Системные метрики: собираем подводные камни

DOTNEXT

Евгений Пешков

e-mail: peshkov@kontur.ru
telegram/twitter: @epeshk

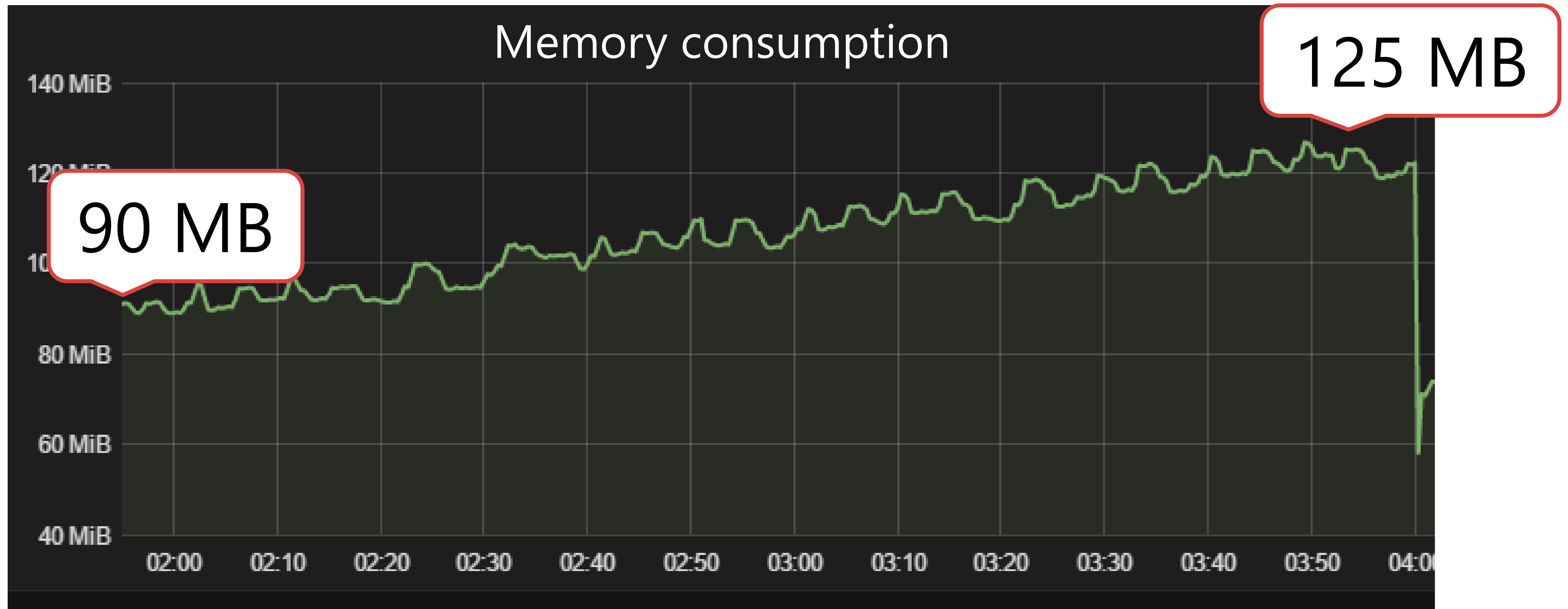
Проблемы

- Средства сбора метрик – обычно GUI приложения
- Сбор метрик может быть ресурсоёмким и приводить к замедлению приложения

План

- Метрики по памяти: чем отличаются и как их собирать
- Класс Process в .NET: что с ним не так
- Performance Counters: что это такое и как их готовить
  - .NET обёртка и её недостатки
  - Особенности нативного API
- ETW для realtime-мониторинга .NET приложений

