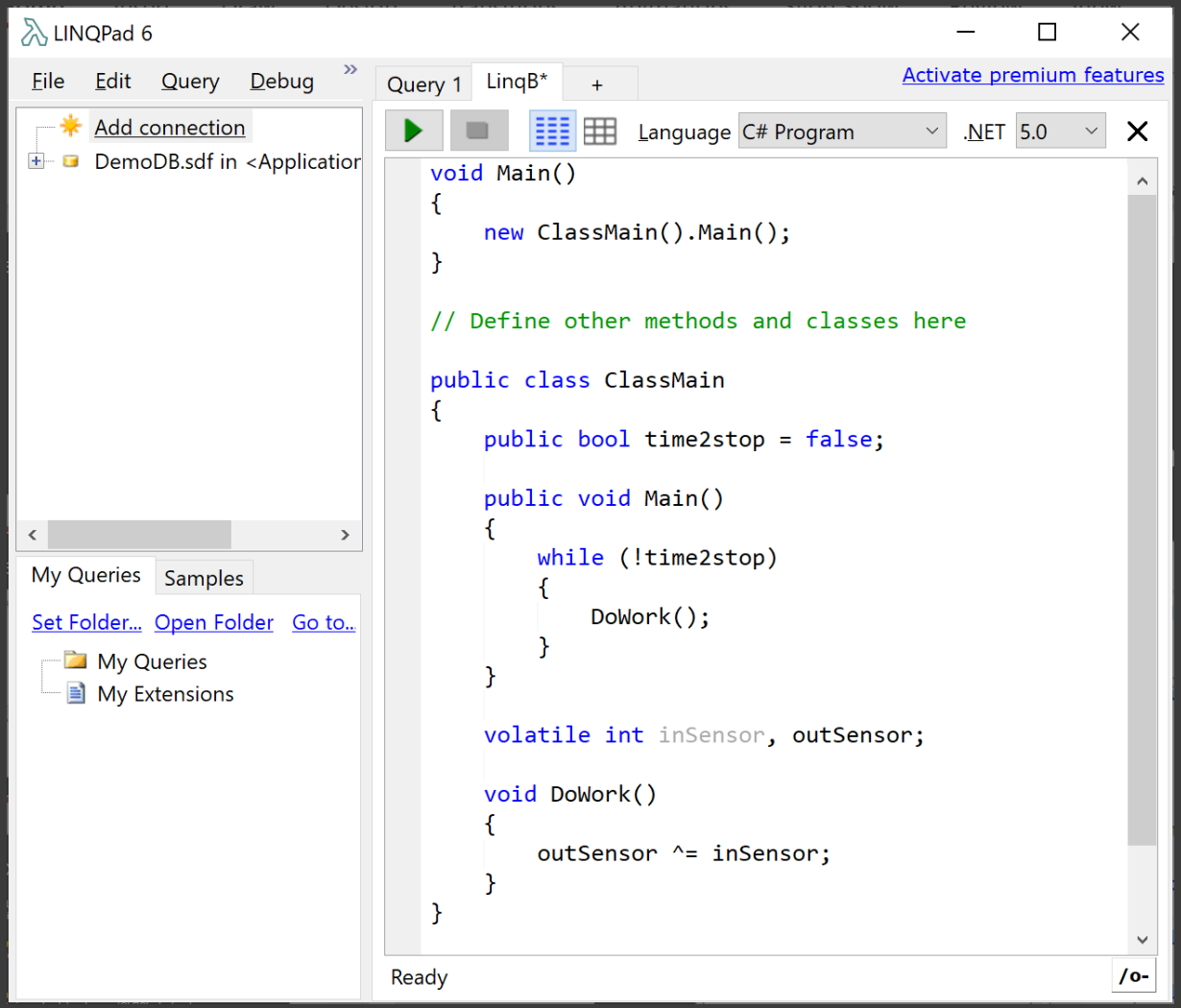
Exercise 0

- **Goal:** Install WinDbg Preview or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly
- **Patterns:** Stack Trace; Incorrect Stack Trace; Truncated Stack Trace
- **Commands:** k
- \ANETCMDA-Dumps\Exercise-0-Download-Setup-WinDbg.pdf

