Источник *http://www.devowl.net/2017/05/minidump-minidumpwritedump-c.html*

17 мая 2017 г.

MiniDumpWriteDump. Создаём minidump на C#.

Что такое minidump?

Minidump - это, в зависимости от запроса, полный или частичный "слепок" оперативной памяти работающего процесса с сохранением полного или частичного состояния на момент получения снимка.
Как это можно поможет в работе? Дело в том, что **minidump** можно создавать самому, а начиная с **Framework 4.0 - Visual Studio** при открытии **minidump** файла может показать точку останова, стек вызова и даже локальные процессы. Если конечно их не убрал **/Release** оптимизатор. Иными словами мы увидим такую же картину, как если бы мы работающее приложение поставили на паузу и решили посмотреть стек.
Заинтересовало не так ли? Тогда приступим!

Способы создания.

Известные мне способы создания дампа:

1. Создание дампа внутри вашего приложения вызывая метод **MiniDumpWriteDump** из **dbghelp.dll**.
2. Используя какие то внешние (**out of process**) утилиты:
   Стандартный диспетчер задач, **Process Explorer** или **Process Hacker** и т.д

Создание Minidump внутри приложения

Для создания дампа внутри нашего приложения нам необходимо всего лишь вызвать функцию MiniDumpWriteDump из библиотеки **dbghelp.dll**, которая есть у всех, или почти всех, версий **Windows**. Из отличий версий могут быть только флаги параметра, которые добавлялись в ходе развития библиотеки.
Зачем создавать дамп из тела приложения? Создание таким образом дает **возможность контролировать место останова, для дальнейшего анализа дампа**. Дампы созданные из диспетчера задач не могу гарантировать, что при открытии в студию вы попадете в "полезное" место и в котором будет все, что требуется для решении вашей задачи или проблемы.

Я создал простенькое консольное приложение для демонстрации работы:

1. Код метода Main():

```
2. namespace Dumper
3. {
4.     using System;
5.     using System.IO;
6.     using System.Linq;
7.     using System.Runtime.CompilerServices;
8.     using System.Threading;
9.     using Dumps;
10.
11.    internal static class Program
12.    {
13.        private static bool _stopThread;
14.
15.        /// <summary>
```

```
        /// Entry point.
        /// </summary>
        static void Main(string[] args)
        {
            Directory.GetFiles(DumpUtils.DumpDirectory).ToList().ForEach(File.Delete);
            CreateThread();
            TestMethod();
            _stopThread = true;
        }

        private static void CreateThread()
        {
            new Thread(() =>
            {
                while (!_stopThread)
                {
                    Console.WriteLine("while(true)");
                }
            })
            {
                Name = "While(true) thread"
            }.Start();
        }

        [MethodImpl(MethodImplOptions.NoOptimization)]
        private static void TestMethod()
        {
            var dateTime = DateTime.Now;
            try
            {
                Console.WriteLine(dateTime);

                int someDouble = 0;
                someDouble = 10 / someDouble;
            }
            catch (Exception)
            {
                DumpUtils.WriteDump();
            }
        }
    }
}
```

58. Код класса создающий **Minidump** файл:

```
namespace Dumper.Dumps
{
    using System;
    using System.Diagnostics;
    using System.IO;
    using System.Reflection;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;

    /// <summary>
    /// Minidump support tools.
    /// </summary>
    public static class DumpUtils
    {
        /// <summary>
        /// Folder for saved minidumps.
        /// </summary>
        public const string DumpDirectory = "Minidump";

        [DllImport("dbghelp.dll")]
        private static extern bool MiniDumpWriteDump(IntPtr hProcess, int processId, Int
   Ptr hFile, int dumpType,
            IntPtr exceptionParam, IntPtr userStreamParam, IntPtr callStackParam);

