Debugging

Agenda

- **Memory dumps**
- **Tools for creating memory dumps**
- **Analysing memory dumps**
  - Exceptions
  - Threads
  - Memory

Beyond Interactive Debugging

- **Interactive debugging not the only debugging activity**
  - Other forms of analysis critical debugging skills
- **Dump analysis**
  - Snapshot of state of process
  - Provides insight beyond that of Visual Studio

Memory Dumps

- **Memory dumps snapshot process address space and can contain**
  - State of all threads
  - State of all objects
  - Code for all modules
- **Two main kinds of memory dumps**
  - Full dump (total address space)
  - Pure Minidump (mostly just stack information)
- **Only full dumps are useful for managed code**
  - Although some tools will add extra info to a minidump to make them useful for managed code (clrdump.exe for example)

Generating Dump Files

- **Visual Studio**
  - Save Dump As … during debugging
- **Task Manager**
  - Create Dump File
- **Debugging Tools for Windows**
  - ADPLUS.VBS
- **DebugDiag**
- **Clrdump.exe**
  - 3rd party tool for creating extended minidumps
- **Programmatically**
  - Clrdump.dll CreateDump

DebugDiag

- **Main data collection tool used by Microsoft PSS**
  - Will ask you to install on production machines and capture dumps with it
- **Rich configuration**
  - Rule based dump capture
  - Injects DLL to track native memory allocations
- **Analysis**
  - Can analyse dump files and identify common problems
- **Installs as service**
  - Configuration tool
- **Analysis only version**
  - Allows the analysis of captured dump files

