PERFVIEW

.NET runtime performance and ETW event analysis tool
OVERVIEW

• Formerly from Vance Morrison (.NET performance architect)
• Open-source Performance-analysis tool
• Can be used to investigate CPU and Memory-related issues

• Lightweight, no installation required
• Scriptable, configurable
  • Automated reports, integration to build process
• Customizable via custom extensions
  • Collecting data from custom performance counters
EVENT TRACING FOR WINDOWS

• Mechanism for tracing and logging events
• Built into the kernel of the OS (since Windows 2000)
• Allows for tracing and profiling in production environment
• Fast and efficient

• CLR publishes many events
  • Garbage collection
  • Lock contention
  • Thread Start/Stop/Wait
  • Exceptions
  • JIT...
WHAT CAN PERFVIEW DO FOR YOU?

• CPU investigation
  • Which methods are using the CPU time
• Managed memory investigation
  • Allocated objects
  • Memory leaks
  • Garbage collections
  • Boxing
• Multi-threading
  • Lock contention
  • Blocked threads
WHERE TO GET PERFVIEW

https://github.com/Microsoft/perfview
LIVE DEMO
EXAMPLE: GC COLLECT EVENTS

• How often did garbage collection run?
• How long does a collection take?
• Which generations were collected?
• Why did collection run?

• Use following options during ETW data collection
  • GC Only – turns of all other providers, approx. 200MB/h of data
  • GC Collect Only – only garbage collections, approx. 200MB/day of data
EXAMPLE: HEAP ALLOCATIONS

• Which objects are being allocated?
  • What is their size?
• Who allocates these objects?
• How often does boxing occur?
  • Which objects are being boxed?

• Normal ETW data collection collects coarse data only (1 sample/100KB)
• Detailed data can be collected using options
  • .NET Alloc – event fired every allocation, high performance impact (2x)
  • .NET SampAlloc – smart sampling, suitable for production use (10-20%)
EXAMPLE: MEMORY LEAKS

• Memory leaks can be present in managed code too!
• Possible sources
  • Delegates, events
  • Static fields of classes
  • Unused long-lived objects

• Which objects are currently allocated on the heap?
• Who owns a reference to them?
• Dump managed heap and examine collected data
EXAMPLE: LOCK CONTENTION

• Which locks are the most contended?

• Use Any Stacks view and look for Events:
  • Microsoft-Windows-DotNETRuntime/Contention/Start
  • Microsoft-Windows-DotNETRuntime/Contention/Stop
EXAMPLE: THREAD BLOCKING

• Threads may be blocked due to
  • Waiting for I/O
  • Waiting for Lock

• Use the Thread Times option when collecting data
• Inspect BLOCKED_TIME and CPU_TIME node
EXAMPLE: EXCEPTIONS

• Which exception were thrown?

• Events
  • Microsoft-Windows-DotNETRuntime/Exception/Start
  • Microsoft-Windows-DotNETRuntime/Exception/Stop
  • Microsoft-Windows-DotNETRuntime/ExceptionCatch/Start
  • Microsoft-Windows-DotNETRuntime/ExceptionCatch/Stop

• Use specialized exception stack viewer
CUSTOM EVENTS

• Simply derive from `System.Diagnostics.Tracing.EventSource`

```csharp
internal sealed class Events : EventSource
{
    ...
    Internal const int ProcessingStartId = 1;

    [Event(ProcessingStartId, Level = EventLevel.Informational, Keywords = Keywords.General)]
    public void ProcessingStart()
    {
        if (isEnabled())
        {
            this.WriteEvent(ProcessingStartId);
        }
    }
    ...
```

• Then add ""*MyEvents@StacksEnabled=true"" to additional providers
CUSTOM REPORTS

• PerfView can generate solution by using command
  • PerfView.exe CreateExtensionProject MyProjectName

• After creating custom command, invoke it using File->User command
  • Or press Alt-U
AUTOMATING THE PROCESS

• PerfView can be called from command line
  • Integration to the build process
  • Automated data collection

./PerfView.exe /AcceptEULA /LogFile=customreport.txt userCommand PrimeReports.PostBuildPerfAnalysis "./Programs/EtlDemo/bin/Debug/EtlDemo.exe"
PRODUCTION MONITORING

• We can run PerfView in production environment to diagnose problems that occur only during production workload

• PerfView collect "/StopOnPerfCounter=Processor:% Processor Time:_Total>90"

• Circular buffer (default 500MB ~ few minutes of data)
• Collection stops when condition is reached (additional 10s of data are collected)
PRODUCTION MONITORING

• Other examples

• PerfView "/MonitorPerfCounter=Memory:Available MBytes:@10" collect
  • Logs available memory every 10 seconds,
  • PerfView/PerformanceCounterUpdate event

• PerfView collect "/StopOnRequestOverMSec:2000" /CollectMultiple:3
  • Stops on ASP.NET request that takes longer than 2000ms, repeats 3 times

• PerfView collect "/StopOnException:ApplicationException"
  /Process:MyService /ThreadTime
  • Stops when exception of given type is thrown in the given process
PERFVIEW AND LINUX

- PerfView itself is just a stack viewer, it analyses set of samples containing
  - Timestamp – optional, defaults to 0
  - Metric – optional, defaults to 1
  - Stack – logically just a set of strings

- Since V1.9, PerfView can read textual representation of Linux perf tool output
  - Recognized extension .data.txt and .trace.zip

- Simply collect data using the PerfCollect script and then transfer to windows machine for analysis