```

```
82.         /// <summary>
83.         /// Write minidump to file.
84.         /// </summary>
85.         /// <param name="minidumpType">Minidump flag(s).</param>
86.         [MethodImpl(MethodImplOptions.NoOptimization)]
87.         public static bool WriteDump(
88.             MinidumpType minidumpType = MinidumpType.MiniDumpWithFullMemory |
89.             MinidumpType.MiniDumpWithHandleData |
90.             MinidumpType.MiniDumpWithUnloadedModules |
91.             MinidumpType.MiniDumpWithFullMemoryInfo |
92.             MinidumpType.MiniDumpWithThreadInfo)
93.         {
94.             try
95.             {
96.                 if (!Directory.Exists(DumpDirectory))
97.                 {
98.                     Directory.CreateDirectory(DumpDirectory);
99.                 }
100.
101.                   var currentProcess = Process.GetCurrentProcess();
102.                   var fileName = GetNewDumpFileName(currentProcess.ProcessName);
103.                   var currentDir = Path.GetDirectoryName(Assembly.GetExecutingAssembly().
    Location);
104.                  var filePath = Path.Combine(currentDir, DumpDirectory, fileName);
105.                  var handler = currentProcess.Handle;
106.                  var processId = currentProcess.Id;
107.
108.                  using (var fileStream = new FileStream(filePath, FileMode.CreateNew))
109.                  {
110.                      return MiniDumpWriteDump(
111.                          handler,
112.                          processId,
113.                          fileStream.SafeFileHandle.DangerousGetHandle(),
114.                          (int) minidumpType,
115.                          IntPtr.Zero,
116.                          IntPtr.Zero,
117.                          IntPtr.Zero);
118.                  }
119.              }
120.              catch (Exception)
121.              {
122.                  return false;
123.              }
124.          }
125.
126.          private static string GetNewDumpFileName(string processName)
127.          {
128.              return string.Format("{0}_{1}_{2}.dmp", processName,
129.                  DateTime.Now.ToString("yyyy-dd-mm_HH-mm-ss"),
130.                  Path.GetRandomFileName().Replace(".", ""));
131.          }
132.      }
    }
```

133. Для гибкости дал возможность пользователю самому задавать параметры дампа:

```
134. namespace Dumper.Dumps
135. {
136.     using System;
137.
138.     /// <summary>
139.     /// Identifies the type of information that will be written to the minidump file by
     the MiniDumpWriteDump function.
140.     /// </summary>
141.     /// <remarks>
142.     /// https://msdn.microsoft.com/en-us/library/windows/desktop/ms680519(v=vs.85).aspx
143.     /// </remarks>
144.     [Flags]
```

```
145.     public enum MinidumpType
146.     {
147.         MiniDumpNormal = 0x00000000,
148.         MiniDumpWithDataSegs = 0x00000001,
149.         MiniDumpWithFullMemory = 0x00000002,
150.         MiniDumpWithHandleData = 0x00000004,
151.         MiniDumpFilterMemory = 0x00000008,
152.         MiniDumpScanMemory = 0x00000010,
153.         MiniDumpWithUnloadedModules = 0x00000020,
154.         MiniDumpWithIndirectlyReferencedMemory = 0x00000040,
155.         MiniDumpFilterModulePaths = 0x00000080,
156.         MiniDumpWithProcessThreadData = 0x00000100,
157.         MiniDumpWithPrivateReadWriteMemory = 0x00000200,
158.         MiniDumpWithoutOptionalData = 0x00000400,
159.         MiniDumpWithFullMemoryInfo = 0x00000800,
160.         MiniDumpWithThreadInfo = 0x00001000,
161.         MiniDumpWithCodeSegs = 0x00002000,
162.         MiniDumpWithoutAuxiliaryState = 0x00004000,
163.         MiniDumpWithFullAuxiliaryState = 0x00008000,
164.         MiniDumpWithPrivateWriteCopyMemory = 0x00010000,
165.         MiniDumpIgnoreInaccessibleMemory = 0x00020000,
166.         MiniDumpWithTokenInformation = 0x00040000
167.     };
168. }
```