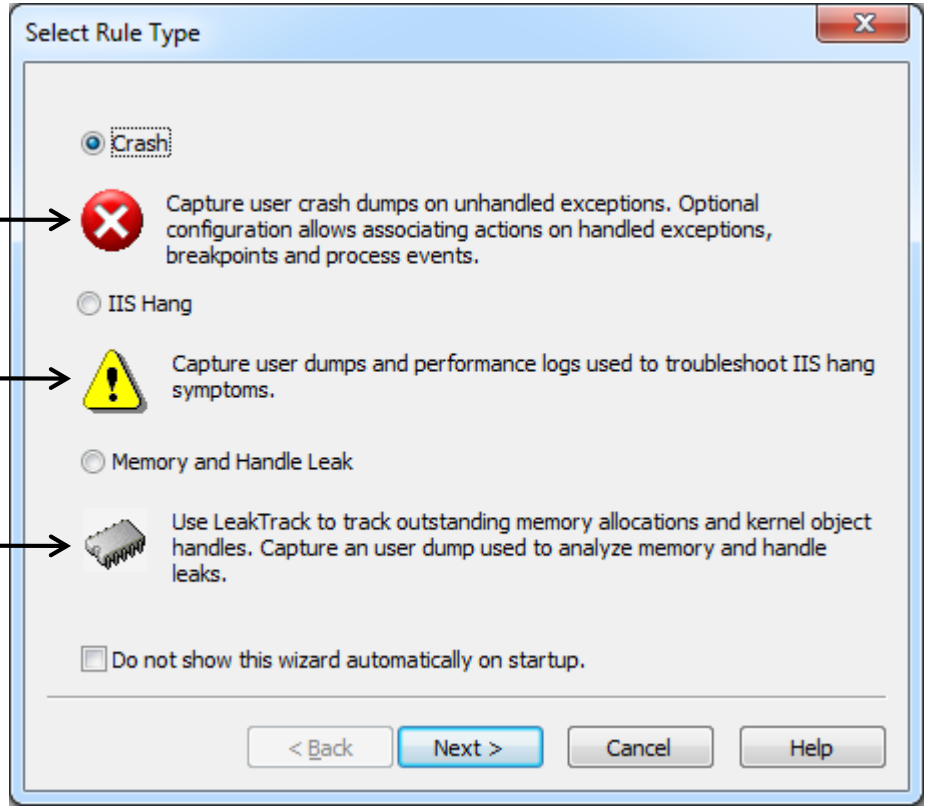
Configuring DebugDiag Rules

- **New rule wizard**

Program terminates abnormally

IIS hanging or responding slowly

Memory and resource leaks



Debugging Tools for Windows

- **Toolkit for native debugging**
  - Ships as part of Windows SDK or DDK
- **ADPLUS allows capture of full dump files**
  - VBScript created to make configuration of cdb.exe easier
- **Two modes**
  - Crash Mode
  - Hang Mode
- **Crash Mode – dump on abnormal termination of process**

```
adplus -crash -o c:\temp -pn MyApp.exe
```

- **Hang Mode – snapshot process at current point in time**

```
adplus -hang -o c:\temp -pn MyApp.exe
```

Capturing Dump on Application Start

- **Startup problems can be tricky to capture dump**
  - App terminates before ability to run tool
  - Need to start app under control of tool
- **ADPLUS Spawning Mode**

```
adplus -crash -o c:\temp -sc MyApp.exe
```

Analysing Dump Files

- **Classic tool is windbg.exe**
  - Part of Debugging Tools for Windows
  - Requires plugin to analyse managed code
- **Visual Studio 2010 can analyse managed code**
  - Only works with .NET 4.0 and beyond
  - Not as powerful as windbg.exe
  - Familiar tool and may provide enough info to solve problem

Visual Studio Dump Analysis

- **Open dump file**
- **Select mixed mode debugging**
  - Gives a debugging snapshot – cannot move forwards or backwards
  - May have to set up symbol paths
- **Can match dump to code**
  - Source code is not contained in dump so must be separately available
- **Debug windows helpful**
  - Threads
  - Call Stack
  - Parallel Stacks (Thread Mode)

Using Windbg

- **Open dump file in windbg**
  - CTRL-D
- **Must load extension to analyse managed code**
  - PSSCOR2
  - SOS (Son of Strike)
  - SOSEX
- **SOS version dependent on CLR version**
  - Need to ensure correct version loaded

```
.loadby sos clr
```

← .NET 4.0

```
.loadby sos mscorwks
```

← .NET 3.5 and earlier

SOS.DLL and SOSEX.DLL

- **Windbg plugins**
  - Understands data structures that support CLR
  - Able to provide insight not available through Visual Studio
- **SOS.DLL**
  - Core module for managed code analysis
- **PSSCOR2.DLL**
  - Superset of 3.5 SOS
  - Extra commands for ASP.NET debugging
  - Only supports 3.5 and below
- **SOSEX.DLL**
  - Extended functionality to assist diagnosis of common issues
  - Available from http://www.stevestechspot.com/

Improving Dump Data

- **JIT Compiler optimization can mask causes of issues**
  - Inlining
  - Passing parameters via registers rather than stack
- **For release builds set up .ini file in same directory as application to disable optimization**
  - <AppName>.ini
  - <DllName>.ini

```
[.NET Framework Debugging Control]
GenerateTrackingInfo=1
AllowOptimize=0
```

Analysing Crash Dump Basics

- **Windbg and SOS are main tools**
  - Visual Studio 2010 can be used for some issues
- **Must make sure to use correct version of windbg and SOS**
  - 64bit and 32bit incompatible as native debugging infrastructure
  - Must load version of SOS based on version of CLR

Where to Start with Dump Analysis?