Process Memory Dumps

Practice Exercises PNC1 – PNC8

Modeling with LINQPad 6/7



http://www.linqpad.net/

Exercise PNC1

- **Goal:** Learn how to use the SOS WinDbg extension to analyze managed space for the presence of exceptions
- **Patterns:** Stack Trace Collection (Unmanaged Space); CLR Thread; Software Exception; Exception Stack Trace; Managed Code Exception; Managed Stack Trace; Invalid Pointer; NULL Pointer (Data)
- **Commands:** .logopen, version, !peb, ~*k, ~*kL, .load, !pe, ~*e, lmv, .chain, .unload, !analyze -v, !CLRStack, .logclose
- \ANETCMDA-Dumps\Exercise-PNC1-Analysis-process-dump-ApplicationA.pdf

Exercise PNC2

- **Goal:** Compare the 64-bit process memory dump from exercise PNC1 with a 32-bit process memory dump
- **Patterns:** Platform-Specific Debugger
- \ANETCMDA-Dumps\Exercise-PNC2-Analysis-process-dump-ApplicationA-32.pdf

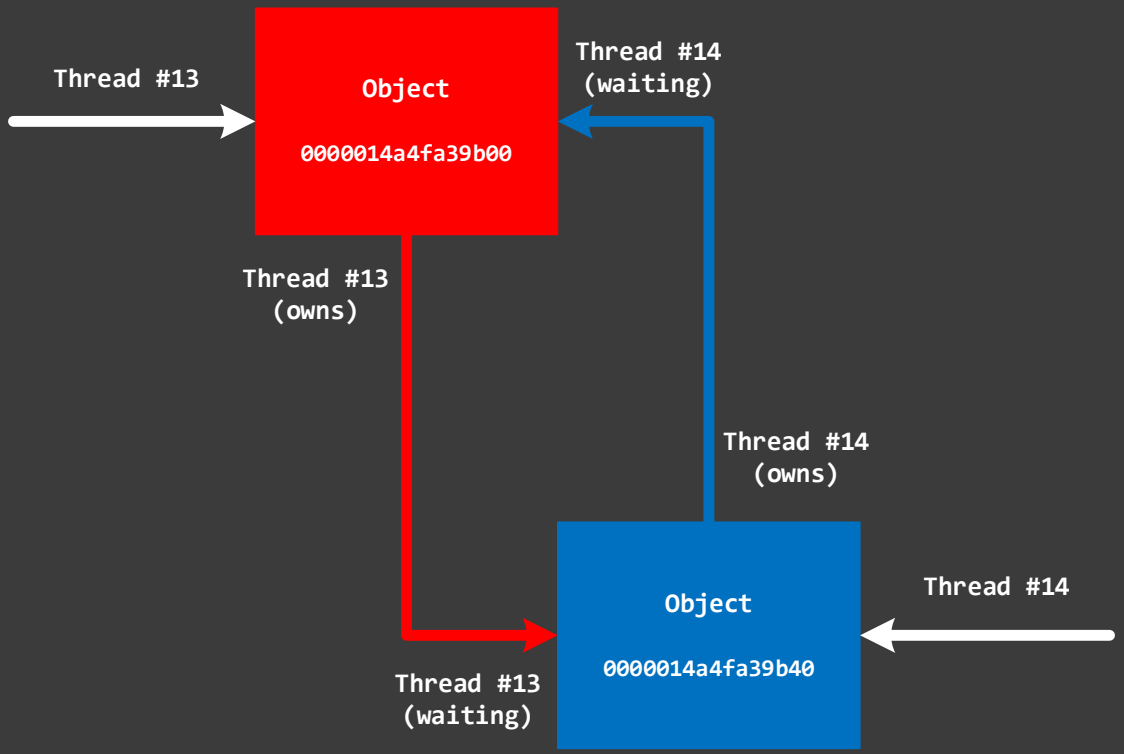
Exercise PNC3

- **Goal:** Learn how to find problem assemblies, modules, classes and methods, disassemble code, analyze CPU spikes
- **Patterns:** Active Thread; Manual Dump (Process); Technology-Specific Subtrace (JIT .NET Code); Spiking Thread; JIT Code; Annotated Disassembly (JIT .NET Code)
- **Commands:** !analyze -v -hang, !IP2MD, !runaway, ~<>k, !U, !DumpMD, !DumpClass, !DumpMT, !DumpModule, !DumpAssembly, !DumpDomain, !DumpIL
- \ANETCMDA-Dumps\Exercise-PNC3-Analysis-process-dump-LINQPadB.pdf