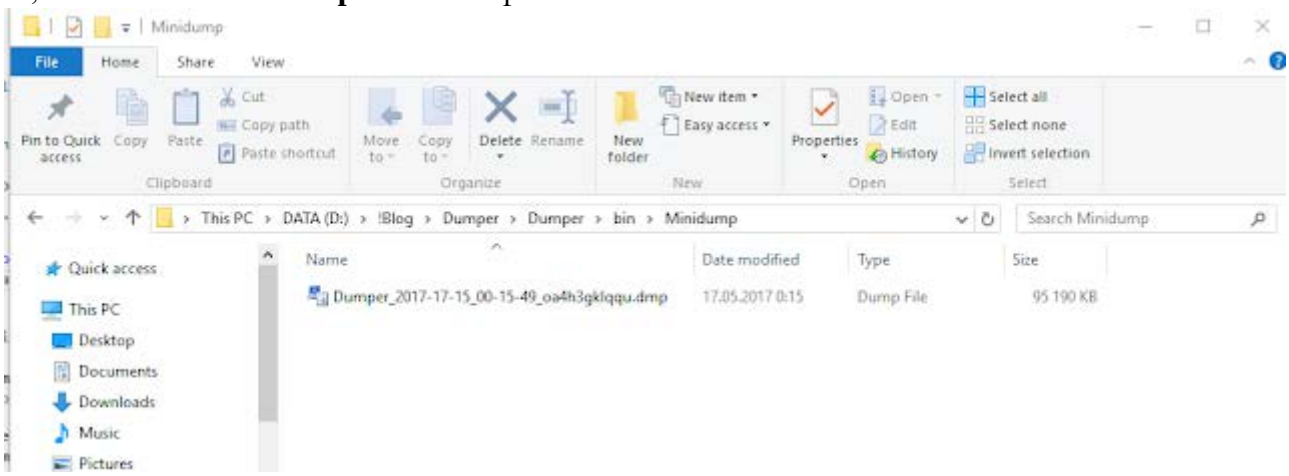
Теперь, что мы получим в итоге:

- В классе **Program** у нас создается поток для дальнейшей демонстрации.

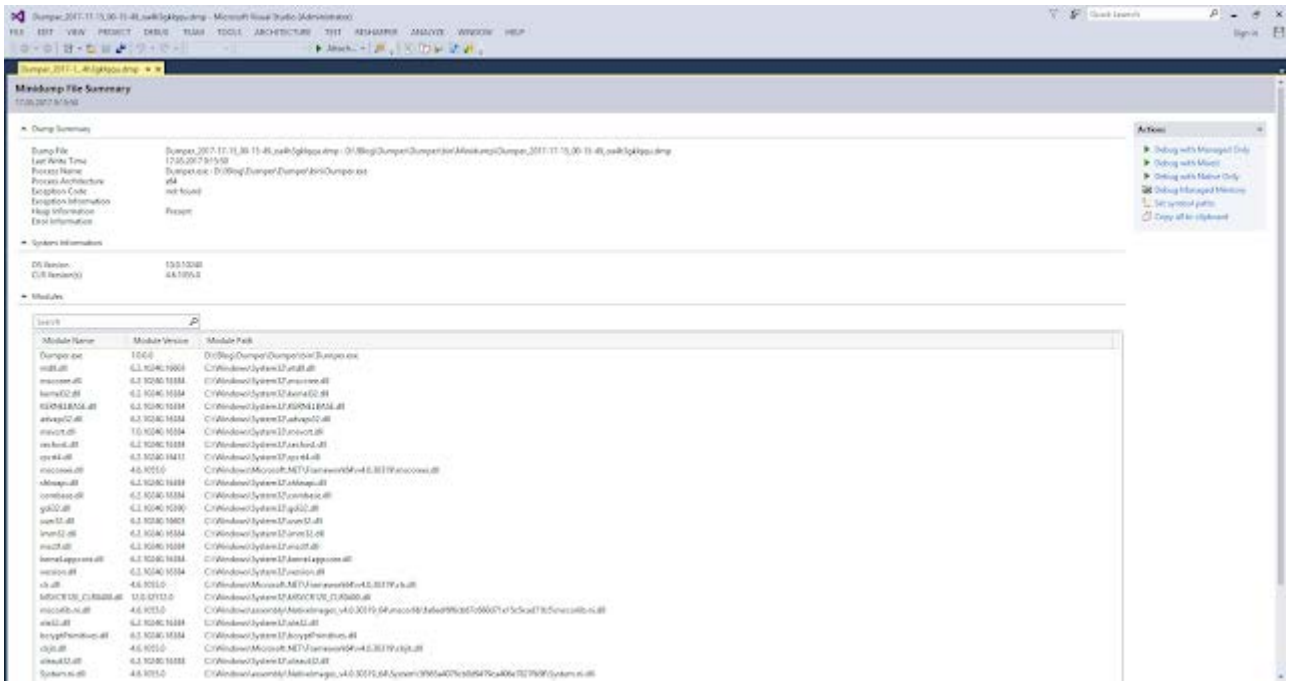
- В методе **TestMethod()** я планирую снимать полный дамп, когда происходит исключение.
- Сам дамп будет располагаться в папке **Minidump**, а для простоты поиска файлов перед запуска приложения я эту папку очищаю.