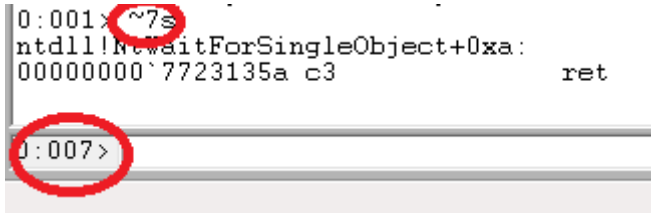
- **Often useful to have some context before you start**
  - Helps focus analysis
- **“Bad” Performance can be caused by a lot of things**
  - Thread contention / deadlock
  - GC thrashing
  - Large numbers of exceptions being thrown
  - Network traffic
  - Etc
- **Perfmon has counters that allow insight of causes**
  - Learn to love the .NET categories; they are your friends

Useful High level SOS Commands

- **Two commands give a lot of context quickly**
  - !threads
  - !DumpHeap - stat
- **!threads shows all the managed threads**
  - Native and managed thread ids
  - The AppDomain the thread is running in
  - COM Apartment state
  - Locks held
  - Any Unhandled exception objects on the thread
- **!DumpHeap –stat shows all types on managed heap**
  - Number of objects
  - Total amount of memory used by this type
  - Remember that objects may not be reachable but just not collected yet

Switching focus between threads

- **Some commands are thread specific**
  - E.g. !clrstack
- **Need to be able to change which thread command executes on**
  - Prefix of ~#e (# is thread number) executes command against specified thread
  - ~*e executes commands against all threads
  - ~#s changes the thread focus for all subsequent commands (new focus indicated in windbg)



Debugging Unhandled Exceptions

- **Need to set up a crash dump**
  - DebugDiag
  - ADPLUS -crash
- **Unhandled exceptions will terminate process and generate dump file**
- **!threads is starting point**

```
0:000> !threads
PDB symbol for clr.dll not loaded
ThreadCount:      2
UnstartedThread:  0
BackgroundThread: 1
PendingThread:    0
DeadThread:       0
Hosted Runtime:   no
                        Lock
       ID  OSID  …      Count APT Exception
   0    1  1b98  …        0   MTA UploadLib.InvalidUploadException …
   2    2  1b7c  …        0   MTA (Finalizer)
```

!PrintException Shows more Detail

- **!PrintException (!pe) shows full exception details for current thread**
  - May need to switch thread focus
  - -nested includes details of all nested exceptions

```
0:000> !pe
Exception object: 000000000226a6a8
Exception type:   UploadLib.InvalidUploadException
Message:          Upload file does not exist
InnerException:   System.IO.FileNotFoundException, Use !PrintException
000000000226a408 to see more.
StackTrace (generated):
    SP               IP               Function
    00000000002AC830 000007FF001502E1 UploadLib!UploadLib.Program.UploadFile …
    00000000002AEC30 000007FF00150151 UploadLib!UploadLib.Program.Main …
```

Don’t Panic!

- `!DumpHeap -stat -type Exception` shows many exception objects
  - `OutOfMemoryException`
  - `ExecutionEngineException`
  - `ThreadAbortException`
  - `StackOverflowException`
- System critical exceptions pre-allocated

```
0:000> !DumpHeap -stat -type Exception
total 0 objects
Statistics:
              MT    Count    TotalSize Class Name
000007ff000341f8        1          160 UploadLib.InvalidUploadException
000007fedd0b7090        1          160 System.ExecutionEngineException
000007fedd0b7008        1          160 System.StackOverflowException
000007fedd0b6f80        1          160 System.OutOfMemoryException
000007fedd0b6d28        1          160 System.Exception
000007fedd0ce630        1          184 System.IO.FileNotFoundException
000007fedd0b7118        2          320 System.Threading.ThreadAbortException
Total 8 objects
```

Gaining Insight into the Call Stack

- **Exceptions contain stack trace**
  - Only the calls not the parameters
- **!clrstack –a shows calls and parameters for current thread**
  - Remember to use the .ini file to show parameters correctly and turn off inlining
- **!DumpObject (!do) shows the state of an object from the address**
  - Can see actual parameter objects this way

Debugging Threads

- **Multithreaded application suffer from specific problems**
  - Deadlocks
  - Runaway threads
  - Race conditions
- **Race conditions hard to diagnose generally**
  - Static analysis of the code (what-if?)
  - Tools such as Chess from MSR can help

