Production debugging

Stas Shteinbook

Senior Consultant

stass@CodeValue.net

CodeValue

Debugging process

- Things to do before you debug
  - Keep debugging tools at hand
  - Set up and manage a symbol server
  - Develop logging and error handling mechanisms in your applications
  - Track changes to your environment

CodeValue

The debugging algorithm

1. Don’t panic
2. Duplicate
3. Always assume the bug is yours
4. Separate facts from interpretation
    1. Wrong interpretation leads to a dead-end
5. Divide and conquer
    1. Do one change at a time
6. Write down what you do
7. Utilize tools
8. Start heavy debugging
9. Verify that the bug is fixed

(c) Debugging Microsoft .NET 2.0 applications by John Robbins + additions

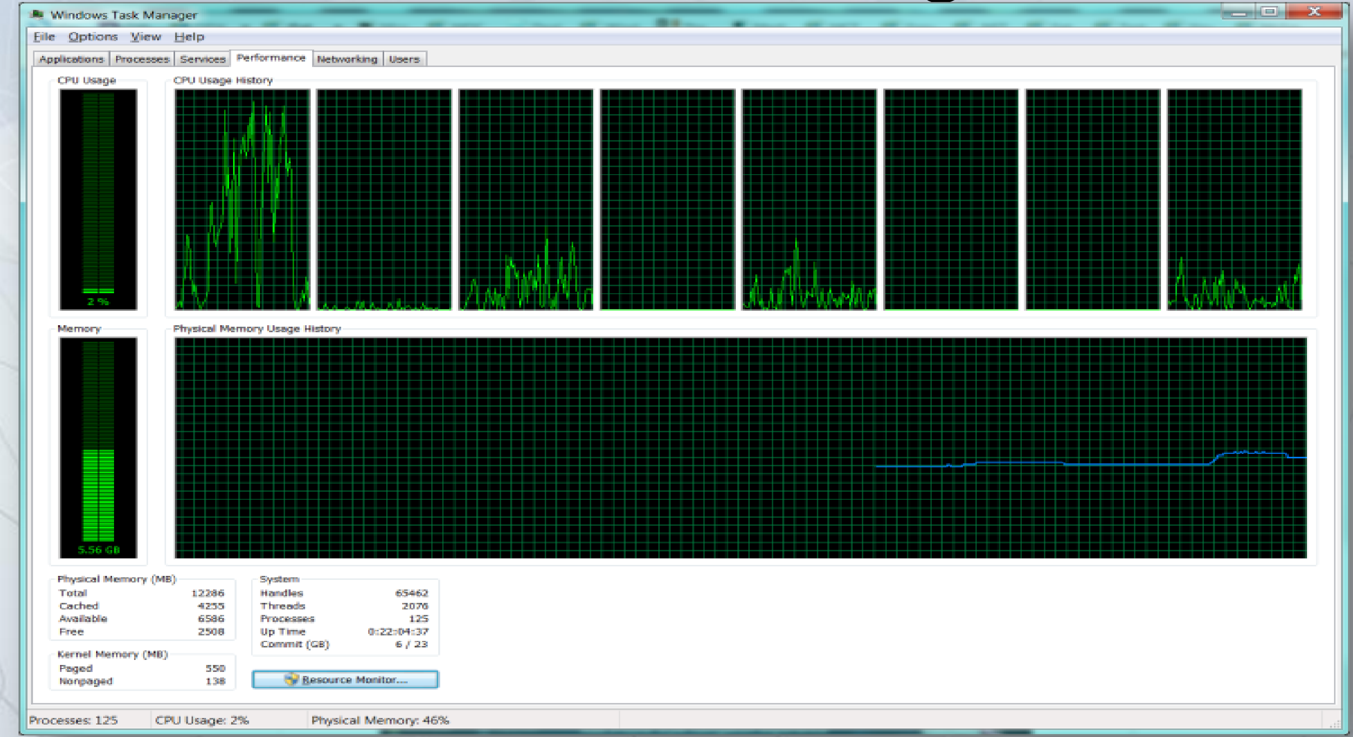
CodeValue

Tools of the Trade

- Process Explorer
- Process Monitor
- DebugView
  - All from www.sysinternals.com
- WinDbg
- ADPlus
  - Both from Debugging Tools For Windows

Windows Built-In Tools

Task Manager



Physical Memory (MB)	
Total	12286
Cached	4255
Available	6586
Free	2508

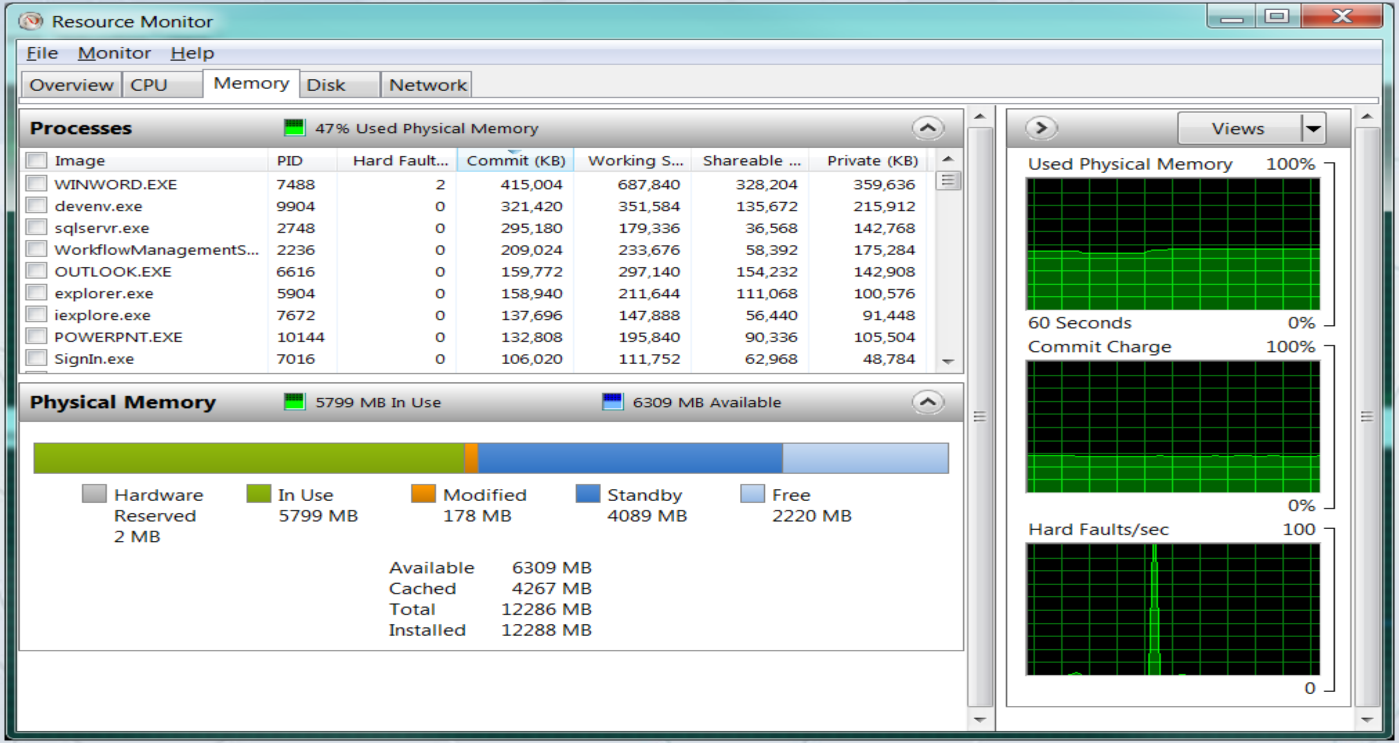
Kernel Memory (MB)	
Paged	550
Nonpaged	138

System	
Handles	65462
Threads	2076
Processes	125
Up Time	0:22:04:37
Commit (GB)	6 / 23

Processes: 125 CPU Usage: 2% Physical Memory: 46%

Windows Built-In Tools

Resource Monitor



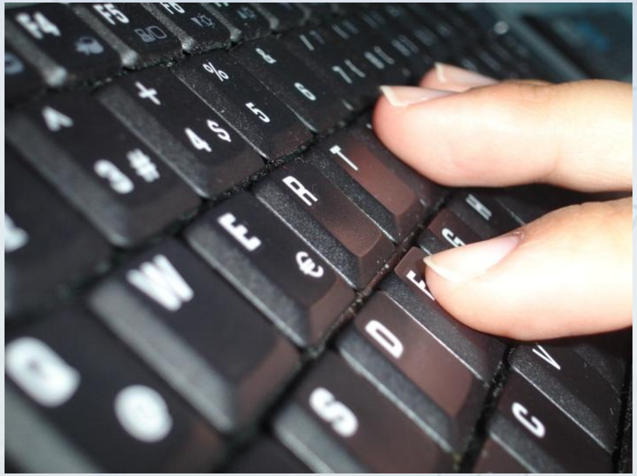
Processes 🟩 47% Used Physical Memory

Image	PID	Hard Fault...	Commit (KB)	Working S...	Shareable ...	Private (KB)
WINWORD.EXE	7488	2	415,004	687,840	328,204	359,636
devenv.exe	9904	0	321,420	351,584	135,672	215,912
sqlservr.exe	2748	0	295,180	179,336	36,568	142,768
WorkflowManagementS...	2236	0	209,024	233,676	58,392	175,284
OUTLOOK.EXE	6616	0	159,772	297,140	154,232	142,908
explorer.exe	5904	0	158,940	211,644	111,068	100,576
iexplore.exe	7672	0	137,696	147,888	56,440	91,448
POWERPNT.EXE	10144	0	132,808	195,840	90,336	105,504
SignIn.exe	7016	0	106,020	111,752	62,968	48,784

Physical Memory 🟩 5799 MB In Use 🟦 6309 MB Available

Hardware Reserved 2 MB	In Use 5799 MB	Modified 178 MB	Standby 4089 MB	Free 2220 MB