Отлично! Мы готовы к запуску демонстрации.

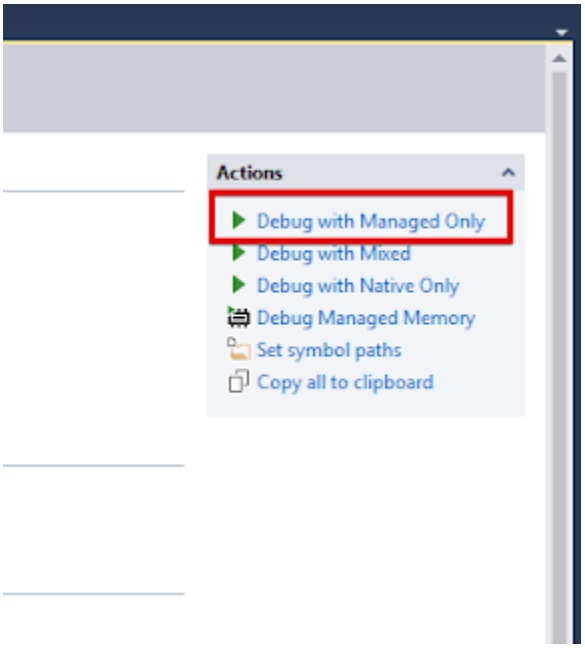
Для приближенности к реальной ситуации выставляем **"Release"**, компилирую и запускаю. Далее видим, что в папке **Minidump** появился файл:



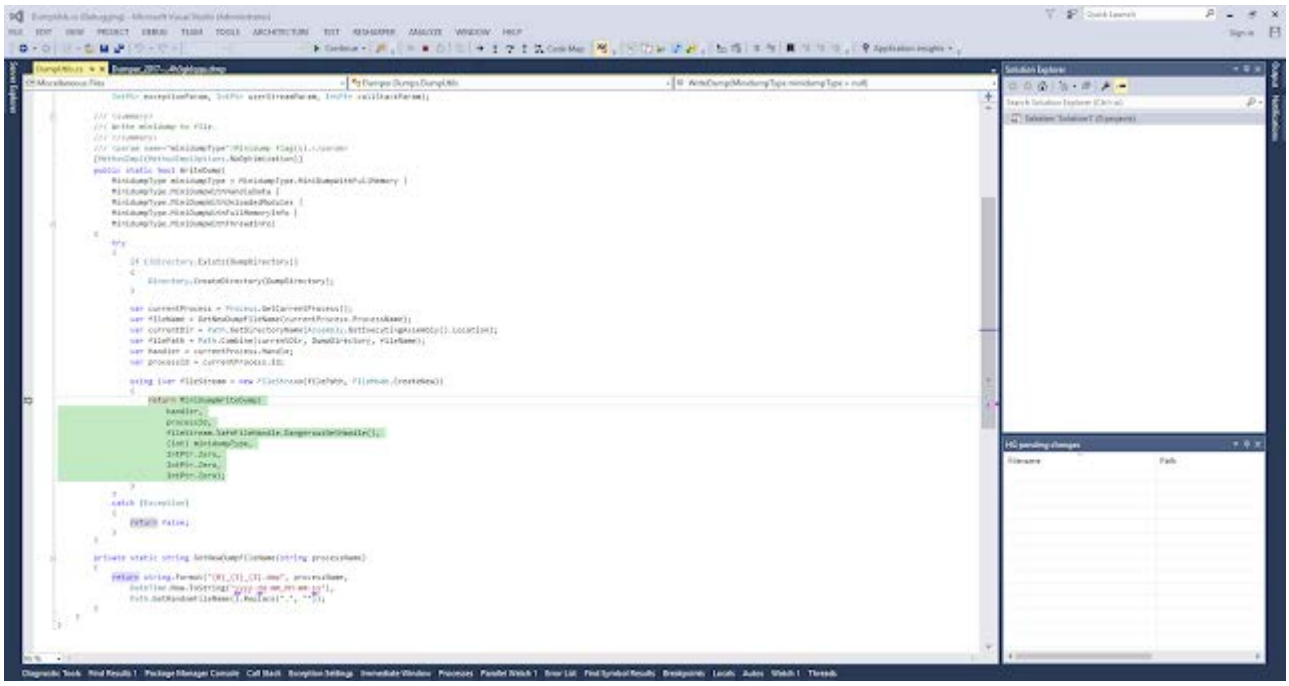
Отрываем файл в **Visual Studio**. (Я использую **Visual Studio** 2015, но проблем с более ранними версиями не должно быть). После открытия мы видим такую картинку:



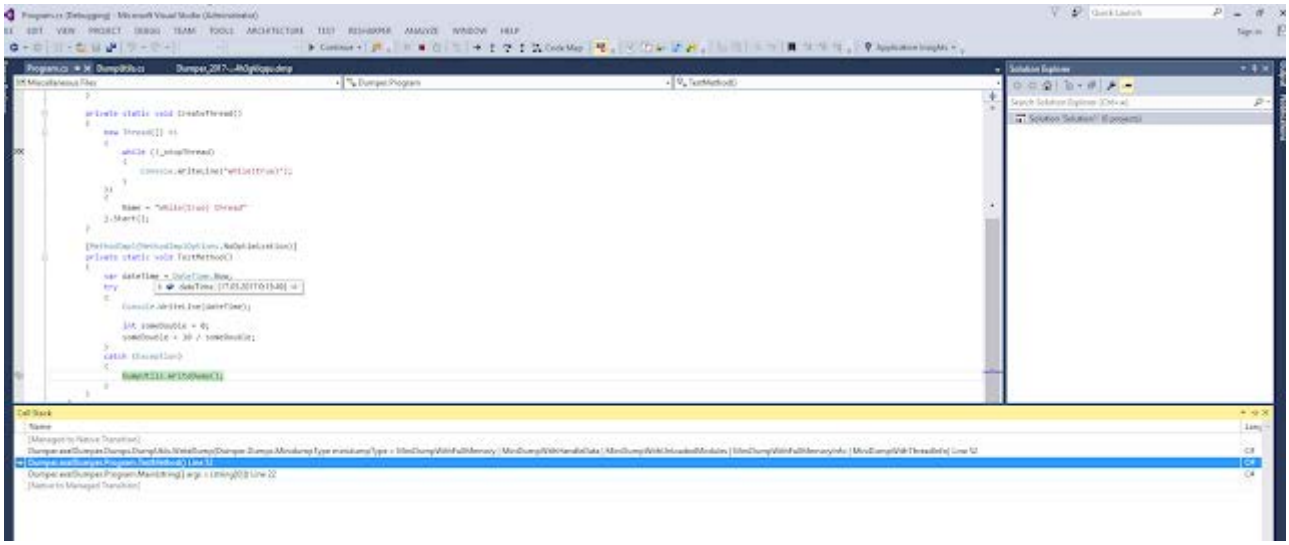
Снизу видим информацию об отработавшем приложении, далее жмем пункт меню **"Debug with Managed Only"**