Exercise PNC4

- **Goal:** Learn how to recognize and analyze deadlocks using SOS, execution residue, handled exceptions, dump object references
- **Patterns:** Special Thread (.NET CLR); Wait Chain (CLR Monitors); Deadlock (Managed Space); Execution Residue (User and Managed Spaces); Value References; Hidden Exception (User and Managed Spaces); Handled Exception (.NET CLR); Coincidental Symbolic Information; Rough Stack Trace (Unmanaged Space); Caller-n-Callee
- **Commands:** ~<>s, !Threads, !syncblk, !DumpObj, ub, dps, !DumpStack, !teb, dpS, !DumpStackObjects
- \ANETCMDA-Dumps\Exercise-PNC4-Analysis-process-dump-LINQPadC.pdf

Deadlock



Exercise PNC5

- **Goal:** Learn how to analyze multiple managed exceptions
- **Patterns:** Stack Trace Collection (Managed Space); Multiple Exceptions (Managed Space)
- **Commands:** .sympath+, !help
- \ANETCMDA-Dumps\Exercise-PNC5-Analysis-process-dump-ApplicationD.pdf

Exercise PNC6

- **Goal:** Learn how to diagnose heap and handle leaks
- **Patterns:** Handle Leak; Object Distribution Anomaly (.NET Heap); Memory Leak (.NET Heap)
- **Commands:** !heap, !address, !DumpHeap, ?, !eeheap, !GCHandles, !FinalizeQueue, !handle
- \ANETCMDA-Dumps\Exercise-PNC6-Analysis-process-dump-LINQPadD.pdf

Exercise PNC7

- **Goal:** Learn how to recognize and analyze heap corruption
- **Patterns:** Regular Data; Dynamic Memory Corruption (Managed Heap)
- **Commands:** .formats, !VerifyHeap
- \ANETCMDA-Dumps\Exercise-PNC7-Analysis-process-dump-LINQPadE.pdf

Exercise PNC8

- **Goal:** Learn how to navigate virtual memory, search stack traces and memory for data and objects
- **Patterns:** Stack Overflow (Managed Space); Stack Trace Set
- **Commands: .**kframes, !VMMap, !address, !ListNearObj, !ObjSize, !uniqstack, !findstack, !DumpArray, dpa, dpu, s
- \ANETCMDA-Dumps\Exercise-PNC8-Analysis-process-dump-LINQPadF.pdf

Pattern Links

CLR Thread
Managed Code Exception
Stack Trace Collection
Memory Leak
JIT Code
Managed Stack Trace
Multiple Exceptions
Caller-n-Callee
Hidden Exception
Technology-Specific Subtrace
Stack Overflow
Dynamic Memory Corruption
Special Thread
Execution Residue
Handled Exception
Annotated Disassembly
Wait Chain
Deadlock
Object Distribution Anomaly

CLR-related and managed (full list)

Incorrect Stack Trace
Execution Residue
Stack Trace
NULL Pointer (Data)
Handle Leak
Exception Stack Trace
Software Exception
Platform-Specific Debugger
Spiking Thread
Hidden Exception
Regular Data
Coincidental Symbolic Information
Active Thread
Truncated Stack Trace
Value References
Manual Dump
Stack Trace Set
Rough Stack Trace
Stack Trace Collection

Unmanaged user space

SOS Checklist

- CLR module and SOS extension versions (lmv and .chain)
- Managed exceptions (~*e !pe -nested)
- Managed threads (!Threads -special)
- Managed stack traces (!CLRStack -all)
- Managed execution residue (~*e !DumpStackObjects)
- Managed heap (!VerifyHeap, !DumpHeap -stat and !eeheap -gc)
- GC handles (!GCHandles)
- Finalizer queue (!FinalizeQueue)
- Sync blocks (!syncblk)

Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- .NET Runtime https://github.com/dotnet/runtime
- Pro .NET Memory Management: For Better Code, Performance, and Scalability
- Accelerated Windows Memory Dump Analysis, 5th Edition, Revision 3, Part 1: Process User Space
- Encyclopedia of Crash Dump Analysis Patterns, 3rd Edition
- Memory Dump Analysis Anthology (Diagnomicon)

Q&A

Please send your feedback using the contact form on PatternDiagnostics.com

Thank you for attendance!