Sasha Goldshtein
CTO, Sela Group
blog.sashag.net
@goldshtn

Monitoring .NET Performance with Event Tracing for Windows

https://s.sashag.net/etwsdd
Agenda

- ETW on the performance monitoring spectrum
- Semantic logging
- Capturing and analyzing traces
- Advanced trace analysis with PerfView
- Memory leak analysis with PerfView
- Programmatic ETW recording and analysis
Introduction to ETW
Performance Monitoring Spectrum

- **Performance metrics and simulations**
- **Development-time profiling**
- **Performance/load tests**
- **Continuous low-overhead monitoring**
- **Production-time performance investigations**

**Today’s focus**
Problems with Traditional Profilers

- **Invasiveness**
  - Often requires restart or code injection

- **Overhead**
  - 2x slowdowns are not unheard of

- **Trace size**
  - Often not applicable for continuous monitoring for hours/days on end

- **Licensing costs**
  - Production mode or remote profiling mode not always available
Event Tracing for Windows

- High-performance facility for emitting 100K+ log events per second with rich payloads and stack trace support
- Used widely across Windows, .NET, drivers, services, third party components
ETW Participants

- A **provider** generates ETW events
- A **controller** starts and stops ETW collection
- A **consumer** logs, analyzes, or processes ETW events
Sample ETW Scenarios

- Profile an app in sampling mode
- Perform wait-time analysis
- Log disk accesses including stacks
- Log memory allocation statistics (.NET/C++)
- Log GC and JIT events
- Custom application event log
Semantic Logging
Semantic Logging

- Unstructured logs are a thing of the past
  - Require additional parsing, discipline, and tools to get through
- Semantic logging associates structure with each log message

<table>
<thead>
<tr>
<th>Unstructured</th>
<th>Structured</th>
</tr>
</thead>
</table>
| [572 09:04:10.112/@prod1] WARN Connection to cluster head node head1 lost, attempting reconnect in 2 seconds from prod2; this may be a temporary failure | TYPE = CONN_HEAD_LOST  
ID = 572  
TIME = 09:04:10.112  
SERVER = prod1  
HEAD_NODE = head1  
RECONNECT_IN = 00:00:02  
RECONNECT_FROM = prod2 |
ETW Events Have Structure

- Each ETW event has a well-defined payload
- Tools can use the payload format to easily consume and analyze large volumes of logging information

```xml
<Event MSec="5905.4352" PID="476" PName="explorer" TID="2352"
EventName="EdgeUi_Invoke_Mouse_HitSingleEdge"
TimeStamp="12/01/14 21:56:06.880587" ID="3205"
Level="Informational" CPU="7" EventIndex="27983"
ProviderName="Microsoft-Windows-Immersive-Shell" ...>
  <PrettyPrint>
    <Event ... EdgeUiComponent="3"/>
  </PrettyPrint>
...  
</Event>
```
Semantic Logging Application Block

🧩 A p&p project that logs events from an ETW provider to a database, text file, Azure storage, Elasticsearch, etc.
The EventSource class greatly simplifies ETW providers in .NET applications.

class PerformanceEventsSource : EventSource
{
    public void StartOperation(string op) { WriteEvent(1, op); }

    [Event(2, Message = "Completed \{0\} with result \{1\}"孛]
    public void StopOperation(string op, string result)
    {
        WriteEvent(2, op, result);
    }
}
Configuring SLAB

ConsoleLog.CreateListener().EnableEvents(
    OperationsEventSource.Log, EventLevel.LogAlways);

RollingFlatFileLog.CreateListener(
    "informational.log", 1000, "yyyy-MM-dd",
    RollFileExistsBehavior.Increment, RollInterval.Hour)
    .EnableEvents(OperationsEventSource.Log,
            EventLevel.Informational);

var listener = new ObservableEventListener();
listener.EnableEvents(OperationsEventSource.Log,
            EventLevel.LogAlways);
listener.LogToElasticsearch(Environment.MachineName,
    "http://localhost:9200", "slab", "logs",
    bufferingCount: 1);
Structured Payload Examples

```
EventId : 1, Level : Informational, Message : Operation 'processing payroll data' started, Payload : 
[operationName : processing payroll data], EventName : OperationStartedInfo, Timestamp : 2014-12-02T06:48:09.3074443Z, ProcessId : 10916, ThreadId : 10720

EventId : 2, Level : Informational, Message : Operation 'processing payroll data' ended with result '253 paychecks issued', Payload : [operationName : processing payroll data] [result : 253 paychecks issued], EventName : OperationEndedInfo, Timestamp : 2014-12-02T06:48:11.3236042Z, ProcessId : 10916, ThreadId : 10720
```