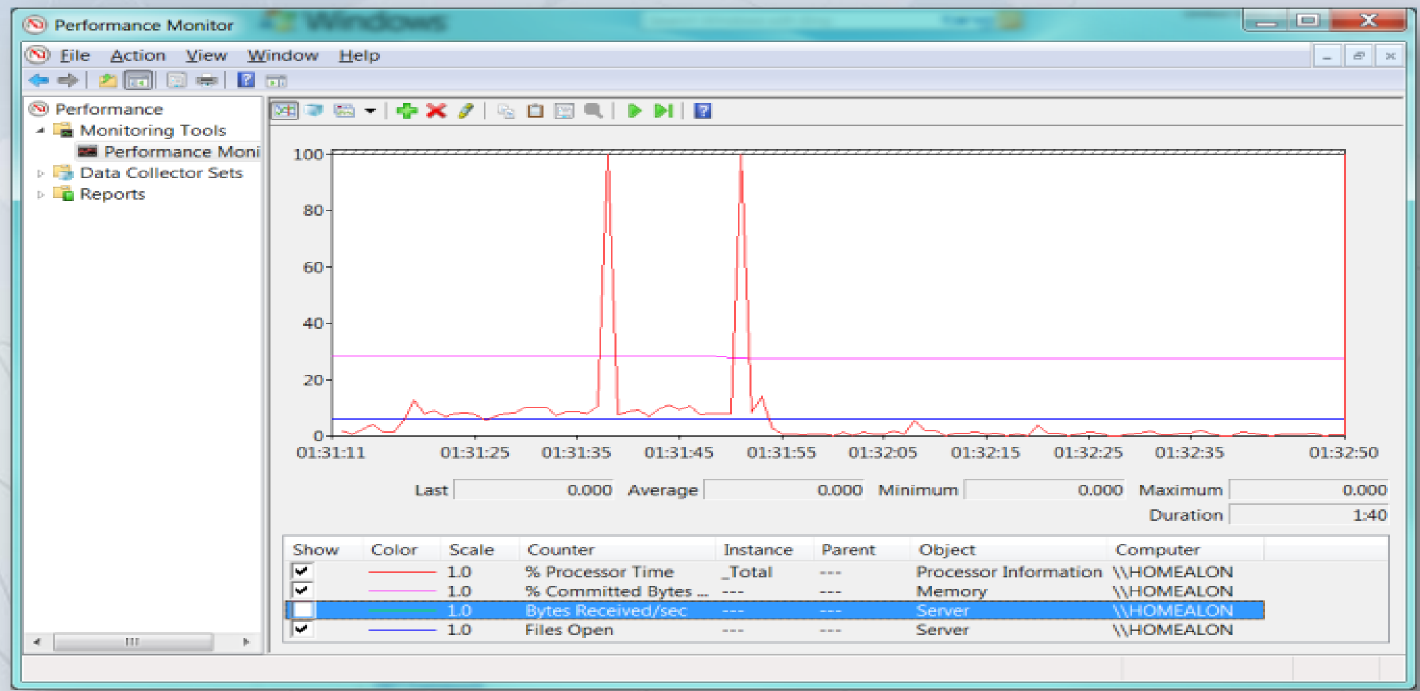
Available	6309 MB
Cached	4267 MB
Total	12286 MB
Installed	12288 MB



Simple deadlock Demo

Windows Built-In Tools

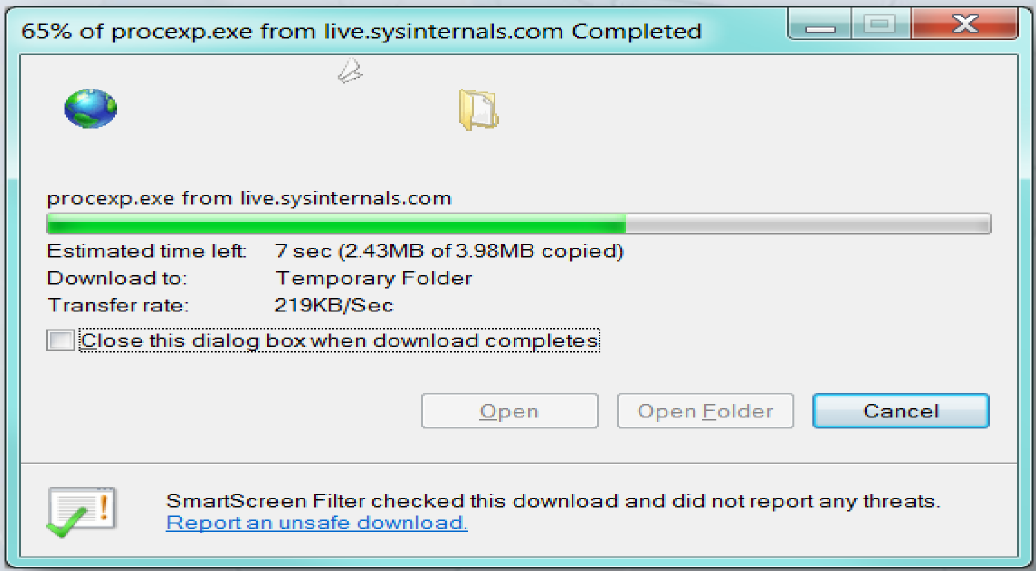
Performance Monitor



Last	Average	Minimum	Maximum	Duration
0.000	0.000	0.000	0.000	1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	🟥	1.0	% Processor Time	_Total	---	Processor Information	\\HOMEALON
✓	🟪	1.0	% Committed Bytes ...	---	---	Memory	\\HOMEALON
	🟩	1.0	Bytes Received/sec	---	---	Server	\\HOMEALON
✓	🟦	1.0	Files Open	---	---	Server	\\HOMEALON

http://live.sysinternals.com


65% of procexp.exe from live.sysinternals.com Completed

procexp.exe from live.sysinternals.com

Estimated time left: 7 sec (2.43MB of 3.98MB copied)
Download to: Temporary Folder
Transfer rate: 219KB/Sec

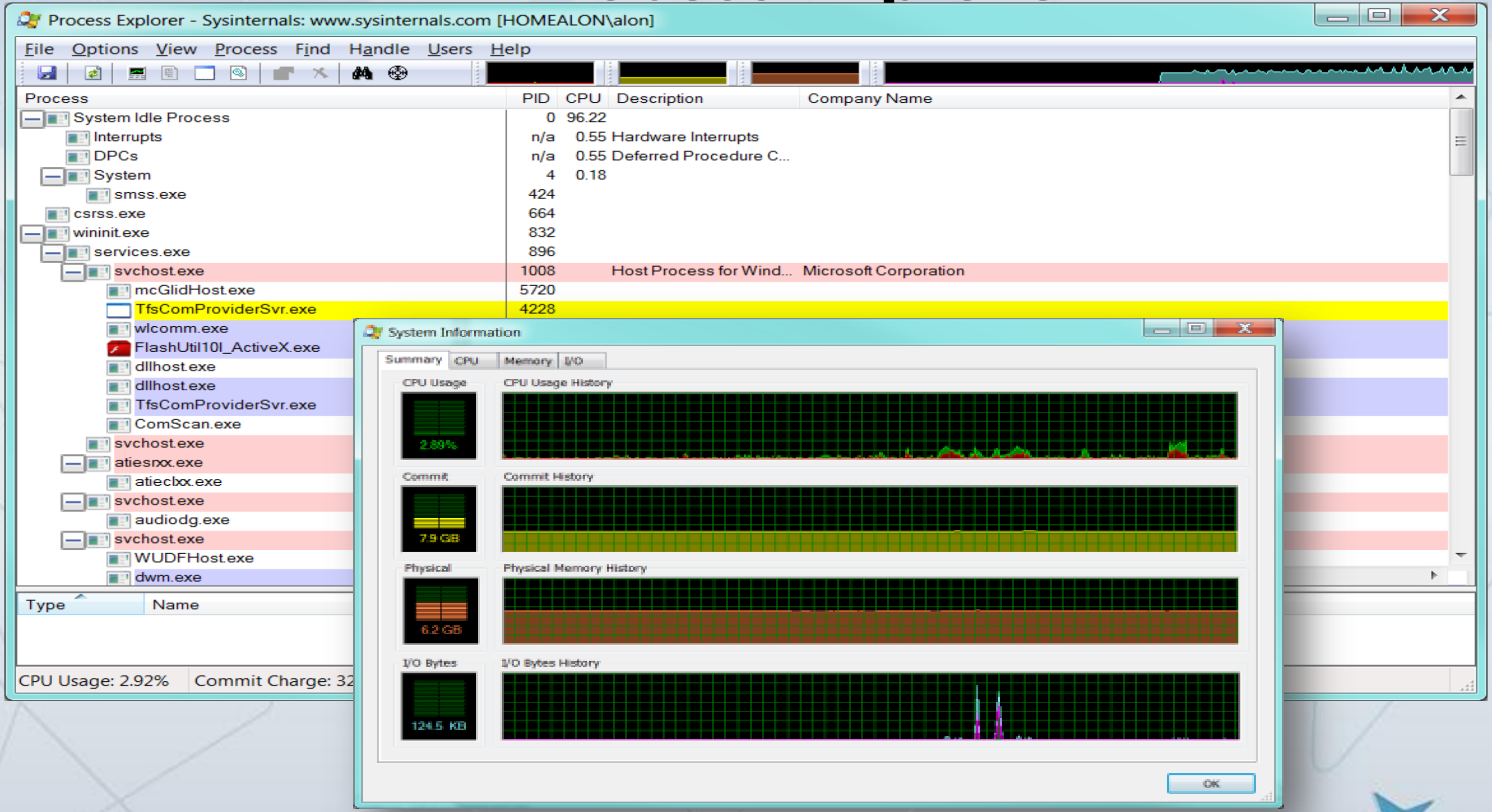
Close this dialog box when download completes

SmartScreen Filter checked this download and did not report any threats.
Report an unsafe download.

CodeValue

System Internals Tools

Process Explorer



Process Explorer - Sysinternals: www.sysinternals.com [HOMEALON\alon]

File Options View Process Find Handle Users Help

Process	PID	CPU	Description	Company Name
System Idle Process	0	96.22		
Interrupts	n/a	0.55	Hardware Interrupts	
DPCs	n/a	0.55	Deferred Procedure C...	
System	4	0.18		
smss.exe	424			
csrss.exe	664			
wininit.exe	832			
services.exe	896			
svchost.exe	1008		Host Process for Wind...	Microsoft Corporation
mcGlidHost.exe	5720			
TfsComProviderSvr.exe	4228			
wlcomm.exe				
FlashUtil10l_ActiveX.exe				
dllhost.exe				
dllhost.exe				
TfsComProviderSvr.exe				
ComScan.exe				
svchost.exe				
atiesrxx.exe				
atieclxx.exe				
svchost.exe				
audiodg.exe				
svchost.exe				
WUDFHost.exe				
dwm.exe				