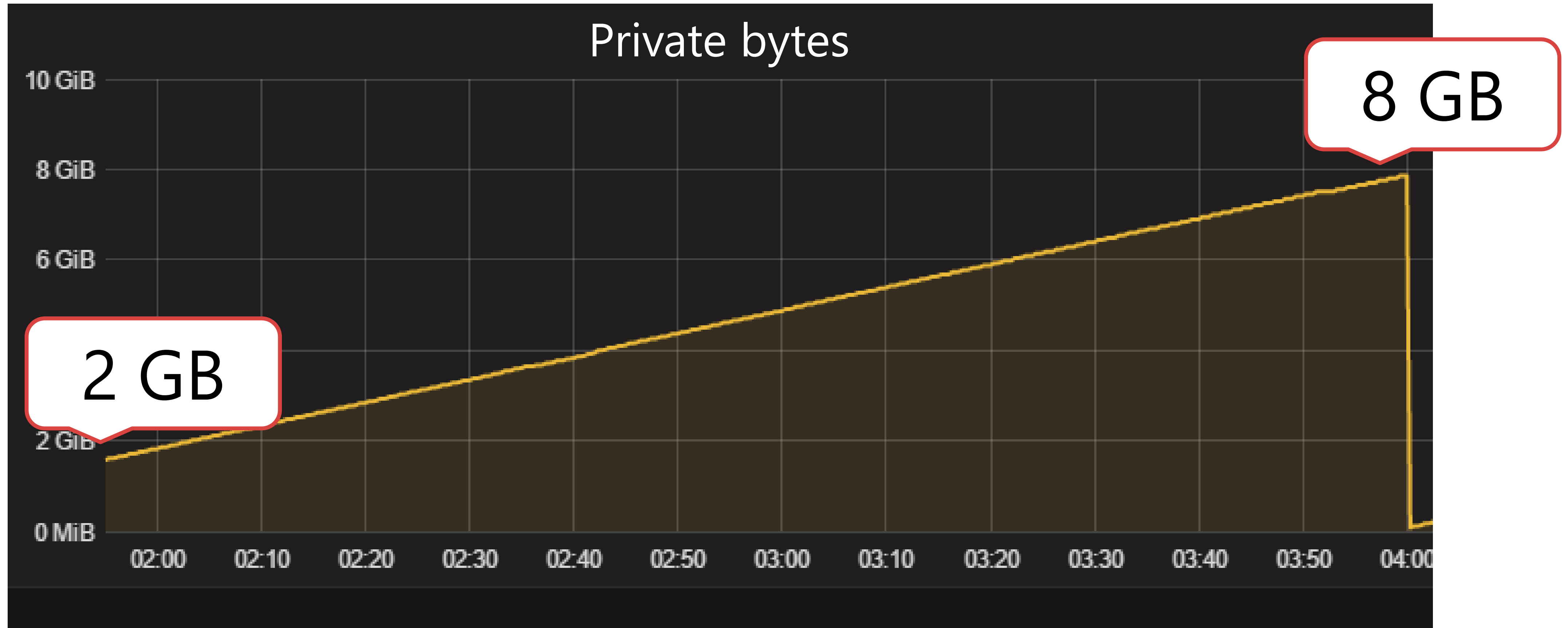
История одного OutOfMemory



История одного OutOfMemory

- Working set
- Private Working Set
- Shared Working Set
- Commit size
- Paged pool
- Nonpaged pool

История одного OutOfMemory



Метрики по памяти процесса

Private Bytes (Commit Size) – сколько байт выделено процессом

```
var arr = new byte[10*MB];
```

Working Set – сколько байт процесса находится в RAM

```
for (var i = 0; i < arr.Length; i+=4096)
    arr[i] = 1;
```

Метрики «Память» – не существует

System.Diagnostics.Process

```
var process = Process.GetCurrentProcess();

var workingSet = process.WorkingSet64;
var privateBytes = process.PrivateMemorySize64;
```

System.Diagnostics.Process

```
var process = Process.GetCurrentProcess();
var before = process.PrivateMemorySize64;

var memory = new byte[10 * MB];



var after = process.PrivateMemorySize64;
double diff = (after - before) / (double) MB;
Console.WriteLine(diff);

//  0
```

System.Diagnostics.Process

```
var process = Process.GetCurrentProcess();
var before = process.PrivateMemorySize64;

var memory = new byte[10 * MB];

process.Refresh();

var after = process.PrivateMemorySize64;
double diff = (after - before) / (double) MB;
Console.WriteLine(diff);

//  11.5
```

Process internals

Platform	Mean	Allocated
FW 4.7.1	12.027 ms	1417 KB
FW 4.7.2	9.576 ms	127 KB
Core 2.1	9.750 ms	3 KB

Windows 10 1803
i7-6700
Time per single call

- Executes in a milliseconds
- GC overhead
- May cause notable performance impact

NtQuerySystemInformation

```
NtQuerySystemInformation(SystemProcessInformation, ...)
```