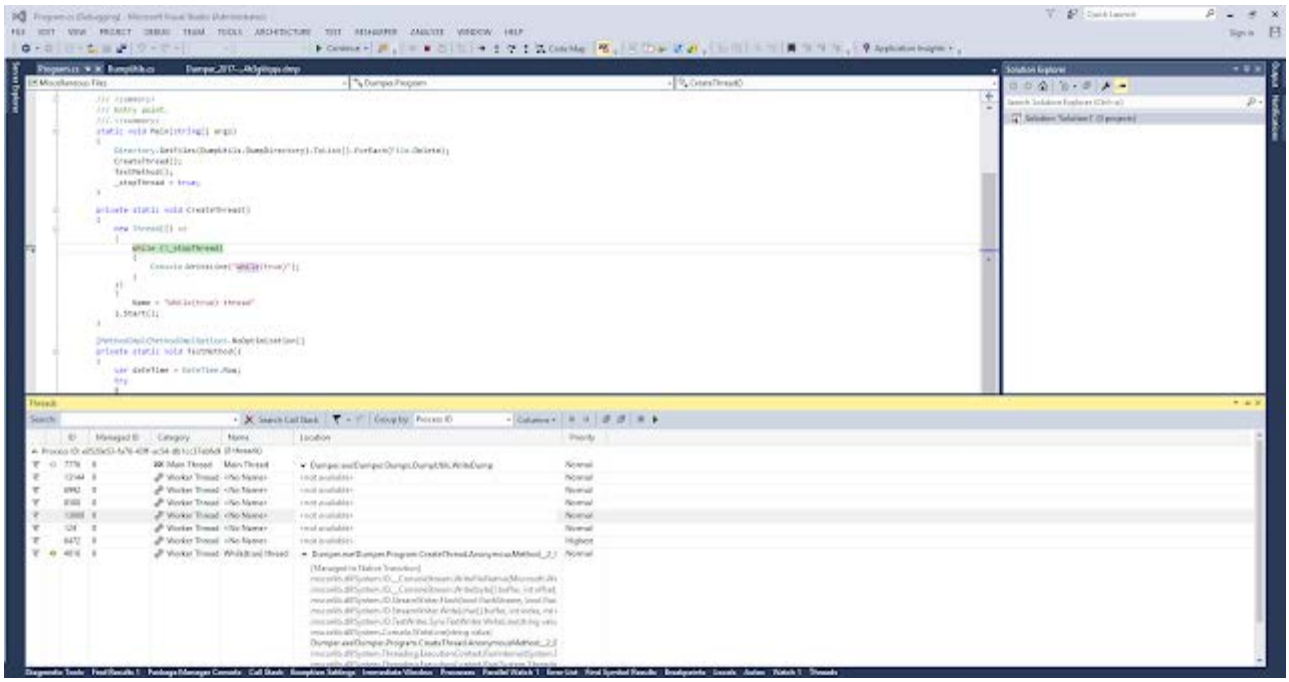
И видим, что мы как будто бы встали на точку останова перед вызовом **MiniDumpWriteDump**:



Отрыв окно **"Stack"** мы можем подняться выше и увидеть всё, в плоть до значений локальной переменной **dateTime** момент вызова:



И самое классное это то, что при полном дампе сохраняются стеки уже работающих потоков. Поэтому открыв окно **"Threads"** мы видим нам поток, созданный в начале запуска, и все остальные:



Согласитесь, выглядит круто? Но такую свободу нам дает **"Full dump"** с сохранением информации о всех потоков. При этом файл получается достаточно большой, даже для такого небольшого приложения. Для урезанных дампов необходимо поэкспериментировать с флагами MinidumpType, выбрав только те, которые сохранят нужные вам данные. Информацию о всех флагах можно найти тут MINIDUMP_TYPE enumeration.