Server localhost:9200

```
GET _search?q=Payload_result:issued
```

```
{
  "_index": "slab-2014.12.02",
  "_type": "logs",
  "_id": "AUoJux57DehBd4Zxx3oP",
  "_score": 0.15342641,
  "_source": {
    "EventId": 2,
    "EventDate": "2014-12-02T06:41:05.6593392Z",
    "Keywords": 0,
    "ProviderId": "8beb246f-be30-59d4-45e1-4fd335ff514e",
    "ProviderName": "OperationsEventSource",
    "InstanceName": "SASHALAPTOPWIN8",
  }
}```
Trace Capturing and Analysis
ETW Tools

- **xperf.exe**: Command-line tool for ETW capturing and processing
- **wpr.exe**: Command-line and GUI for end users
- **wpa.exe**: Visual trace analysis tool
- **PerfView.exe**: Visual tool for capturing and recording ETW events from managed providers and the CLR
- **logman.exe, tracerpt.exe**: Built-in Windows tools for trace recording and formatting
Production Use

❖ All ETW tools are suitable for production use
❖ Some things to watch out for:
  ❖ Choose event providers carefully to minimize the performance impact on the system
  ❖ Capture to a circular log file to avoid running out of disk space
  ❖ Set triggers to stop collection (and keep all preceding events) when a critical event occurs
Capturing a Trace

Xperf

xperf -on DiagEasy
...
xperf -d diag.etl
Xperf Providers and Flags

Xperf makes it easy to capture a trace from the kernel provider using a set of flags and groups

Examples:

<table>
<thead>
<tr>
<th>Group</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>PROC_THREAD, LOADER, DISK_IO, HARD_FAULTS, PROFILE, MEMINFO, MEMINFO_WS</td>
</tr>
<tr>
<td>FileIO</td>
<td>PROC_THREAD, LOADER, DISK_IO, HARD_FAULTS, FILE_IO, FILE_IO_INIT</td>
</tr>
<tr>
<td>SysProf</td>
<td>PROC_THREAD, LOADER, PROFILE</td>
</tr>
</tbody>
</table>

- **Base** group includes flags for process/thread creation/deletion, file operations, and profiling.
- **FileIO** group includes flags for file operations and a 1ms sampling event for profiling.
- **SysProf** group includes flags for process/thread creation/deletion and profiling.
What’s In A Trace?

› A trace is a huge list of events
› Events have multiple columns (payload)
› Useless without additional processing
Trace Processing with Xperf

- I/O summary report per file
  
  xperf -i fileio.etl
  -o fileio.csv -a diskio
  -summary

- Interactive profiling report (for a specific process)
  
  xperf -i cpu.etl
  -o cpu.html -symbols
  -a stacks -process
  app.exe -butterfly
Managed Stacks

- To display managed stack traces correctly, additional CLR data is required.
- WPR & PerfView take care of this automatically.
- If using Xperf, see:
Trace Analysis with WPA

<table>
<thead>
<tr>
<th>List of graphs</th>
<th>Graph display</th>
<th>Ungrouped columns</th>
<th>Grouped columns</th>
<th>Grouping bar</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="List of graphs" /></td>
<td><img src="image2.png" alt="Graph display" /></td>
<td><img src="image3.png" alt="Ungrouped columns" /></td>
<td><img src="image4.png" alt="Grouped columns" /></td>
<td><img src="image5.png" alt="Grouping bar" /></td>
</tr>
</tbody>
</table>
Stack Summaries

➤ Learn how to read stack summaries
  ➤ Group by Stack column
  ➤ Expand “hot path”, like in profiler

➤ Stack resolution requires symbols (slow)
Collecting File I/O Information

Lab
PerfView

- ETW collection and analysis tool tailored for .NET applications (but not only)
- Can be used as a sampling profiler
- Can be used as an allocation profiler
- Can be used for heap snapshot analysis
Collecting Data with PerfView

**CLI**

PerfView run app.exe

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/MaxCollectSec:N</td>
<td>Stop collection after N seconds</td>
</tr>
<tr>
<td>/StartOnPerfCounter</td>
<td>Start/stop collection based on performance counter</td>
</tr>
<tr>
<td>/StopOnPerfCounter</td>
<td></td>
</tr>
<tr>
<td>/Providers=...</td>
<td>Restrict to specific set of providers</td>
</tr>
<tr>
<td>/OnlyProviders=...</td>
<td></td>
</tr>
<tr>
<td>/CircularMB:N</td>
<td>Circular logging N megabytes of newest events</td>
</tr>
</tbody>
</table>
PerfView Collection Options

Collecting ETW Data while running a command

This dialog gives displays options for collecting ETW profile data. The only required field is the 'Command' field and this is only necessary when using the 'Run' command.

If you wish to analyze on another machine use the Zip option when collecting data. See Collecting ETW Profile Data for more.

Command: C:\\courses\\NET Performance\\Exercises\\AllocationProfiling\\JackCompiler.exe
Data File: C:\\Temp\\PerfViewData.etl

Advanced Options

- Kernel Base: check box
- CPU Samples: check box
- Page Faults: check box
- File I/O: check box
- Registry: check box
- VirtAlloc: check box

CPU Sample Interval: milliseconds
- .NET Symbol Collection: check box
- Symbols: no V3.x NGEN Symbols
- Symbol Timeout: 120 seconds

Max Collect: seconds
- Stop Trigger: check box

Profiling wall-clock time