- Возвращает информацию по **всем** процессам и потокам
  - `SYSTEM_PROCESS_INFORMATION`
  - `SYSTEM_THREAD_INFORMATION`

- Process в .NET 4.7.1 создаёт объекты для всех структур
  - `class SystemProcessInformation`
  - `class SystemThreadInformation`

Process internals

- .NET 4.7.2
  - Create `SystemThreadInformation` instances only for target process threads
- .NET Core
  - class -> struct
- `NtQuerySystemInformation` call is expensive

https://github.com/dotnet/corefx/pull/24198

NtQuerySystemInformation slowdown

`O(Process + Threads) time complexity`

- Threads leak
- Zombie processes

Можно ли лучше?

```
[DllImport("psapi.dll")]
static extern bool GetProcessMemoryInfo(
    IntPtr process,
    out PROCESS_MEMORY_COUNTERS_EX ppsmemCounters,
    int cb);

struct PROCESS_MEMORY_COUNTERS_EX {
    ...
    IntPtr WorkingSetSize;
    ...
    IntPtr PrivateUsage;
}
```

Process internals

Platform	Mean	Allocated
FW 4.7.1	12.027 ms	1417 KB
FW 4.7.2	9.576 ms	127 KB
Core 2.1	9.750 ms	3 KB
WinApi	**526.400 ns**	**0 B**

Process internals

Platform	Mean	Allocated
FW 4.7.1	12027000.0 ns	1417 KB
FW 4.7.2	9576000.0 ns	127 KB
Core 2.1	9750000.0 ns	3 KB
WinApi	**526.4 ns**	**0 B**

Process internals: выводы

- Класс Process неэффективен для частого сбора метрик по памяти
- Использовать нативные API – несложно

Метрики потребления CPU

Задача: посчитать % загрузки процессора процессом

```
Process(MyProgram)\% Processor Time
```

Performance counters

Performance counter's

Плюсы:

- Универсальное API для сбора метрик в Windows
- Метрики Windows/.NET Framework/custom

Минусы:

- Разработан под графический интерфейс
- Сложное API
- Нестабильны

Performance counter's

```
Category(Instance)\Counter
```

Categories:

- Process
- .NET CLR Memory
- Physical Disk
- Memory

Performance counter's

`Category(Instance)\Counter`

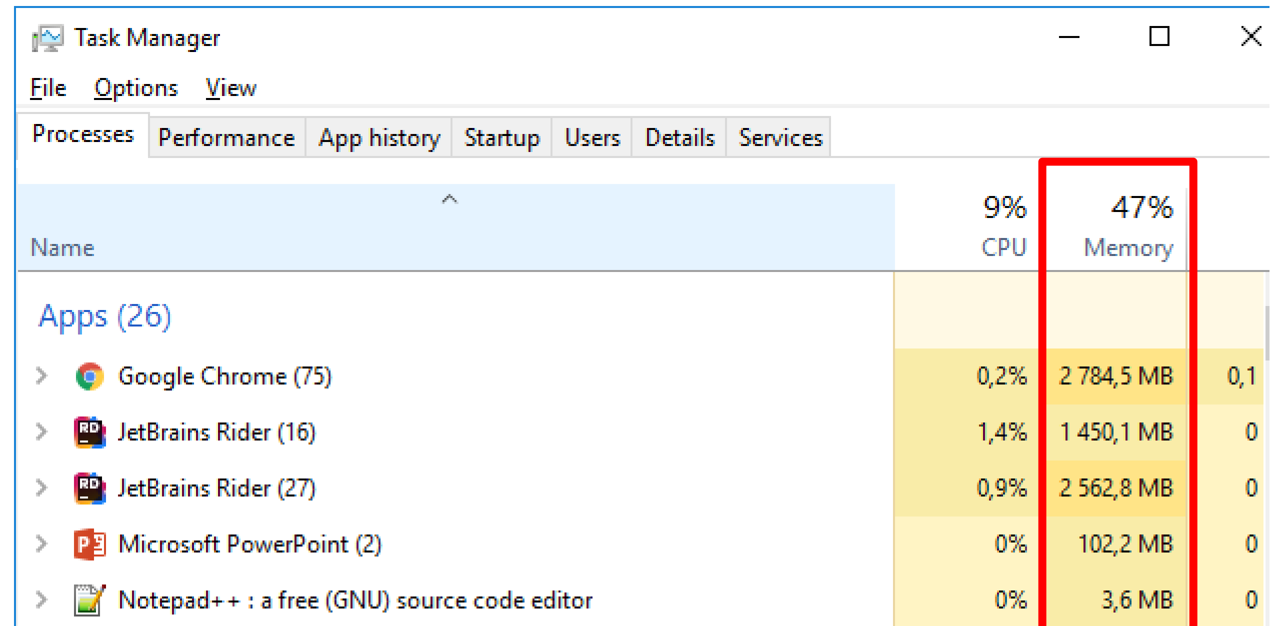
Instances:

- Process
  - dotnet
  - devenv
  - svchost
- Physical Disk:
  - 0 C:
  - 1 D: E:
- Memory:
  - no instances

Performance counter's

```
Category(Instance)\Counter
```

Counters:
- Process
  - % Processor Time
  - Working Set
  - Working Set - Private
- Memory:
  - Committed bytes



Categories

Counters

Instances

Description

Add Counters

Available counters

Select counters from computer:

<Local computer>

Browse...

PowerShell Workflow

Print Queue

Process

% Privileged Time

% Processor Time

% User Time

Creating Process ID

Elapsed Time

Instances of selected object:

fsnotifier64

Idle

igfxCUIService

igfxEM

IntelCpHDCPSvc

IntelCpHeciSvc

Search

Add >>

Added counters

Counter	Parent	Inst...	Computer

Remove <<

Show description

OK

Cancel

Description:

The total elapsed time, in seconds, that this process has been running.

Performance counters

```
var pc = new PerformanceCounter(
    "Process",
    "% Processor Time",
    "MyProgram");

double value = pc.NextValue();
```

Что может пойти не так?

Performance counters

```
var pc = new PerformanceCounter(          << Exception
    "Process",
    "% Processor Time",
    "MyProgram");

double value = pc.NextValue();            << Exception
```

Возвращаемое значение

Process\% Processor Time

от 0% до 100% ?

Возвращает значение от 0 до 100 * LogicalCores

Instance name

`Process(MyProgram)\% Processor Time`

MyProgram.exe PID: 4004	MyProgram.exe PID: 3760	MyProgram.exe PID: 12600
MyProgram	MyProgram#1	MyProgram#2

Performance counters

```
Process(MyProgram)\ID Process
```

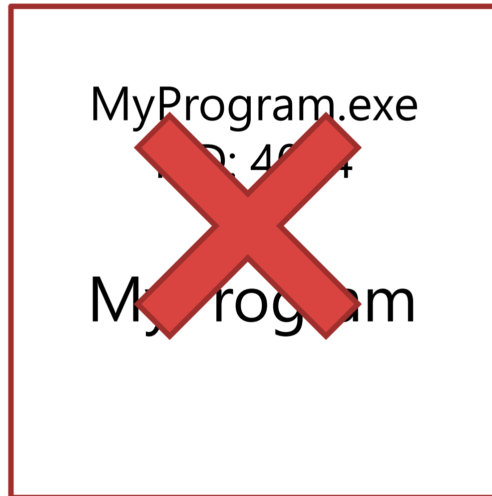
```
var processCategory = new PerformanceCounterCategory("Process");
var instanceNames = processCategory.GetInstanceNames();

foreach (var name in instanceNames)
    using (var pc = new PerformanceCounter("Process", "ID Process", name))
        if ((int) pc.NextValue() == pid)
            return name;
```

https://stackoverflow.com/questions/29922791/performance-counter-instance-name-vs-process-name

Instance name

`Process(MyProgram)\% Processor Time`



MyProgram.exe
PID: 3760

MyProgram#1

MyProgram.exe
PID: 12600

MyProgram#2

Instance name

`Process(MyProgram)\% Processor Time`

MyProgram.exe [illegible] [illegible]	MyProgram.exe PID: 3760 ~~MyProgram#1~~ **MyProgram**	MyProgram.exe PID: 12600 ~~MyProgram#2~~ **MyProgram#1**

Performance counters

Пересоздаём `PerformanceCounter` на каждый замер:

```
var pc = new PerformanceCounter(
    "Process",
    "% Processor Time",
    GetActualInstanceName(pid));

double value = pc.NextValue();
```