Diagnosing Deadlocks

- **Multiple threads enter deadly embrace**
  - Requiring locks that other threads already own
- **If lucky the whole application hangs**
  - Requires that every thread becomes involved in deadlock
- **Server and parallel algorithms generally just perform badly**
  - Less threads available to do work
- **Cannot use crash dump**
  - Application does not crash
- **Use hang dump instead**
  - Snapshot of process
  - May need to take more than one to diagnose problem

!syncblk

- **A SyncBlock is the internal name for a monitor**
  - Monitor interaction is behind the lock keyword
- **!syncblk shows all currently owned monitors**

```
0:000> !syncblk
Index SyncBlock MonitorHeld Recursion Owning Thread Info SyncBlock Owner
    2 0057ad78            3         1 0054ffb0 1e70    0   02801948 System.Object
    3 0057adc8            3         1 00580ae0 20c4    4   02801960 System.Object
```

- **Deadlocked threads will be in Monitor.Enter**

```
0:000> !clrstack
OS Thread Id: 0x1e70 (0)
Child SP          … Call Site
001fe780 772318ca …
001fe928 772318ca …
001fe8d8 772318ca … System.Threading.Monitor.Enter(System.Object)
001fea20 001402f0 … Deadlock.Program.Main(System.String[])
001feea0 df1010b4
```

Which Monitor is Blocking

- **Monitor.Enter does not show object parameter**
- **Use !dso to show the objects on the stack**
  - The one passed to Monitor.Enter will be at the top

```
0:000> !dso
OS Thread Id: 0x1e70 (0)
RSP/REG  Object   Name
rcx      02801960 System.Object
001FE770 02801960 System.Object
001FE878 02801960 System.Object
001FEA48 028019b8 System.Threading.Thread
001FEA58 02801948 System.Object
001FEA60 02801960 System.Object
001FEA70 02801978 System.Threading.ThreadStart
001FEA78 02801978 System.Threading.ThreadStart
001FEA80 028019b8 System.Threading.Thread
001FEA88 028019b8 System.Threading.Thread
001FEA90 02801948 System.Object
001FEA98 02801960 System.Object
001FEAD0 02801928 System.Object[]    (System.String[])
001FEC38 02801928 System.Object[]    (System.String[])
001FEE08 02801928 System.Object[]    (System.String[])
```

Let SOSEX.dll Take the Strain

- **SOSEX has extended functionality over SOS**
  - http://www.stevestechspot.com/
- **!dlk shows deadlocks**
  - Works in most situations
- **!rwlock shows ReaderWriterLock**

```
0:000> !dlk
Examining SyncBlocks...
Scanning for ReaderWriterLocks...
Scanning for lock holders on ReaderWriterLocks...
Scanning for threads waiting on SyncBlocks...
Scanning for threads waiting on ReaderWriterLocks...
Deadlock detected:
CLR thread 0x1 owns SyncBlock 0057ad78 OBJ:02801948[System.Object]
             is waiting for SyncBlock 0057adc8 OBJ:02801960[System.Object]
CLR thread 0x3 owns SyncBlock 0057adc8 OBJ:02801960[System.Object]
             is waiting for SyncBlock 0057ad78 OBJ:02801948[System.Object]
CLR Thread 0x1 is waiting at Deadlock.Program.Main(System.String[]) ...
CLR Thread 0x3 is waiting at Deadlock.Program.DoWork() ...

1 deadlock detected.
```

Runaway Threads

- **A runaway thread is one that is spinning on a CPU core**
  - Caught in tight loop
- **Can be difficult to spot on multi-core machines**
- **!runaway show stats of CPU usage for all threads**
  - Runaway threads commonly use significantly more CPU than other threads
- **Can examine stack of runaway thread for insight into what thread is doing**
  - ~#e !clrstack -a

Debugging Memory Issues

- **Three main memory related issues for managed code**
  - Objects staying rooted unintentionally
  - Objects being collected from expensive collections
  - Excessive number of finalizers executing
- **Expensive collection best diagnosed with Perfmon and memory profiling**
- **Be careful if GC in progress**
  - Snapshot will be useless for memory issues
  - **`!threads -special`** will show **`GC SuspendEE`** if GC in progress

Why Isn't my Object being Collected?