Type Name

CPU Usage: 2.92% Commit Charge: 32

System Information

Summary CPU Memory I/O

CPU Usage 2.89% — CPU Usage History

Commit 7.9 GB — Commit History

Physical 6.2 GB — Physical Memory History

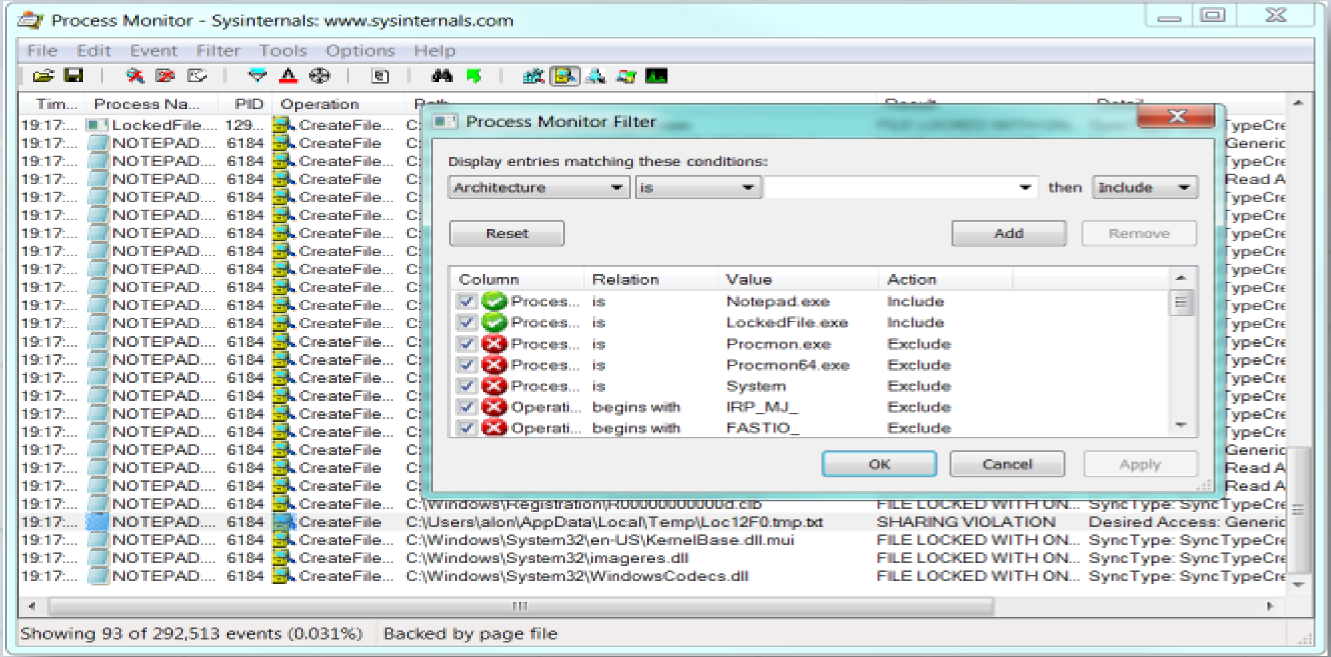
I/O Bytes 124.5 KB — I/O Bytes History

OK

CodeValue

System Internals Tools

Process Monitor


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

```
Tim...   Process Na...   PID   Operation     Path                                            Result                    Detail
19:17:...  LockedFile...  129...  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile     C:                                                                    Generic
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile     C:                                                                    Read A
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:                                                                    TypeCre
19:17:...  NOTEPAD....  6184  CreateFile     C:                                                                    Generic
19:17:...  NOTEPAD....  6184  CreateFile     C:                                                                    Read A
19:17:...  NOTEPAD....  6184  CreateFile     C:                                                                    Read A
19:17:...  NOTEPAD....  6184  CreateFile...  C:\Windows\Registration\R000000000000d.clb      FILE LOCKED WITH ON...   SyncType: SyncTypeCre
19:17:...  NOTEPAD....  6184  CreateFile     C:\Users\alon\AppData\Local\Temp\Loc12F0.tmp.txt SHARING VIOLATION      Desired Access: Generic
19:17:...  NOTEPAD....  6184  CreateFile...  C:\Windows\System32\en-US\KernelBase.dll.mui    FILE LOCKED WITH ON...   SyncType: SyncTypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:\Windows\System32\imageres.dll                FILE LOCKED WITH ON...   SyncType: SyncTypeCre
19:17:...  NOTEPAD....  6184  CreateFile...  C:\Windows\System32\WindowsCodecs.dll           FILE LOCKED WITH ON...   SyncType: SyncTypeCre
```

Showing 93 of 292,513 events (0.031%)   Backed by page file

Process Monitor Filter

Display entries matching these conditions:

Architecture | is | | then | Include

Reset   Add   Remove

	Column	Relation	Value	Action
✓	Proces...	is	Notepad.exe	Include
✓	Proces...	is	LockedFile.exe	Include
✓	Proces...	is	Procmon.exe	Exclude
✓	Proces...	is	Procmon64.exe	Exclude
✓	Proces...	is	System	Exclude
✓	Operati...	begins with	IRP_MJ_	Exclude
✓	Operati...	begins with	FASTIO_	Exclude

OK   Cancel   Apply

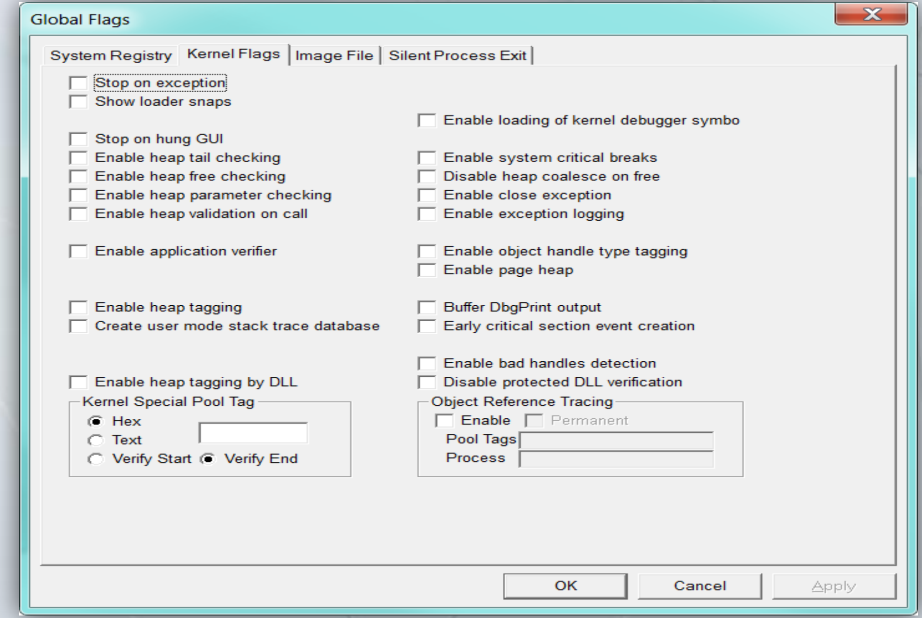
CodeValue

Sharing Violation Detection Exercise

- Exercise
  - Run – LockedFile program
    - Use Process Monitor to find out why notepad complains about a sharing violation
      - Hint: look for a SHARING_VIOLATION result
    - Once you’ve found the file name, use Process Monitor to see who holds (has a handle) to the file.
      - Go to the process that has a handle to the file (double-click) and try to close the handle
        - » Process explorer has to run as admin for that
    - See if you can use notepad to open the file now
    - Look at the source code and solve the problem

Global Flags

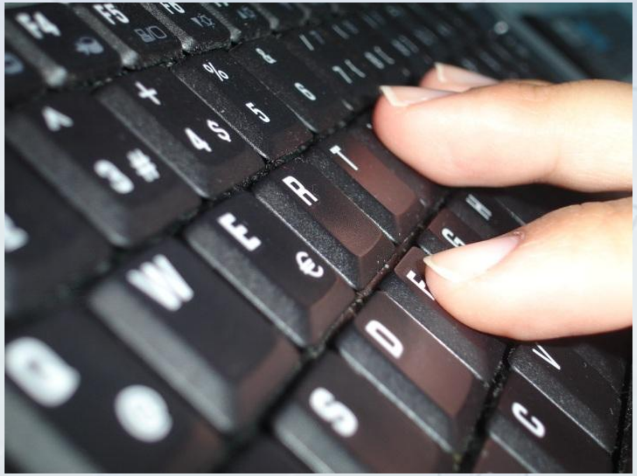
- GFlags (the Global Flags Editor), gflags.exe, enables and disables advanced debugging, diagnostic, and troubleshooting features



☐ Stop on exception
☐ Show loader snaps
☐ Enable loading of kernel debugger symbo
☐ Stop on hung GUI
☐ Enable heap tail checking
☐ Enable heap free checking
☐ Enable heap parameter checking
☐ Enable heap validation on call
☐ Enable application verifier
☐ Enable heap tagging
☐ Create user mode stack trace database
☐ Enable heap tagging by DLL
☐ Enable system critical breaks
☐ Disable heap coalesce on free
☐ Enable close exception
☐ Enable exception logging
☐ Enable object handle type tagging
☐ Enable page heap
☐ Buffer DbgPrint output
☐ Early critical section event creation
☐ Enable bad handles detection
☐ Disable protected DLL verification

Kernel Special Pool Tag
[✓ Hex] Text
Verify Start [✓ Verify End]

Object Reference Tracing
☐ Enable ☐ Permanent
Pool Tags
Process

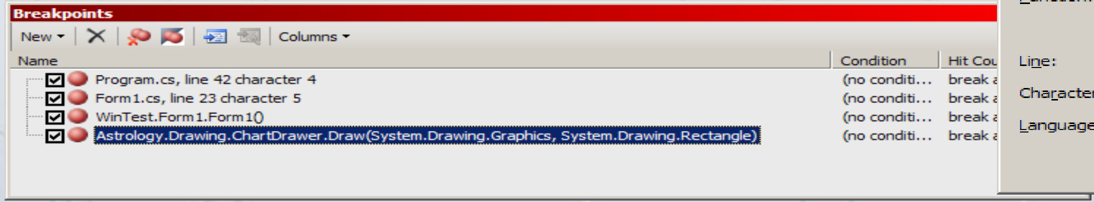


Gflags Demo

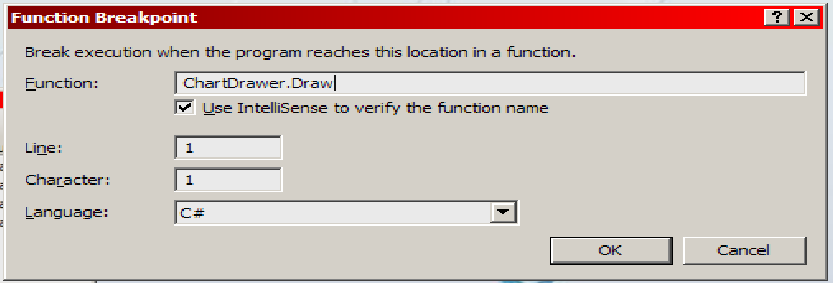
VISUAL STUDIO TOOLS

Setting New Breakpoints

- The familiar way
  - Click on the left margin of the line (or press F9 when the cursor is on the line)
- Sometimes a better way
  - Create a new breakpoint (Ctrl + B)
  - Can specify the method name without opening the required source



Name	Condition	Hit Cou
☑ Program.cs, line 42 character 4	(no conditi...	break a
☑ Form1.cs, line 23 character 5	(no conditi...	break a
☑ WinTest.Form1.Form1()	(no conditi...	break a
☑ Astrology.Drawing.ChartDrawer.Draw(System.Drawing.Graphics, System.Drawing.Rectangle)	(no conditi...	break a



Break execution when the program reaches this location in a function.