Raw values

`Process\% Processor Time`

Возвращает 0 при первом вызове

Raw values

- Raw value
- Performance counter type:
  - CounterDelta64
  - Timer100Ns
  - …

$$Value[i] = \frac{RawValue[i] - RawValue[i-1]}{\Delta t}$$

Instance name

```
var pc = new DynamicPerformanceCounter(
    "Process",
    "% Processor Time",
    () => GetActualInstanceName(pid));

double value = pc.NextValue();
```

Instance name – other way

```
[HKLM\System\CurrentControlSet\Services\Perfproc\Performance]
"ProcessNameFormat" = 2
"ThreadNameFormat" = 2
```

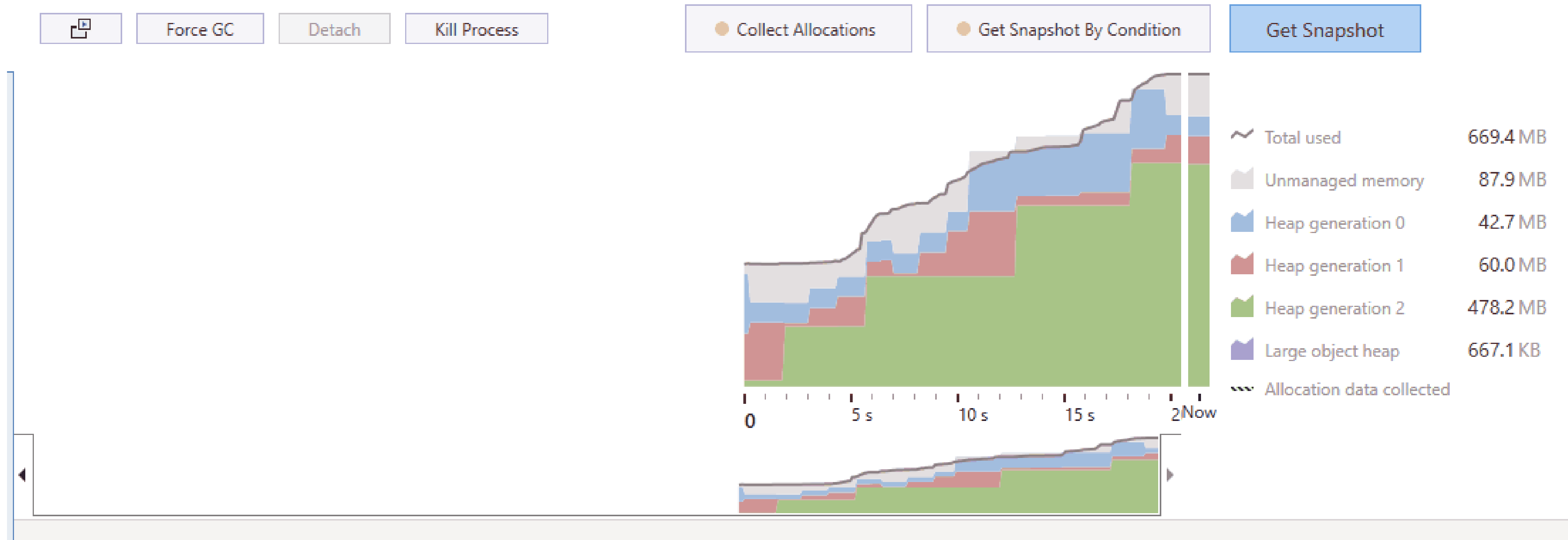
App#1 -> App_{ProcessID}

https://docs.microsoft.com/en-us/windows/desktop/perfctrs/handling-duplicate-instance-names

Performance counters: выводы

- Будьте готовы к исключениям
- Проверяйте, что возвращает counter
- Instance name процесса может изменяться
- Performance counters могут работать нестабильно

dotMemory: heap stats



.NET Performance counters

- .NET CLR Memory
  - Gen 0 Heap Size
  - Gen 1 Heap Size
  - Gen 2 Heap Size
  - Large Object Heap Size
- .NET CLR Jit
  - # IL Bytes Jitted
- .NET CLR Exceptions
  - # Exceps Thrown

.NET Performance counters

```
var pc = new DynamicPerformanceCounter(
    ".NET CLR Memory", "Gen 2 Heap Size",
    () => GetActualInstanceName(pid));
double value = pc.NextValue();
```

.NET Instance name

- Instance name существуют в рамках одной категории
- В разных категориях Instance name может различаться

- `Process(App)\ID Process`
- `.NET CLR Memory(App)\Process ID`

.NET Performance counters

Если для процесса нет .NET Instance name

- Процесс, собирающий метрики – 32-битный
- Значение .NET CLR Memory\Process ID ещё не собиралось

Что вычисляет значения для категории .NET CLR Memory?