- **Extant live root to object**
  - Collections and events main culprits
- **!DumpHeap –stat shows objects on heap**
  - Ascending order of memory consumption
  - MT column identifies MethodTable which identifies type

```
0:000> !dumpheap -stat

Statistics:
              MT    Count    TotalSize Class Name
000007fedbe43378        1           24 System.Security.HostSecurityManager
000007fedbe42838        1           24 System.Collections.Generic.Object …
000007fedbe41220        1           24 System.Security.Permissions.UIPermission

...

000007fedbe36960      470        20352 System.String
000007fedbe3ae68       66        38256 System.Object[]
Total 804 objects
```

Zeroing-in on an Instance

- **Can find all instances of a type**
  - !Dumpheap –MT <MethodTable address>

```
0:000> !dumpheap -MT 000007fedbe3ae68
         Address               MT     Size
00000000026e19e8 000007fedbe3ae68      176
00000000026e7508 000007fedbe3ae68      112
00000000026e75b8 000007fedbe3ae68       40
00000000026e75e0 000007fedbe3ae68       64
00000000026e7620 000007fedbe3ae68       40
00000000026e7668 000007fedbe3ae68      216
00000000026e7fa8 000007fedbe3ae68       48
00000000026e7fd8 000007fedbe3ae68       40
00000000026e8000 000007fedbe3ae68       40
…
```

Finding the Live Root

- **Can track back to the root from an object**
  - !gcroot <object address>

```
0:000> !gcroot 14385190
...
RSP:16ed80:Root:
026918e0(System.Collections.Generic.List`1[[Rooting.ObjectManager, Rooting]])->
  02691c30(System.Object[])->
  02691d98(Rooting.ObjectManager)->
  14385190(System.Int32[])
...
```

Finalization Issues

- **Can look at the stats of the finalization infrastructure**
  - !FinalizeQueue

```
0:000> !finalizequeue
PDB symbol for clr.dll not loaded
SyncBlocks to be cleaned up: 0
MTA Interfaces to be released: 0
STA Interfaces to be released: 0
-----------------------------------
generation 0 has 131072 finalizable objects (000000003afe0058->000000003b0e0058)
generation 1 has 0 finalizable objects (000000003afe0058->000000003afe0058)
generation 2 has 3 finalizable objects (000000003afe0040->000000003afe0058)
Ready for finalization 8780389 objects (000000003b0e0058->000000003f3dd380)
```

Other Useful Commands

- **Can extract code from dump file**
  - Debugging issues involving 3rd party components
  - Confirming which version of code was executing
- **lm (no !) list modules and base addresses**

```
0:000> lm
start             end                 module name
00000000`00b70000 00000000`00b78000   Rooting     (deferred)
00000000`76fc0000 00000000`770ba000   user32      (deferred)
00000000`770c0000 00000000`771df000   kernel32    (deferred)
00000000`771e0000 00000000`77389000   ntdll       (export symbols)      ntdll.dll
000007fe`dcbf0000 000007fe`dde67000   mscorlib_ni    (deferred)
000007fe`df0c0000 000007fe`dfa25000   clr         (export symbols)       clr.dll
```

- **Can extract loaded module**
  - !savemodule <start address> <file name>
  - Examine in reflector to find problem

Summary

- Interactive debugging not always possible / appropriate
- Dump files can provide insight not available through interactive debugging
- Many ways to create dump files
- Dump file analysis involves arcane but powerful tools