Function: ChartDrawer.Draw

☑ Use IntelliSense to verify the function name

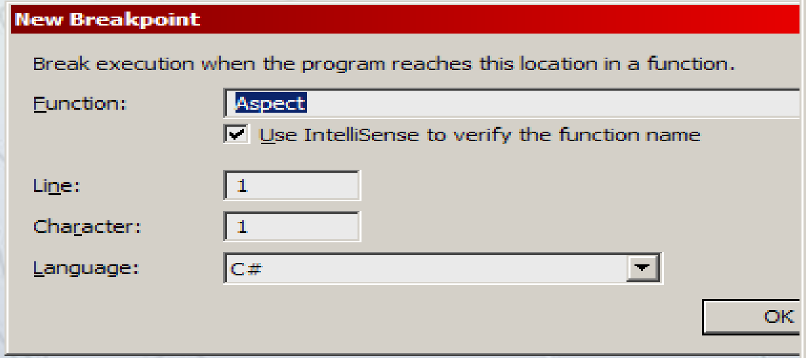
Line: 1

Character: 1

Language: C#

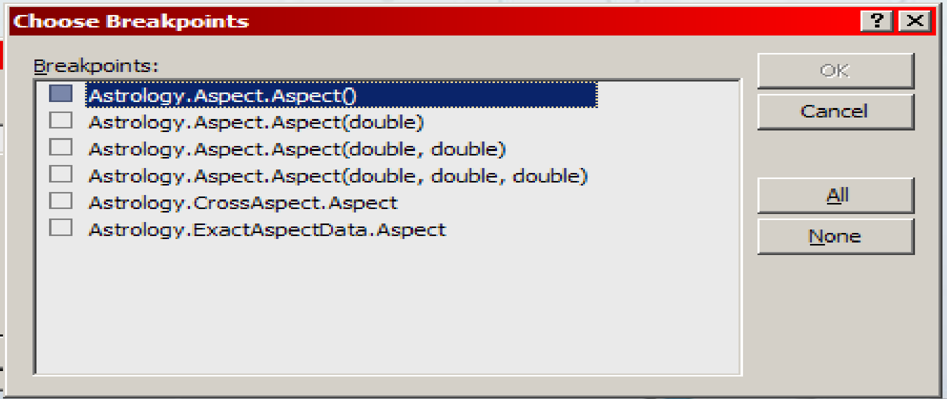
More Method Breakpoints

- If you type a method name (without class name)
  - A dialog opens, allowing selection of methods with the same name


New Breakpoint

Break execution when the program reaches this location in a function.

Function:	Aspect
	[✓] Use IntelliSense to verify the function name
Line:	1
Character:	1
Language:	C#


Choose Breakpoints

Breakpoints:

- [ ] Astrology.Aspect.Aspect()
- [ ] Astrology.Aspect.Aspect(double)
- [ ] Astrology.Aspect.Aspect(double, double)
- [ ] Astrology.Aspect.Aspect(double, double, double)
- [ ] Astrology.CrossAspect.Aspect
- [ ] Astrology.ExactAspectData.Aspect

Breakpoints Attributes

- Condition
  - An expression that is evaluated when the breakpoint is hit
    - Actually breaks into the debugger if the condition Is true / has changed
    - Expression can be anything in scope, including method calls
- Hit Count
  - Breaks into the debugger if a specified hit count is reached
    - Break always (no hit count, the default)
    - Break when the hit count is equal to <some value>
    - Break when the hit count is a multiple of <some value>
    - Break when the hit count is greater than or equal to <some value>
- Filter
  - Specific thread, process, machine

Tracepoints

- Special kind of breakpoints
- Start out as normal breakpoints
- Turning into a tracepoint
  - Right click and select “When Hit…”
  - If the “Continue execution” is selected, the breakpoint turns into a tracepoint (marked as a diamond in the left margin)
- Options
  - Display some trace message
  - Run a macro

Tracing with Tracepoints

- Any single line of text is acceptable
- Can display property/field values by surrounding with {}
- Special code can be used

Tracepoint variable name	Meaning
$ADDRESS	Current instruction address
$CALLER	Caller name of current method
$CALLSTACK	The call stack at the current location
$FUNCTION	Current function name
$PID	Current process ID
$PNAME	Current process name
$TID	Current thread ID
$TNAME	Current thread name

The “Watch” Windows

- Including Watch, Locals, Autos, This, etc.
- Things you can do
  - Change values
  - Apply conversion using the usual cast operators
  - Apply a format specifier (as comma and the specifier)

,h	show in hexadecimal
,d	show in decimal
,nq	show string without escaped quotes
,ac	force expression evaluation (if disabled globally)
,raw	raw output (valid only with **[DebuggerTypeProxy]**)
,hidden	displays all public and non-public members

  - Make an object ID

Make Object ID

- Right click a live object and select “Make Object ID”
  - “{1#}” appears after the value
  - Now “1#” indicates that object regardless of scope
- Type “1#” (no quotes) in the Watch window
  - The object can be watched at all times

WinDbg Overview

- WinDbg is a standalone GUI debugger
  - No need to install – simply xcopy
  - Used by Microsoft to debug Windows itself
  - User mode or kernel mode debugger
- UI windows
  - Command – most important window
  - Call Stack, Processes & Threads, Source, Locals, Watch, Registers, others
- Command window can do anything
  - Some shortcuts available through the GUI

CodeValue

WinDbg Commands

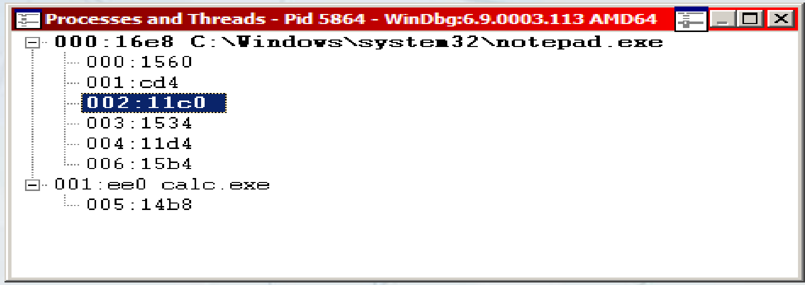
- Regular commands
  - Intrinsic to WinDbg
  - Have no prefix
  - Work on the debugged process
- Meta commands
  - Work on the debugger itself or the process of debugging itself
  - Prefixed with a dot (.)
- Extension commands (“bang” commands)
  - Supplied by extension (custom) DLLs
  - Prefixed with an exclamation mark (!)
  - Some extension DLLs are loaded automatically

Starting Process Debugging

- WinDbg can debug multiple processes at the same time
  - Create a new process
    - `.create <exe_path>`
    - Or **File->Open Executable…**
  - Attach to a running process
    - `.attach <process_id>`
    - Or **File->Attach to a Process…**
- By default, stops at **Initial Breakpoint**

Processes & Threads

- Can view processes and threads in the Processes & Threads window
- Commands
  - | (lists all processes)
  - ~ (lists all threads)
  - |*x*s (switch to process *x*)
  - ~*x*s (switch to thread *x*)
  - ~*x* (show thread *x* info)
  - !*runaway* (shows running times for all threads in descending order)


Processes and Threads - Pid 5864 - WinDbg:6.9.0003.113 AMD64

```
000:16e8 C:\Windows\system32\notepad.exe
    000:1560
    001:cd4
    002:11c0
    003:1534
    004:11d4
    006:15b4
001:ee0 calc.exe
    005:14b8
```

Configuring Symbols

- Debugging symbols come in several flavors
  - Full program database (PDB files)
  - Public symbols only (PDB files)
  - Exported symbols only (in the DLL itself)
- Select **File->Symbol File Path…**
  - Add search folders as appropriate
- Automatically uses the _NT_SYMBOL_PATH environment variable (if exists)
- To get the symbols of the OS automatically, add the following string

SRV*c:\Symbols*http://msdl.microsoft.com/download/symbols

Symbol Commands

`.reload [options] [module]`

- Reload all symbol files (useful if symbol path updated)
- !sym –noisy (will turn on noisy symbol prompts)
- Useful options
  - `/f` (forces loading of symbols now)

`lm [options]`

- Lists all loaded modules
- Shows in parenthesis the symbol loading status
- WinDbg uses deferred symbol loading
  - `lm v m [module]` (shows detailed info for module)

`ld <module_name>`

- Loads symbols for the specified module

Walking the Native Stack

- Open the Call Stack window (**View->Call Stack**)
- Issue the k command
- Common variants

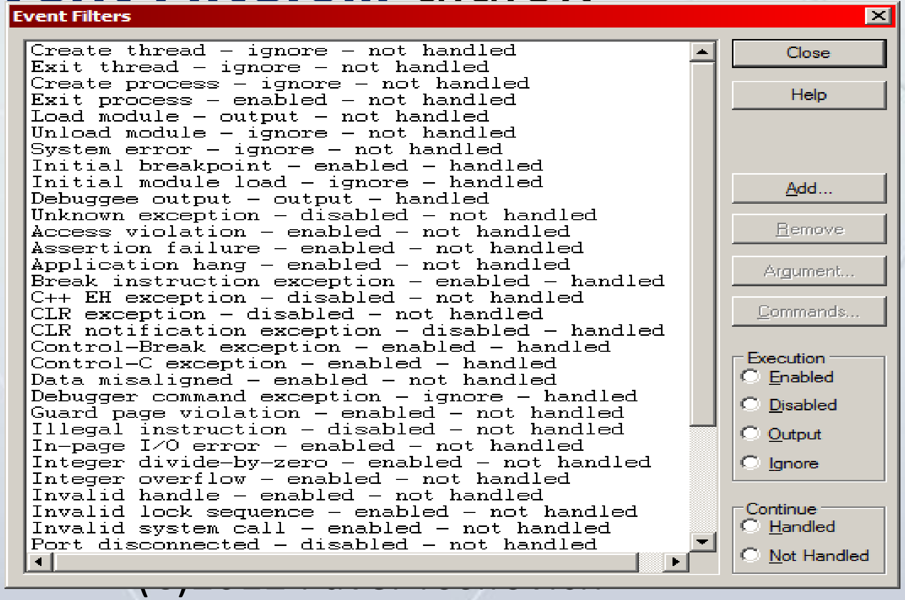
  kP (lists all function parameters)

  kb (lists first 3 parameters to each function)

  k [*n*] (lists no more than *n* stack frames)

Exceptions and Events

- What happens when an exception (SEH) occurs?
- Can be configured with the `sx*` family of commands, or the **Debug->Event Filters…** dialog



```
Create thread - ignore - not handled
Exit thread - ignore - not handled
Create process - ignore - not handled
Exit process - enabled - not handled
Load module - output - not handled
Unload module - ignore - not handled
System error - ignore - not handled
Initial breakpoint - enabled - handled
Initial module load - ignore - handled
Debuggee output - output - handled
Unknown exception - disabled - not handled
Access violation - enabled - not handled
Assertion failure - enabled - not handled
Application hang - enabled - not handled
Break instruction exception - enabled - handled
C++ EH exception - disabled - not handled
CLR exception - disabled - not handled
CLR notification exception - disabled - handled
Control-Break exception - enabled - handled
Control-C exception - enabled - handled
Data misaligned - enabled - not handled
Debugger command exception - ignore - handled
Guard page violation - enabled - not handled
Illegal instruction - disabled - not handled
In-page I/O error - enabled - not handled
Integer divide-by-zero - enabled - not handled
Integer overflow - enabled - not handled
Invalid handle - enabled - not handled
Invalid lock sequence - enabled - not handled
Invalid system call - enabled - not handled
Port disconnected - disabled - not handled
```