Garbage collector

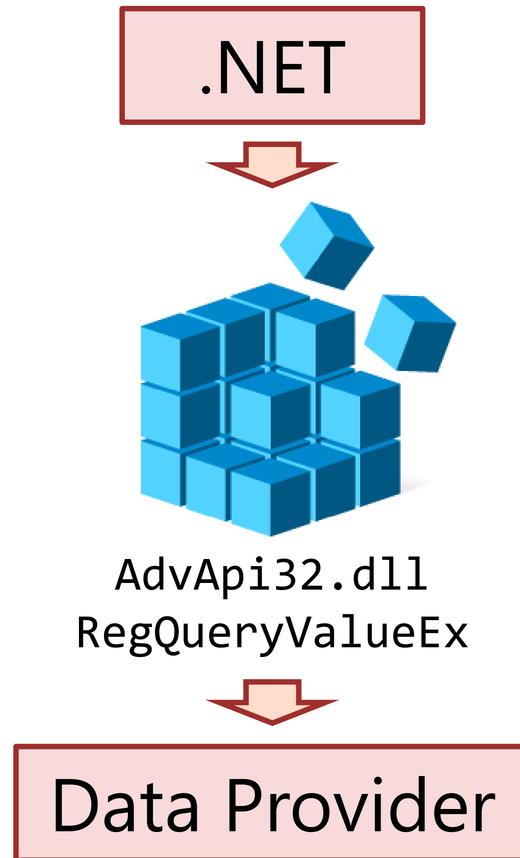
Performance of performance counters

Получение Instance name для процесса:

Method	Mean	Allocated
Process	25.782 ms	3770 KB
.NET CLR Memory	1.750 ms	874 KB

300 processes (+100 managed), time per single iteration

Performance counters under the hood



RegQueryValueEx

Process\ID Process via Registry API

```
RegQueryValueEx(
    HKEY_PERFORMANCE_DATA,
    "230 784",
    ...);
```

- 230 – "Process" Category ID
- 784 – "ID Process" Counter ID

Where is the Instance?

RegQueryValueEx cons

RegQueryValueEx:

- Большие бинарные структуры (~500 KB)
- Много лишних данных

.NET wrapper:

- Memory traffic
- Неэффективное использование API

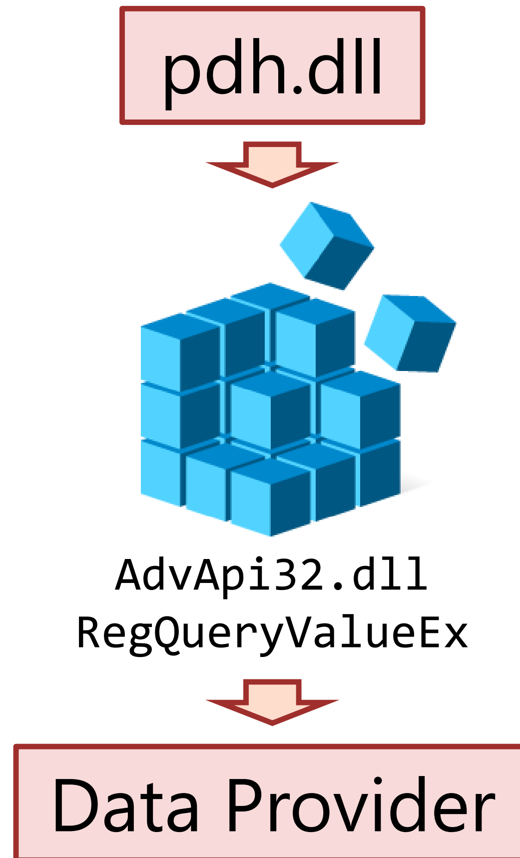
MSDN time

Using the Registry Functions to Consume Counter Data

"You should not use the registry functions to consume counter data. Instead, you should use the Performance Data Helper (PDH) functions to consume counter data." - MSDN

https://docs.microsoft.com/en-us/windows/desktop/perfctrs/using-the-registry-functions-to-consume-counter-data

PDH – Performance Data Helpers



PDH: пример

```
query = PdhOpenQuery()
counter1 = PdhAddCounter(query, "Process(App*)\ID Process")
counter2 = PdhAddCounter(query, "Process(App*)\Working Set")

PdhCollectQueryData(query)
PdhGetRawCounterValue(counter1, ...)

PdhCloseQuery(query)
```

PDH: performance

Obtain Process ID by instance name:

Method	Mean	Allocated
Process	25.782 ms	3792 KB
Process (PDH)	4.530 ms	0 B
.NET CLR Memory	1.750 ms	874 KB
.NET CLR Memory (PDH)	0.042 ms	0 B

Time per single iteration

Pdh

Всё ли теперь хорошо?