CPU sampling profiling
File/registry accesses
Allocation profiling
# Browsing Installed Providers

![](image)

<table>
<thead>
<tr>
<th>Processes</th>
<th>Providers</th>
<th>Keywords</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Microsoft-Windows-DotNETRuntime</td>
<td>GCKeyword</td>
<td>Always</td>
</tr>
<tr>
<td>wininit</td>
<td>ACPI Driver Trace Provider</td>
<td>GCHandleKeyword</td>
<td>Critical</td>
</tr>
<tr>
<td>lsass</td>
<td>Active Directory Domain Services: SAM</td>
<td>FusionKeyword</td>
<td>Error</td>
</tr>
<tr>
<td>winlogon</td>
<td>Active Directory: Kerberos Client</td>
<td>LoaderKeyword</td>
<td>Warning</td>
</tr>
<tr>
<td>svchost</td>
<td>Active Directory: NetLogon</td>
<td>JitKeyword</td>
<td>Informational</td>
</tr>
<tr>
<td>dwm</td>
<td>ADODB:1</td>
<td>NGenKeyword</td>
<td>Verbose</td>
</tr>
<tr>
<td>atiesnox</td>
<td>ADOMD:1</td>
<td>StartEnumerationKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>Application Popup</td>
<td>EndEnumerationKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>Application-Addon-Event-Provider</td>
<td>SecurityKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>ASP.NET Events</td>
<td>AppDomainResourceManagementKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>ATA Port Driver Tracing Provider</td>
<td>JitKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>AuthW NetShell Plugin</td>
<td>InteropKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>BCP:1</td>
<td>ContentionKeyword</td>
<td></td>
</tr>
<tr>
<td>svchost</td>
<td>BFE Trace Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Microsoft-Windows-DotNETRuntime:GCKeyword:Verbose</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Provider Browser**
PerfView Reports

PerfView has built-in support for CPU utilization, GC and JIT information, disk and file I/O, and a bunch of additional reports.
CPU Stacks

<table>
<thead>
<tr>
<th>Name</th>
<th>Inc %</th>
<th>Inc</th>
<th>Exc %</th>
<th>Exc</th>
<th>Fold</th>
<th>When</th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT</td>
<td>100.0</td>
<td>3,594.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,362,255</td>
<td>16,406,892</td>
</tr>
<tr>
<td>Process64 conhost (8576)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreadStart</td>
<td>96.2</td>
<td>3,464.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,362,255</td>
<td>16,406,892</td>
</tr>
<tr>
<td>SetThreadInitThunk</td>
<td>96.2</td>
<td>3,464.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,362,255</td>
<td>16,406,892</td>
</tr>
<tr>
<td>NtUserThreadCreate</td>
<td>96.2</td>
<td>3,464.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,362,255</td>
<td>16,406,892</td>
</tr>
<tr>
<td>NtUserWriteConsole</td>
<td>79.2</td>
<td>2,848.0</td>
<td>1.4</td>
<td>50</td>
<td>27</td>
<td>0</td>
<td>1,362,255</td>
<td>16,406,892</td>
</tr>
<tr>
<td>conhostSB_DoSvWriteConsole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,634,611</td>
<td>15,364,782</td>
</tr>
<tr>
<td>conhostSB_WriteRegionToScreen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,634,611</td>
<td>15,364,782</td>
</tr>
<tr>
<td>gdi32!PolyTextOut</td>
<td>43.4</td>
<td>1,560.0</td>
<td>0.6</td>
<td>29</td>
<td>21</td>
<td>0</td>
<td>4,634,611</td>
<td>15,360,782</td>
</tr>
<tr>
<td>gdi32!GdiFlush</td>
<td>1.0</td>
<td>35.0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4,769,634</td>
<td>14,704,707</td>
</tr>
<tr>
<td>conhostStreamScrollRegion</td>
<td>18.3</td>
<td>658.0</td>
<td>0.8</td>
<td>27</td>
<td>11</td>
<td>0</td>
<td>4,859,640</td>
<td>15,363,783</td>
</tr>
<tr>
<td>conhostSB_WriteChars</td>
<td>14.5</td>
<td>522.0</td>
<td>0.3</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>4,638,614</td>
<td>15,364,782</td>
</tr>
<tr>
<td>ntoskrnl?</td>
<td>0.1</td>
<td>2.0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8,373,016</td>
<td>10,350,237</td>
</tr>
</tbody>
</table>

In-trace activity highlighter

Call stack tree

Grouping options

Filtering options
### Disk I/O Stacks

#### Methods that call Disk Read DiskNum(0) net_kernel.elt (C:\Temp)\n
<table>
<thead>
<tr>
<th>Name</th>
<th>Inc %</th>
<th>Inc</th>
<th>Inc Ct</th>
<th>Exc %</th>
<th>Exc</th>
<th>Ex</th>
<th>Ex</th>
<th>Fs</th>
<th>Fc</th>
<th>W</th>
<th>Fi</th>
<th>Le</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Read DiskNum(0) net_kernel.elt (C:\Temp)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service Time Disk 0</td>
<td>13.8</td>
<td>31.9</td>
<td>20</td>
<td>13.8</td>
<td>31.9</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
</tr>
<tr>