Event Filter Options

- Execution

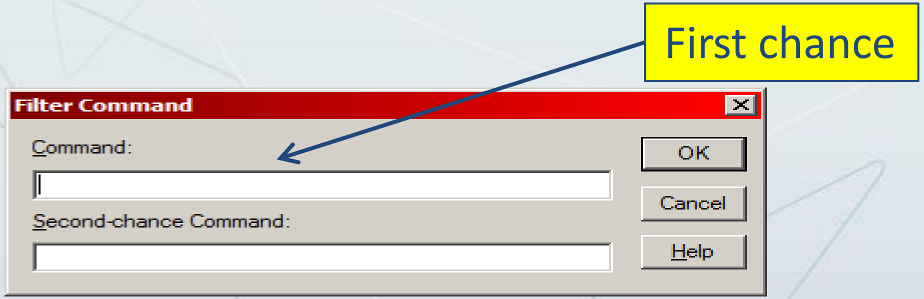
Setting	Description
Enabled	When the event/exception occurs, always break into the debugger
Disabled	The first time the event/exception occurs, the debugger ignores it (but outputs to the command window). The second time it occurs (if the debuggee did not handle it), breaks into the debugger
Output	The debugger only outputs the event/exception info without breaking
Ignore	The debugger completely ignores the event/exception

- Continue
  - Use “Not handled” so that WinDbg does not “eat” the event/exception

CodeValue

Execute On Event/Exception

- Selecting an event/exception in the Event Filter dialog and then clicking **Commands…**
- Allow you to list a set of commands to execute automatically (for first and second chance)
  - Separate several commands with a semicolon (;)



Native Breakpoints

`bp <address>` (set a normal breakpoint)
`bu <address>` (set unresolved breakpoint)
`bl` (list all breakpoints and status)
`bc [* | number]` (clear all or a specific breakpoint)
`be [* | number]` (enable all or a specific breakpoint)
`bd [* | number]` (disable all or a specific breakpoint)
`ba <access> <size> [address]` (break on access)

- In WinDbg can use **Edit->Breakpoints**
- Specifying `<address>`
  - A literal address value

    `Moudle!FunctionName` (C function)

    `Module!ClassName::FunctionName` (C++ function)
  - Can add an offset

CLR Exceptions

- Every exception type has an exception code in Structured Exception Handling (SEH)
- CLR exceptions are given the code 0xe0434f4d
  - Why this number?
- You can stop on every CLR exception using the Event Filter dialog box
  - Or use the equivalent command `sxe clr`

The `sx*` Family of Commands

- Allow setting exception options similar to the Event Filter dialog box
  - `sx` (list all exceptions/events and their status)
  - `sxe` (always break, same as "Enabled")
  - `sxd` (break on second chance, same as "Disabled")
  - `sxn` (just output, same as "Output")
  - `sxi` (ignore exception, same as "Ignore")
- Useful exception codes
  - `clr` (any CLR exception, 0xe0434f4d)
  - `out` (call to `OutputDebugString` or `Trace.Write*` for the default trace listener)
  - `eh` (C++ exception), `av` (access violation)
  - `ct` (Thread creation), `et` (thread exit)

Execute Commands on Events

- Add commands in quotes after a `-c` switch (first chance) or `-c2` (second chance)
- Example

```
sxe -c "kP;gc" clr
```

  - Does a stack trace after any CLR exception and then continues execution

Stopping on Trace Statements

- Breaking on calls to `OutputDebugString` (which is called by `Trace.Write*` methods if using the default trace listener) can be done by the command `sxe out`
- Can append a string to stop on
  - Cannot have a space or a colon, but supports the wildcards ? and *
  - Case insensitive
- Example

```
sxe out Starting?Operation*
```

Introduction to SOS

- SOS is an extension DLL for WinDbg that understands the CLR
- SOS.DLL exists on every .NET 2.0 (and up) installation
- Must be explicitly loaded before use

Loading SOS

- If the process has already loaded the CLR

  ```
  .loadby sos clr
  ```

  - Replace `clr` with `mscorwks` for .NET 3.5 and lower
- Otherwise
  - Set a breakpoint on CLR startup

    ```
    bu clr!EEStartup
    ```

  - When the debugger stops

    ```
    .loadby sos clr
    ```

  - Can combine into one command

    ```
    bu clr!EEStartup ".loadby sos clr"
    ```

- Another option
  - Just copy SOS.DLL to the WinDbg folder

  ```
  .load sos.dll
  ```

General SOS Commands

!help

!help <command>

!help faq

!eeversion

- Displays version of CLR and type of GC

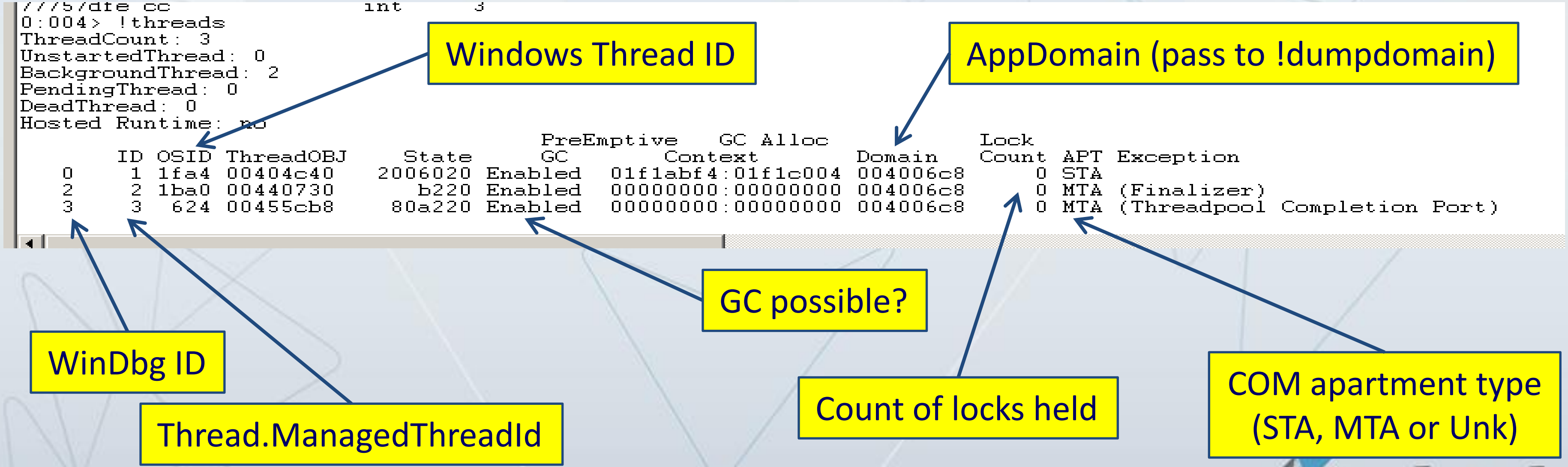
!procinfo [-mem] [-time] [-env]

- General process info

Thread Information

!threads

- **UnstartedThread**: threads created but **Start()** not yet called
- **BackgroundThread**: threads having their **IsBackground** property



```
7775/dfe cc                 int      3
0:004> !threads
ThreadCount: 3
UnstartedThread: 0
BackgroundThread: 2
PendingThread: 0
DeadThread: 0
Hosted Runtime: no
                                       PreEmptive   GC Alloc           Lock
       ID OSID ThreadOBJ    State     GC       Context       Domain   Count APT Exception
   0    1 1fa4 00404c40   2006020 Enabled  01f1abf4:01f1c004 004006c8     0 STA
   2    2 1ba0 00440730     b220 Enabled  00000000:00000000 004006c8     0 MTA (Finalizer)
   3    3  624 00455cb8    80a220 Enabled  00000000:00000000 004006c8     0 MTA (Threadpool Completion Port)
```

Walking the Managed Stack

`!clrstack`

- Shows managed stack of current thread

- Useful switches
  - `-p` (shows parameters and values)
  - `-l` (show locals)
  - `-a` (same as `-p -l`)

`!dumpstack`

- Hard working stack dump (managed & unmanaged)
  - `-ee` (managed calls only)

`!eestack`

- Same as `~*e!dumpstack` (dumps stacks of all threads)
  - `-short` (shows only “interesting” thread stacks)

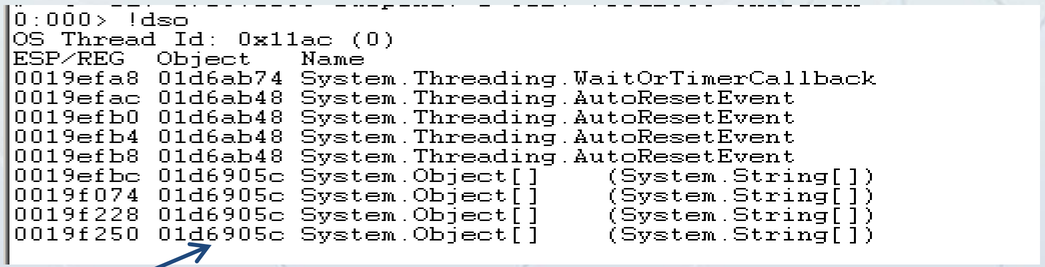
Displaying Stack Objects

!DumpStackObjects (!dso)

- Shows objects on the stack

-verify (make sure no false positives)

- Shows objects internally created by the CLR



```
0:000> !dso
OS Thread Id: 0x11ac (0)
ESP/REG  Object   Name
0019efa8 01d6ab74 System.Threading.WaitOrTimerCallback
0019efac 01d6ab48 System.Threading.AutoResetEvent
0019efb0 01d6ab48 System.Threading.AutoResetEvent
0019efb4 01d6ab48 System.Threading.AutoResetEvent
0019efb8 01d6ab48 System.Threading.AutoResetEvent
0019efbc 01d6905c System.Object[]    (System.String[])
0019f074 01d6905c System.Object[]    (System.String[])
0019f228 01d6905c System.Object[]    (System.String[])
0019f250 01d6905c System.Object[]    (System.String[])
```