Локализация

- Процесс\% загруженности процессора
- Процесс\Рабочий набор
- Исключения CLR .NET\Глубина от выдачи до захвата

Локализация

```
PdhAddCounter -> PdhAddEnglishCounter
```

Pdh: performance

Obtain Process ID by instance name:

Method	Mean	Allocated
Process	25.782 ms	3792 KB
Process (PDH)	4.530 ms	0 B
.NET CLR Memory	1.750 ms	874 KB
.NET CLR Memory (PDH)	0.042 ms	0 B

Time per single iteration

Почему категория Process - медленная

- Performance Counters могут получать данные из разных источников (data providers)

- Производительность источников данных - разная

- Данные для категории Process берутся из **`NtQuerySystemInformation`**

Измерение загрузки CPU без Performance Counters

```
[DllImport("kernel32.dll")]
static extern bool GetSystemTimes(out ...);

[DllImport("kernel32.dll")]
static extern bool GetProcessTimes(IntPtr hProcess, out ...)
```

Измерение загрузки CPU без Performance Counters

Method	Mean
PdhCounter	3,057.1 us
WinApi	4.1 us

Time per single iteration

Performance of performance counters

На хосте закончилась память
Запущенные процессы её не потребляют

Утекла память ядра в Nonpaged pool из-за драйвера с багом

Performance of performance counters

Задача: собрать machine-wide метрики по памяти – память ядра, системные кэши

```
1) Performance counters in "Memory" category

2) WinApi:
[DllImport(psapi, SetLastError = true)]
private static extern bool GetPerformanceInfo(
    out PERFORMANCE_INFORMATION pPerformanceInformation,
    int cb
);
```

Performance of performance counters

Method	Mean
PdhCounter	43.319 us
ManagedCounter	331.758 us
GetPerformanceInfo	3,929.100 us

Time per single iteration

Performance of performance counters: выводы

- Не используйте категорию Process
- Сравнивайте разные способы сбора метрик
- Используйте PDH, если упёрлись в ограничения .NET-обёртки

Performance counters is not enough

- Задача: собрать метрики по GC для .NET Core приложения

- Нет поддержки со стороны .NET Core
  - .NET CLR категории – только для .NET Framework
- Возможности Performance counters ограничены возвратом числовых значений

Event Tracing for Windows

Event Tracing for Windows

- “The worst API ever made”
  - https://caseymuratori.com/blog_0025
- “How the worse API (LTTng) is refined than the worst API (ETW), how to write a good interface”
  - https://www.codeblogbt.com/archives/7221

Event Tracing for Windows

- Realtime (in memory) мониторинг/логгирование в файл
- Performance oriented
- Сырые бинарные данные

.NET Events

- GC Heap Stats (.NET CLR Memory analog)