<td>I/O Size 0x8000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module partmgr.sys &lt;&lt; partmgr.sys!PmGlobalDispatch &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module fhevol.sys &lt;&lt; fhevol.sys!FveFilterRundownRewrite &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module ntosknl &lt;&lt; ntosknl!KeExpandKernelStackAndCalloutInternal &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module fitmgr.sys &lt;&lt; fitmgr.sys!FtlDispatch &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module ntosknl &lt;&lt; ntosknl!KiPageFault &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module msvcr &lt;&lt; msvcr!memcopy &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module MpRtp &lt;&lt; MpRtp!RealtimeProtection!EngineVfzReadFileCallback &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module mengine &lt;&lt; mengine!&lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module MpSvc &lt;&lt; MpSvc!signal_wrapper &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module MpRtp &lt;&lt; MpRtp!RealtimeProtection!CFilterCommunicatorBase!CommunicatorThread &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module msvcr &lt;&lt; msvcr!endthread &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module kernel32 &lt;&lt; kernel32!BaseThreadThunk &lt;&lt; &gt;&gt;</td>
<td>9.1</td>
<td>21.1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,255,567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O Size 0x7000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O Size 0x1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell Contents: Thread (3256) CPU=174ms
### Methods that call Type System.String

<table>
<thead>
<tr>
<th>Name</th>
<th>Inc %</th>
<th>Inc</th>
<th>Inc Ct</th>
<th>Exc %</th>
<th>Exc</th>
<th>Exc Ct</th>
<th>Fold</th>
<th>V</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type System.String</td>
<td>100.0</td>
<td>279,526,4000.0</td>
<td>697</td>
<td>100</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.String.Concat(System.Object, System.Object) &gt;&gt;</td>
<td>100.0</td>
<td>279,526,4000.0</td>
<td>697</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.String.Concat(System.String, System.Object) &gt;&gt;</td>
<td>100.0</td>
<td>279,526,4000.0</td>
<td>697</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JackCompiler!JackCompiler.CompilerDriver.CompilerDriverWithCodeGenerator() &lt;&lt; class System.String, class System.String &gt;&gt;</td>
<td>21.9</td>
<td>61,343,4600.0</td>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JackCompiler!JackCompiler.CompilerDriver.CompilerDriverWithCodeGenerator() &lt;&lt; class System.String, class System.String &gt;&gt;</td>
<td>20.5</td>
<td>57,430,0000.0</td>
<td>143</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JackCompiler!JackCompiler.CompilerDriver.CompilerDriverWithCodeGenerator() &lt;&lt; System.Int32 &gt;&gt;</td>
<td>10.6</td>
<td>29,086,9100.0</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OTHER &lt;&lt; mscorlib!System.IO.TextWriter.WriteLine(System.String) &gt;&gt;</td>
<td>46.9</td>
<td>121,057,1000.0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Lab 3: Diagnosing Excessive Allocs
Memory Analysis with PerfView
.NET Memory Leak Analysis Process

1. **Ascertain** that a memory leak is present
2. **Capture** memory snapshots of target process
   - usually done by capturing dump files or using a profiler that can capture snapshots
3. **Compare** snapshots to understand which objects are being added and not removed
4. **Determine** why these objects aren’t being garbage collected
   - i.e., which reference paths are keeping the objects alive
Memory Leak Analysis with PerfView

- PerfView can generate heap snapshots (smaller than a dump), analyze, and compare them.
- Can also import dumps directly.
Comparing Snapshots

- Baseline with referencing paths

- Diff between snapshots
Programmatic ETW Analysis
Automatic ETW Analysis

- The **TraceEvent** library provides ETW analysis API
  - Understands kernel and CLR events
  - Supports call stacks (incl. managed)
  - Can start ETW sessions and/or process log files
Example Analysis Scenarios

- Monitor the system for CLR exceptions with stacks
  - ExceptionTraceData

- Get a profiling trace and look for regressions
  - TraceLog
  - SampledProfileTraceData
  - TraceCallStack

![SampledProfileTraceData Table]

<table>
<thead>
<tr>
<th>Name</th>
<th>Inc %</th>
<th>Inc %</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Process32 VSDebugging (10068)</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Thread (10340) CPU=32ms (Startup Thread)</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>OTHER  &lt;&lt; nt dll!_RtlUserThreadStart &gt;&gt;</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>VSDebugging!VSDebugging.Program.Main(class System.String[])</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>OTHER  &lt;&lt; cli!_Throw &gt;&gt;</td>
<td>100.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Throw(System.ApplicationException) foo</td>
<td>50.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Throw(System.ApplicationException) something bad happened</td>
<td>50.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Trace Analysis Example