Instance address in memory

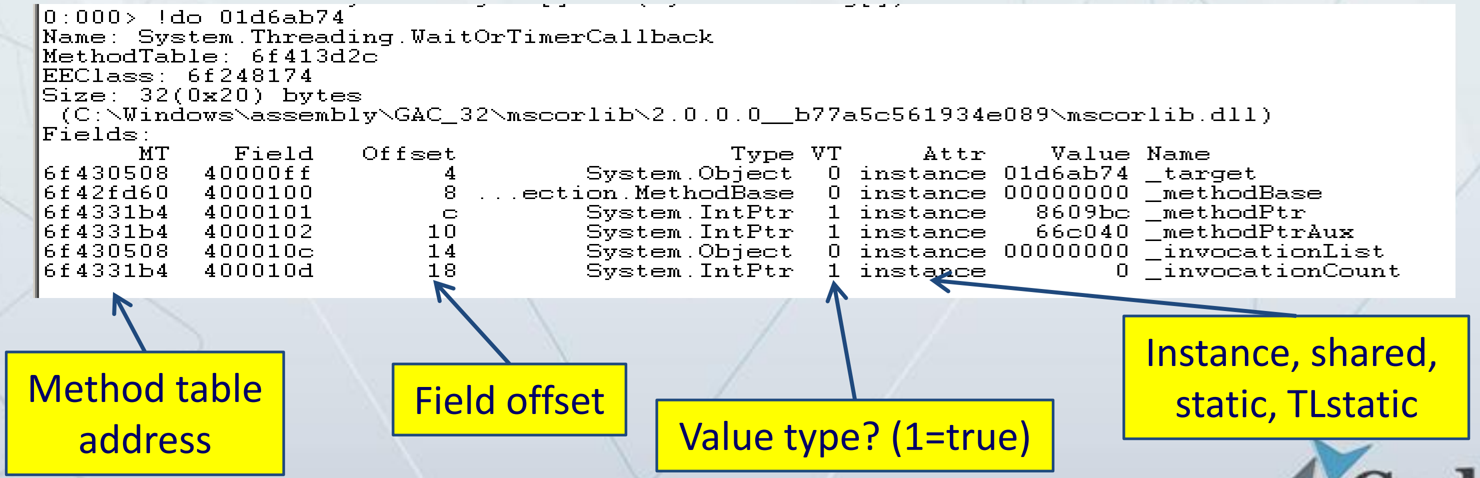
CodeValue

Displaying Object Data

!DumpObj (!do) <address>

– Shows object members

- MethodTable: address of type metadata
- EEClass: internal data for this type
- Size: size of instance (“sizeof” style)



```
0:000> !do 01d6ab74
Name: System.Threading.WaitOrTimerCallback
MethodTable: 6f413d2c
EEClass: 6f248174
Size: 32(0x20) bytes
 (C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll)
Fields:
      MT    Field   Offset                 Type VT     Attr    Value Name
6f430508  40000ff        4        System.Object  0 instance 01d6ab74 _target
6f42fd60  4000100        8 ...ection.MethodBase  0 instance 00000000 _methodBase
6f4331b4  4000101        c        System.IntPtr  1 instance   8609bc _methodPtr
6f4331b4  4000102       10        System.IntPtr  1 instance   66c040 _methodPtrAux
6f430508  400010c       14        System.Object  0 instance 00000000 _invocationList
6f4331b4  400010d       18        System.IntPtr  1 instance        0 _invocationCount
```

More Dumps

`!dumpmt <address>`

– Dump the metadata info given an MT address

`!dumpvc <mtaddress> <address>`

– Dump complex value types

`!dumpdomain`

– Show information on AppDomains

`!dumparray (!da) [options] <address>`

– Show array elements

`-details` (uses `!do` to show each element's data)

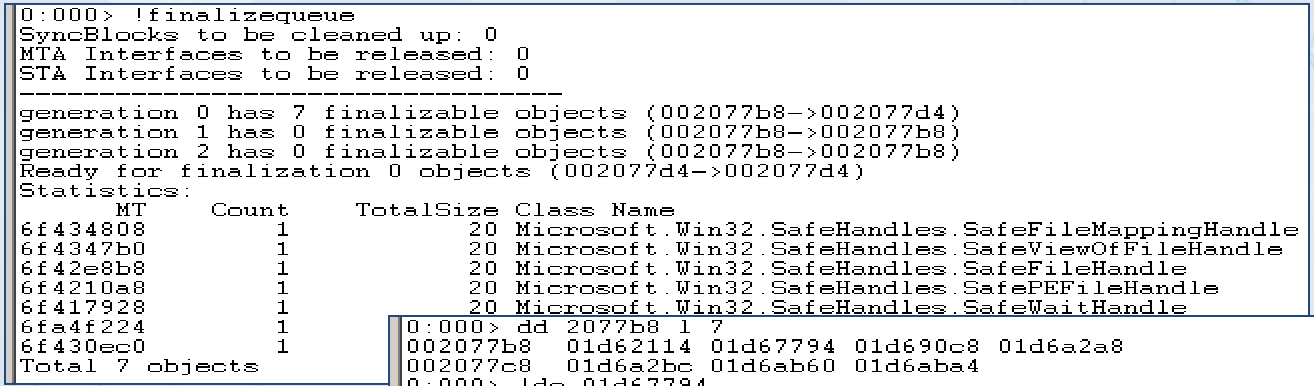
`-start <n> -length <m>` (shows only some elements)

`-nofields` (skip fields data)

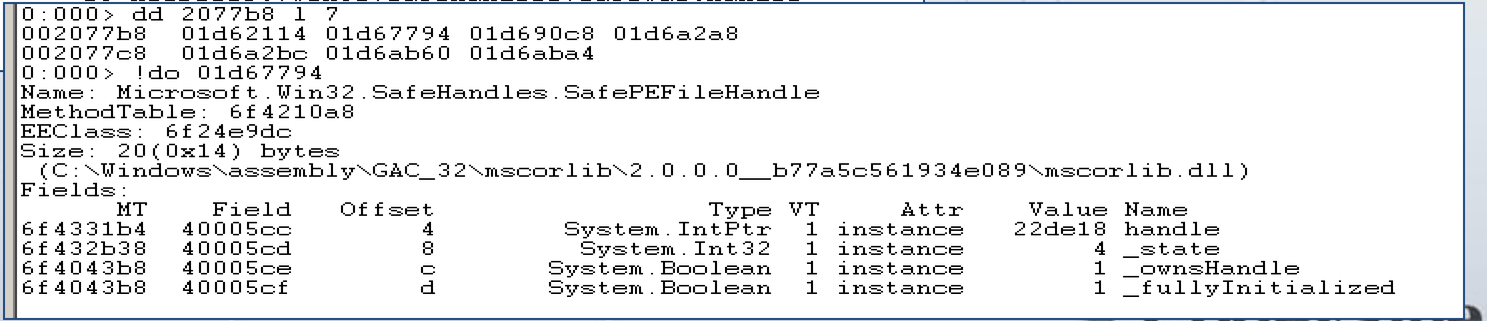
GC Finalization Queue

!FinalizeQueue

- Shows objects that have finalizers
- Can use the dd command to dump object instances and inspect with !do



```
0:000> !finalizequeue
SyncBlocks to be cleaned up: 0
MTA Interfaces to be released: 0
STA Interfaces to be released: 0
----------------------------------
generation 0 has 7 finalizable objects (002077b8->002077d4)
generation 1 has 0 finalizable objects (002077b8->002077b8)
generation 2 has 0 finalizable objects (002077b8->002077b8)
Ready for finalization 0 objects (002077d4->002077d4)
Statistics:
      MT    Count    TotalSize Class Name
6f434808        1           20 Microsoft.Win32.SafeHandles.SafeFileMappingHandle
6f4347b0        1           20 Microsoft.Win32.SafeHandles.SafeViewOfFileHandle
6f42e8b8        1           20 Microsoft.Win32.SafeHandles.SafeFileHandle
6f4210a8        1           20 Microsoft.Win32.SafeHandles.SafePEFileHandle
6f417928        1           20 Microsoft.Win32.SafeHandles.SafeWaitHandle
6fa4f224        1
6f430ec0        1
Total 7 objects
```



```
0:000> dd 2077b8 l 7
002077b8  01d62114 01d67794 01d690c8 01d6a2a8
002077c8  01d6a2bc 01d6ab60 01d6aba4
0:000> !do 01d67794
Name: Microsoft.Win32.SafeHandles.SafePEFileHandle
MethodTable: 6f4210a8
EEClass: 6f24e9dc
Size: 20(0x14) bytes
 (C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll)
Fields:
      MT    Field   Offset                 Type VT     Attr    Value Name
6f4331b4  40005cc        4        System.IntPtr  1 instance   22de18 handle
6f432b38  40005cd        8         System.Int32  1 instance        4 _state
6f4043b8  40005ce        c       System.Boolean  1 instance        1 _ownsHandle
6f4043b8  40005cf        d       System.Boolean  1 instance        1 _fullyInitialized
```

GC Heaps Data

`!eeheap -gc`

– Shows GC heap info

`!DumpHeap [options]`

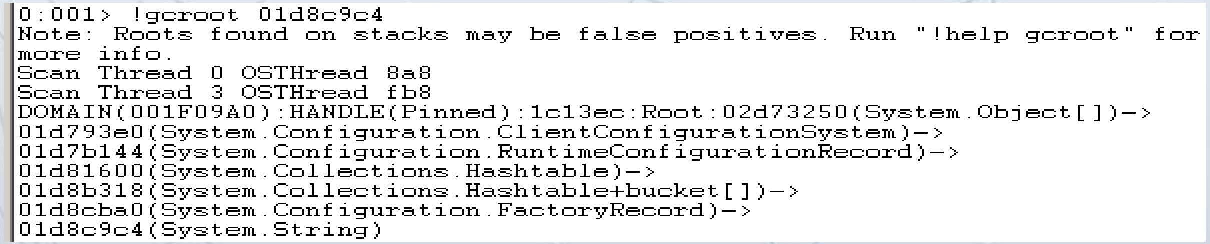
- `-stat` (show statistics of how many objects of each type exist on the heap)
- `-strings` (show strings only)
- `-short` (shows object addresses only)
- `-min n` (shows object whose size is at least n bytes)
- `-max n` (can also combine with `-min`)
- `-mt <address>` (shows only object with the specified method table value)
- `-type <typename>` (case sensitive partial match – no need for `-mt`)
- `[<from> <to>]` (shows only the objects residing in this address range)

GC Roots

!gcroot [options] <address>

- Lists the objects referencing this object up to a root
- If ends in a domain handle, its state is reported
- Otherwise, will be eligible for GC after stack reference goes out of scope

-nostacks (don't check stack objects)



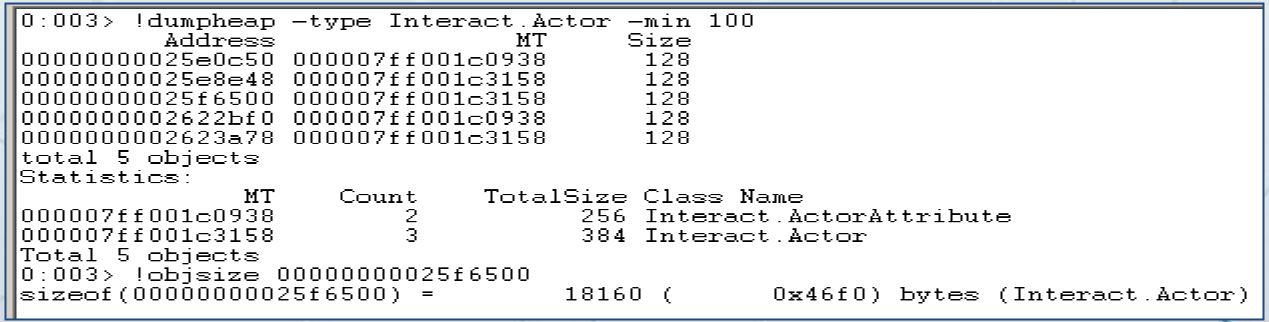
```
0:001> !gcroot 01d8c9c4
Note: Roots found on stacks may be false positives. Run "!help gcroot" for
more info.
Scan Thread 0 OSTHread 8a8
Scan Thread 3 OSTHread fb8
DOMAIN(001F09A0):HANDLE(Pinned):1c13ec:Root:02d73250(System.Object[])->
01d793e0(System.Configuration.ClientConfigurationSystem)->
01d7b144(System.Configuration.RuntimeConfigurationRecord)->
01d81600(System.Collections.Hashtable)->
01d8b318(System.Collections.Hashtable+bucket[])->
01d8cba0(System.Configuration.FactoryRecord)->
01d8c9c4(System.String)
```