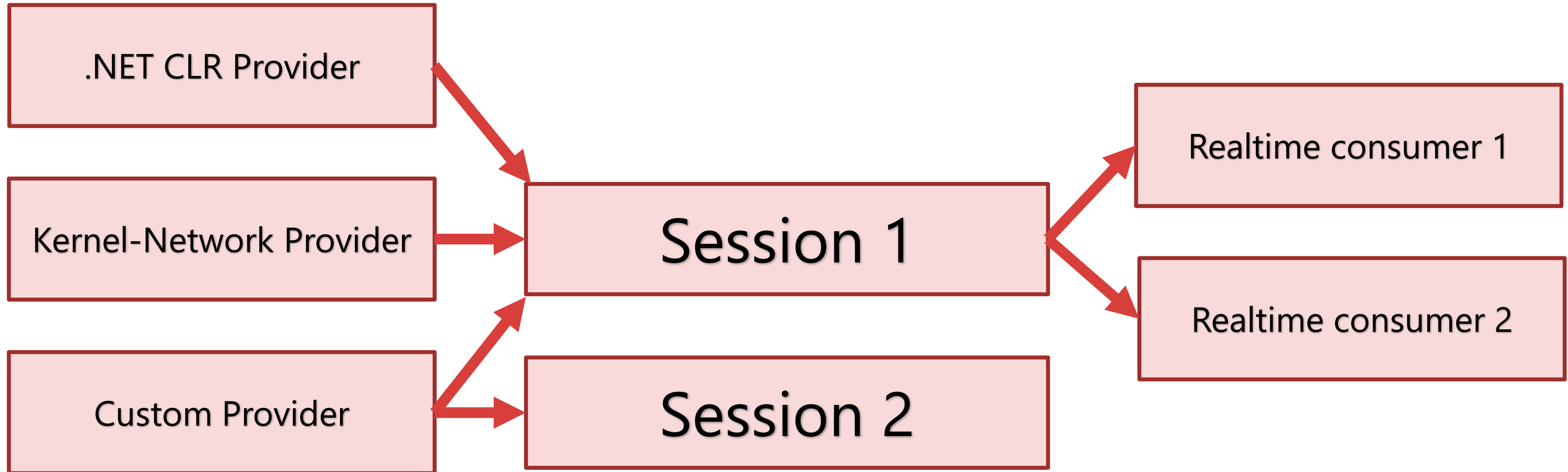
- GC Start
- GC End

- GC Start + GC End = GC Monitor

```
WARN [GC Monitor] Gen-2 GC occured which lasted
for 6.877 seconds.
```

https://docs.microsoft.com/en-us/dotnet/framework/performance/clr-etw-events

Event Tracing for Windows



ETW via .NET

Microsoft.Diagnostics.Tracing.TraceEvent

- Managed-wrapper over ETW
- “PerfView from code”
- Predefined ETW event parsers

https://www.nuget.org/packages/Microsoft.Diagnostics.Tracing.TraceEvent/
https://github.com/Microsoft/perfview/

TraceEvent library: example

```
TraceEventSession session = new TraceEventSession("...");

session.EnableProvider(ClrTraceEventParser.ProviderGuid);

ETWTraceEventSource source = session.Source;
source.Clr.GCHeapStats += data => ...;

source.Process();
```

TraceEvent library: example

```
source.Clr.GCHeapStats += (GCHeapStatsTraceData data) => ...;
```

- `class GCHeapStatsTraceData`
- Объекты `GCHeapStatsTraceData` переиспользуются

ETW Limits

- Machine wide session limit (64 on Win 10)
- Per-producer session limit (8)

ETW Session reuse

```
new TraceEventSession(
    ETWSessionName,
    TraceEventSessionOptions.NoRestartOnCreate)
    {
        StopOnDispose = false
    };
```

ETW: выводы

- Универсальный способ сбора managed memory метрик для .NET Framework/.NET Core на Windows
- Производительная .NET-обёртка
- Machine-wide лимиты количества ETW-объектов

Совсем кратко о Linux

- LTTng – Linux Tracing Toolkit next generation
- Аналог ETW
- Есть поддержка со стороны .NET runtime

- Нет .NET API для realtime-мониторинга

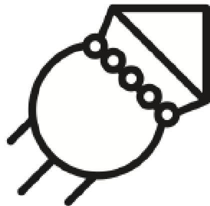
ВЫВОДЫ

- Сбор системных метрик не так прост, как кажется
- Встроенные в .NET средства сбора метрик могут создавать нежелательную нагрузку на приложение
- Performance counters – неудобное, но полезное API, если его правильно готовить
- С помощью ETW можно реализовать сложные сценарии мониторинга

LINKS

- Bruce Dawson blog
  - Making Windows Slower
  - 24-core CPU and I can’t (move my mouse/type an email)
  - Zombie Processes are Eating your Memory

  https://randomascii.wordpress.com
- Our PDH performance counters and other metrics libraries

  https://github.com/vostok/sys.metrics.perfcounters
  https://github.com/vostok/sys.metrics.windows
  https://github.com/vostok/sys.metrics.etw
- Samples and benchmarks

  https://github.com/epeshk/dotnext-2018-sysmetrics

ВОПРОСЫ

DOTNEXT

Евгений Пешков

e-mail: peshkov@kontur.ru
telegram/twitter: @epeshk