```csharp
var traceLog = TraceLog.OpenOrConvert("trace.etl");
var process = traceLog.Processes.LastProcessWithName(...);
var symbolReader = new SymbolReader(Console.Out, symPath);

foreach (var exc in process.EventsInProcess.ByEventType<ExceptionTraceData>())
{
    Console.WriteLine(exc.ExceptionType);
    Console.WriteLine(exc.ExceptionMessage);
    var stack = exc.CallStack();
    while (stack != null)
    {
        Console.WriteLine(stack.CodeAddress.Method.FullName);
        stack = stack.Caller;
    }
}
```
Trace Session Example

```csharp
var session = new TraceEventSession("ObserveGCs");
session.EnableProvider(ClrTraceEventParser.ProviderGuid,
    TraceEventLevel.Verbose,
    (ulong)ClrTraceEventParser.Keywords.GC);

// Allocation tick every 100KB
var alloc = 
    session.Source.Clr.Observe<GCAllocationTickTraceData>();
alloc.Subscribe(ad => Console.WriteLine(ad.AllocationAmount));

var gc = session.Source.Clr.Observe<GCHeapStatsTraceData>();
gc.Subscribe(cd => Console.WriteLine(cd.GenerationSize2));

session.Source.Process();
```
Summary

- Semantic logging is valuable for automatically and easily analyzing large volumes of log data.
- ETW is the present and future of performance monitoring and diagnostics in development and production environments.
- PerfView is the tool of choice for .NET developers and offers advanced profiling and leak analysis capabilities.
Sasha Goldshtein
CTO, Sela Group

blog.sashag.net
@goldshtn

Thank You!