GC Handle States

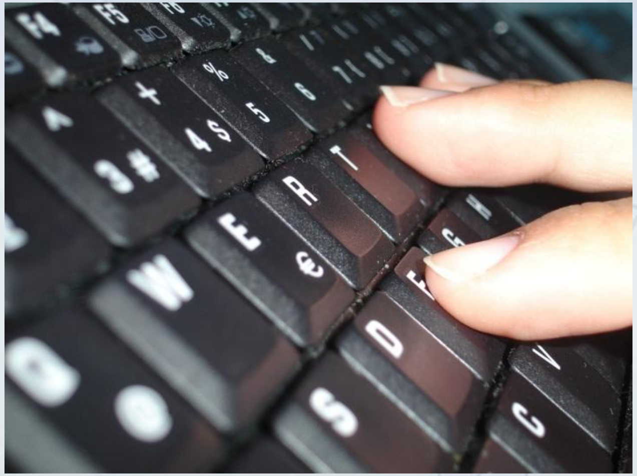
Code	Meaning	Description
WeakLn	Long lived weak handle	Tracked handle until reclaimed. Tracking occurs through finalization and across resurrections
WeakSh	Short lived weak handle	Tracks object until the first time it's unreachable
Strong	Strong handle	A normal object reference
Pinned	Pinned handle	An strong object reference that cannot move in memory during GC
RefCnt	Referenced counted handle	Strong handle when ref count greater than zero, otherwise becomes a weak handle
AsyncPinned	Asynchronous pinned handle	Used internally by the CLR
Unknown	Unknown handle	Cannot report handle

Object Size

- Object size listed with `!dumpheap` is the immediate fields size (where a reference type is 4 or 8 bytes)
- Full object size including object sizes of all contained objects can be viewed with `!ObjSize`



```
0:003> !dumpheap -type Interact.Actor -min 100
         Address               MT     Size
00000000025e0c50 000007ff001c0938      128
00000000025e8e48 000007ff001c3158      128
00000000025f6500 000007ff001c3158      128
0000000002622bf0 000007ff001c0938      128
0000000002623a78 000007ff001c3158      128
total 5 objects
Statistics:
              MT    Count    TotalSize Class Name
000007ff001c0938        2          256 Interact.ActorAttribute
000007ff001c3158        3          384 Interact.Actor
Total 5 objects
0:003> !objsize 00000000025f6500
sizeof(00000000025f6500) =        18160 (       0x46f0) bytes (Interact.Actor)
```



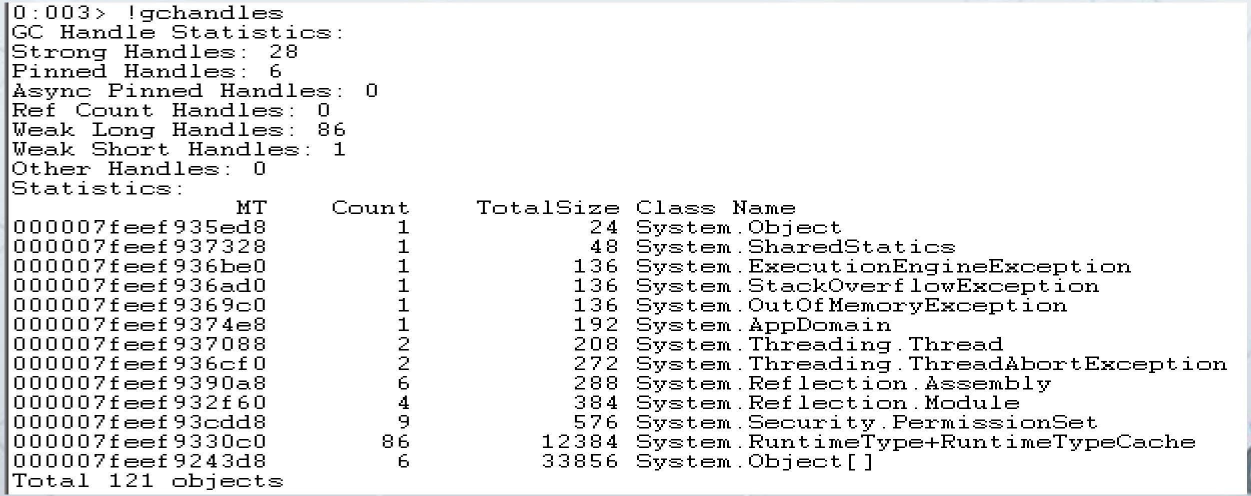
LOH fragmentation Demo

GC Handles

- GC handles are stored on an AppDomain basis
- Can be created programmatically by calling `GCHandle.Alloc`

!GChandles [-perdomain]

- – Shows all GC handles in use in the process

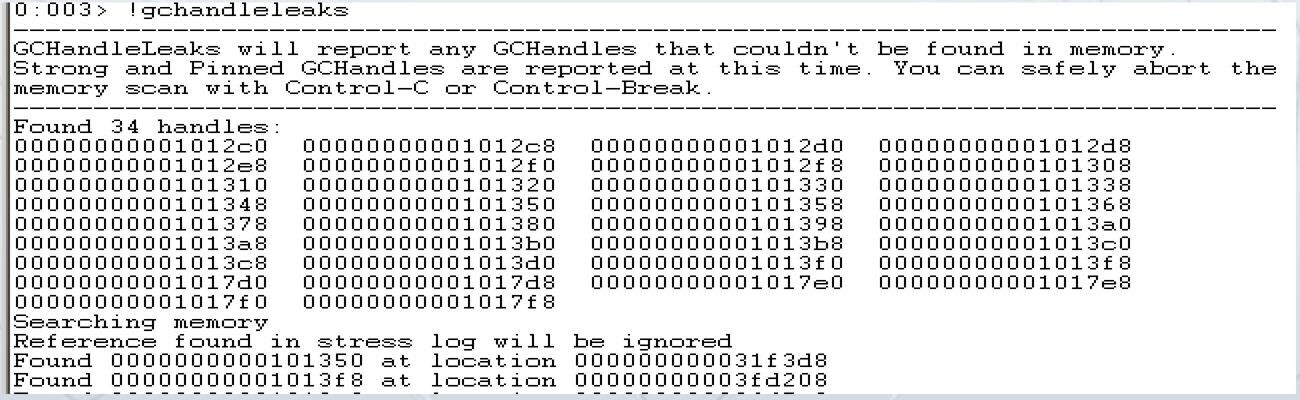


```
0:003> !gchandles
GC Handle Statistics:
Strong Handles: 28
Pinned Handles: 6
Async Pinned Handles: 0
Ref Count Handles: 0
Weak Long Handles: 86
Weak Short Handles: 1
Other Handles: 0
Statistics:
              MT    Count    TotalSize Class Name
000007feef935ed8        1           24 System.Object
000007feef937328        1           48 System.SharedStatics
000007feef936be0        1          136 System.ExecutionEngineException
000007feef936ad0        1          136 System.StackOverflowException
000007feef9369c0        1          136 System.OutOfMemoryException
000007feef9374e8        1          192 System.AppDomain
000007feef937088        2          208 System.Threading.Thread
000007feef936cf0        2          272 System.Threading.ThreadAbortException
000007feef9390a8        6          288 System.Reflection.Assembly
000007feef932f60        4          384 System.Reflection.Module
000007feef93cdd8        9          576 System.Security.PermissionSet
000007feef9330c0       86        12384 System.RuntimeType+RuntimeTypeCache
000007feef9243d8        6        33856 System.Object[]
Total 121 objects
```

GC Handle Leaks

!GCHandleLeaks

- Searches for leaking GC handles
  - Searches memory
  - May not locate some leaks (rare)
    - Can compare with previous checks
- Looks for Strong and Pinned handles only



```
0:003> !gchandleleaks
-------------------------------------------------------------------------------
GCHandleLeaks will report any GCHandles that couldn't be found in memory.
Strong and Pinned GCHandles are reported at this time. You can safely abort the
memory scan with Control-C or Control-Break.
-------------------------------------------------------------------------------
Found 34 handles:
00000000001012c0  00000000001012c8  00000000001012d0  00000000001012d8
00000000001012e8  00000000001012f0  00000000001012f8  0000000000101308
0000000000101310  0000000000101320  0000000000101330  0000000000101338
0000000000101348  0000000000101350  0000000000101358  0000000000101368
0000000000101378  0000000000101380  0000000000101398  00000000001013a0
00000000001013a8  00000000001013b0  00000000001013b8  00000000001013c0
00000000001013c8  00000000001013d0  00000000001013f0  00000000001013f8
00000000001017d0  00000000001017d8  00000000001017e0  00000000001017e8
00000000001017f0  00000000001017f8
Searching memory
Reference found in stress log will be ignored
Found 0000000000101350 at location 00000000031f3d8
Found 00000000001013f8 at location 00000000003fd208
```

General Heap Check

- When working heavily with native code, managed heap might be corrupt

!VerifyHeap

  – Checks every object on the heap to make sure all fields point to valid objects
  – Displays nothing if all is ok

CLR Exceptions

- Break on a CLR exception

```
!StopOnException (!soe) [options] <exception_type>
    <temp_reg#>
```

- Options
  - `-create` (first chance)
  - `-create2` (second chance)
  - `-derive` (stop on derived exceptions as well)
- `temp_reg` is 0 to 19 ($t0 to $t19 pseudo registers)
- Examples

```
!soe -create System.ArgumentException 1
!soe -create -derived System.OutOfMemoryException 2
```

Displaying Exception Info

!PrintException (**!pe**) **[-nested] [addr]**

- Without arguments, prints the last exception on the current thread
- With **–nested**, prints inner exception data
- With address, interprets the object as an exception object

Setting Managed Breakpoints

!bpmd <module> <method_name>

!bpmd –md <address>

- <moudle> must include extension
- Method name must be fully qualified
- If overloads exist, breaks on all of them
- <address> is method table address
  - Can be obtained with !dumpmt
- Examples

```
!bpmd mscorlib.dll System.Threading.Thread.Join
!bpmd MyApp.Exe Program.Main
```

Thread Synchronization

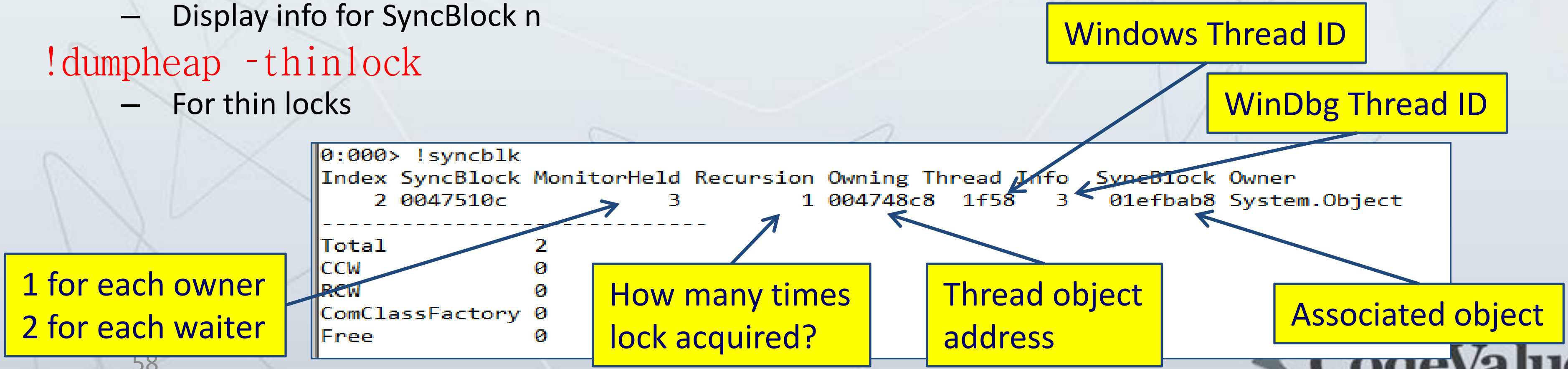
- Sync Block
  - A data structure used to hold extra information that is not needed for every object
  - Maintains a lock object used behind the scenes by `Monitor.Enter/TryEnter/Exit`

`!SyncBlk`
  - Displays SyncBlocks held

`!SyncBlk -all`
  - Display all SyncBlocks

`!SyncBlk n`
  - Display info for SyncBlock n

`!dumpheap -thinlock`
  - For thin locks



```
0:000> !syncblk
Index SyncBlock MonitorHeld Recursion Owning Thread Info  SyncBlock Owner
    2 0047510c            3         1 004748c8  1f58   3   01efbab8 System.Object
-----------------------------
Total           2
CCW             0
RCW             0
ComClassFactory 0
Free            0
```

The SOSEX Extension

- Supplements the SOS extension with new commands
  - Created by Steve Johnson (http://www.stevestechspot.com)
  - Version 2.0 released on March 7th, 2009
- Commands
  - `!dlk` – displays sync block deadlocks (those caused by **Monitor.Enter/Exit**/ lock in C#, **ReaderWriteLock(Slim)** and Win32 Critical Sections)
  - `!dumpgen` – displays the objects in the specified generation (3=large object heap)
  - `!gcgen` – displays the generation of the specified object

More SOSEX Commands

- `!mbp`, `!mbd`, `!mbe`, `!mbc`, `!mbl` – set, enable, disable, clear and list managed breakpoints (similar to their native counterparts)
- `!mbm` – set a managed breakpoint on a method matching the specified filter
- `!mdt` – display the contents of an object or type
- `!mx` – display managed types/methods/fields matching the specified filter
- `!refs` – display all references from/to the specified object

Minidump Files

- A minidump is a snapshot of a process
  - May be created at any time, not just when a process crashes
- Minidump types
  - Kernel minidumps (not relevant for this course)
  - Basic (usually enough for native processes)
  - Full (required to get useful info for managed processes)
- Minidump creation

```
.dump [options] <filename>
```

  - On Vista, 2008 and up can use Task Manager
  - ADPlus
  - ProcDump from SysInternals

Dump File Creation (1)

- WinDbg
  `.dump [options] filename`
  - Options
    `/ma` (full minidump)
    `/o` (overwrite existing file)
    `/u` (ensure unique filename)
- ADPlus
  - Hang mode – noninvasive attach
  - Crash mode – attaches the CDB debugger
  - Common options
    `-hang` (hang mode)
    `-crash` (crash mode)
    `-pn` (specify process name including extension)
    `-p` (specify process ID)
    `-c` (specify XML config file to read options from)
    `-quiet` (don't show various confirmation dialogs)

CodeValue

Dump File Creation (2)

- Using ProcDump
  - **-c**
    - Capture a dump when CPU consumption is above a certain percent (-c)
  - **-s**
    - CPU usage time to trigger a dump (used with –c)
  - **-n**
    - Captures a number of dumps (used with –c or –s)
  - **-h**
    - Capture a dump when a window becomes unresponsive
  - **-e**
    - Captures a dump when an exception occurs (optionally first chance)
  - **-t**
    - Captures a dump when the process terminates

Opening a Minidump

- In WinDbg, **File->Open Crash Dump…**

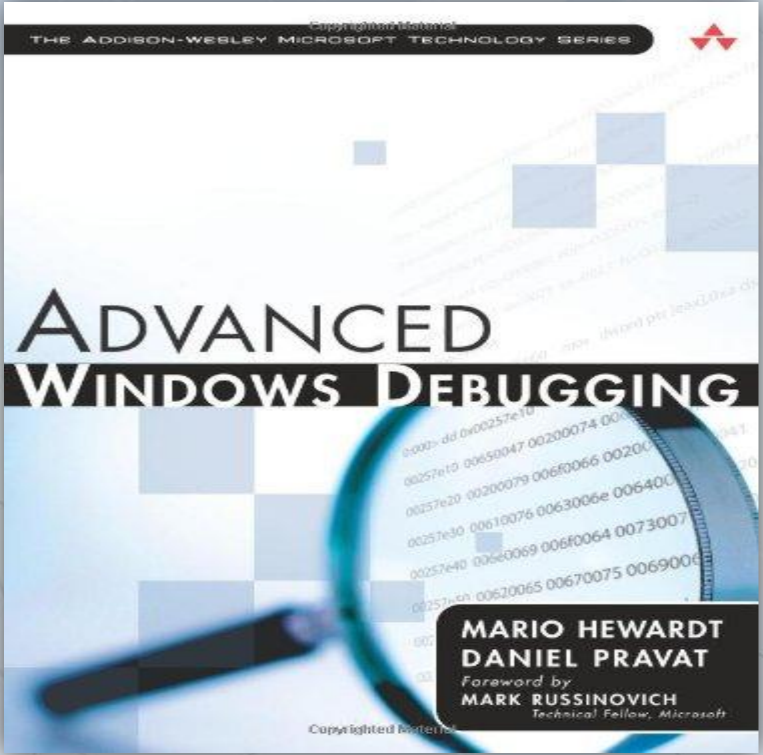
WinDbg –z <dump_file>

- Issue the "magical" command

!analyze –v

- Can use most other WinDbg/SOS commands
- Can open a dump file in Visual Studio too (File -> Open Project/Solution…)

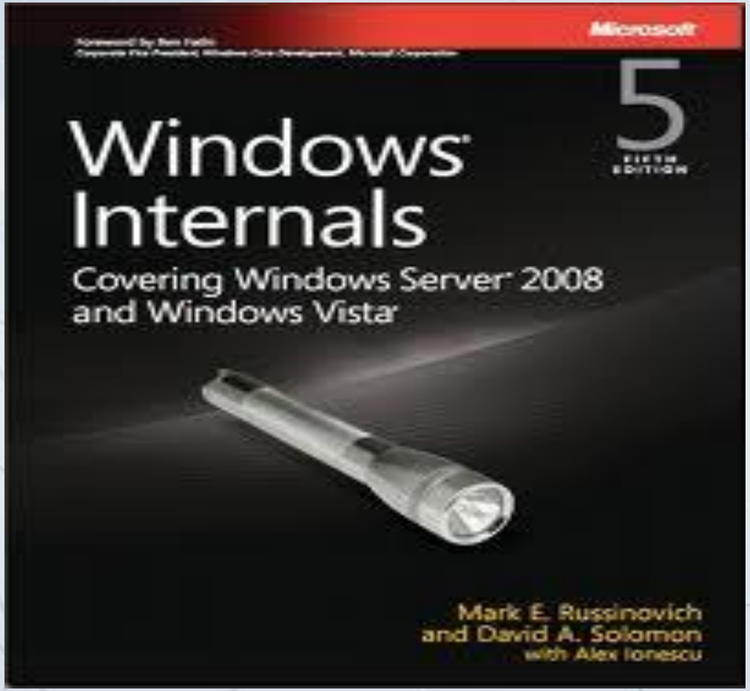
Read and Use This Excellent Book


The Addison-Wesley Microsoft Technology Series
Advanced Windows Debugging
Mario Hewardt
Daniel Pravat
Foreword by Mark Russinovich
Technical Fellow, Microsoft

The Addison-Wesley Microsoft Technology Series
Advanced .NET Debugging
Mario Hewardt
Foreword by Patrick Dussud
Technical Fellow, CLR, .NET FX, Microsoft Corporation

CodeValue

Get More Details About the System



Foreword by Ben Fathi
Corporate Vice President, Windows Core Development, Microsoft Corporation

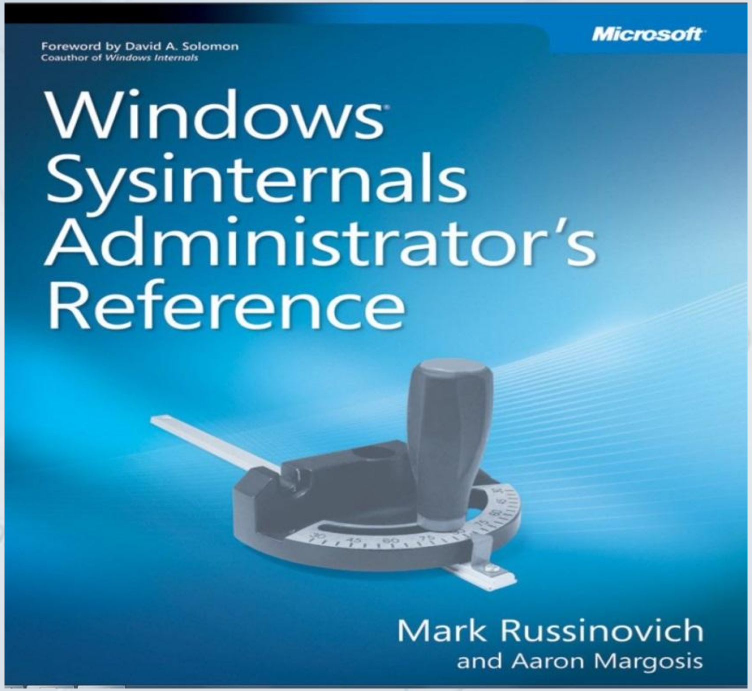
Microsoft

5 FIFTH EDITION

Windows Internals

Covering Windows Server 2008 and Windows Vista

Mark E. Russinovich
and David A. Solomon
with Alex Ionescu



Foreword by David A. Solomon
Coauthor of *Windows Internals*

Microsoft

Windows Sysinternals Administrator's Reference

Mark Russinovich
and Aaron Margosis

CodeValue

Resources

- Debugging Tools For Windows Documentation
  http://msdn.microsoft.com/en-us/library/ff551063(v=VS.85).aspx
- Windows NT based OS Data Structures
  http://msdn.moonsols.com/
- NT Debugging Blog
  http://blogs.msdn.com/b/ntdebugging/

Q & A

Only Production debugging can prevent explosions

Thank You

Stas Shteinbook

stass@CodeValue.net