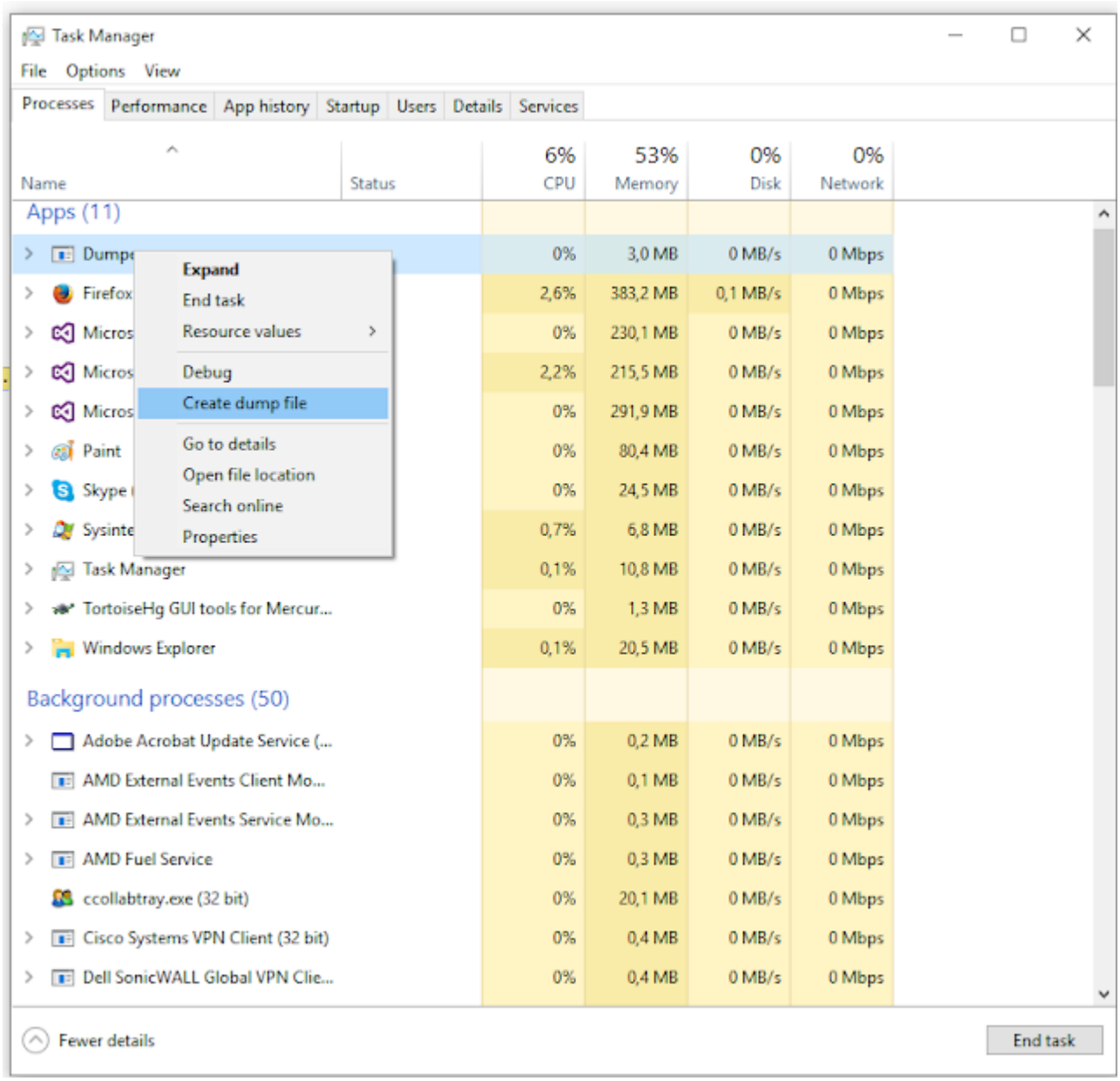
Создание Minidump используя сторонние приложения

Главный недостаток созданного таким образом дампа это то, что мы получим остановку в произвольном месте кода. Как тогда можно использовать этот способ с пользой?

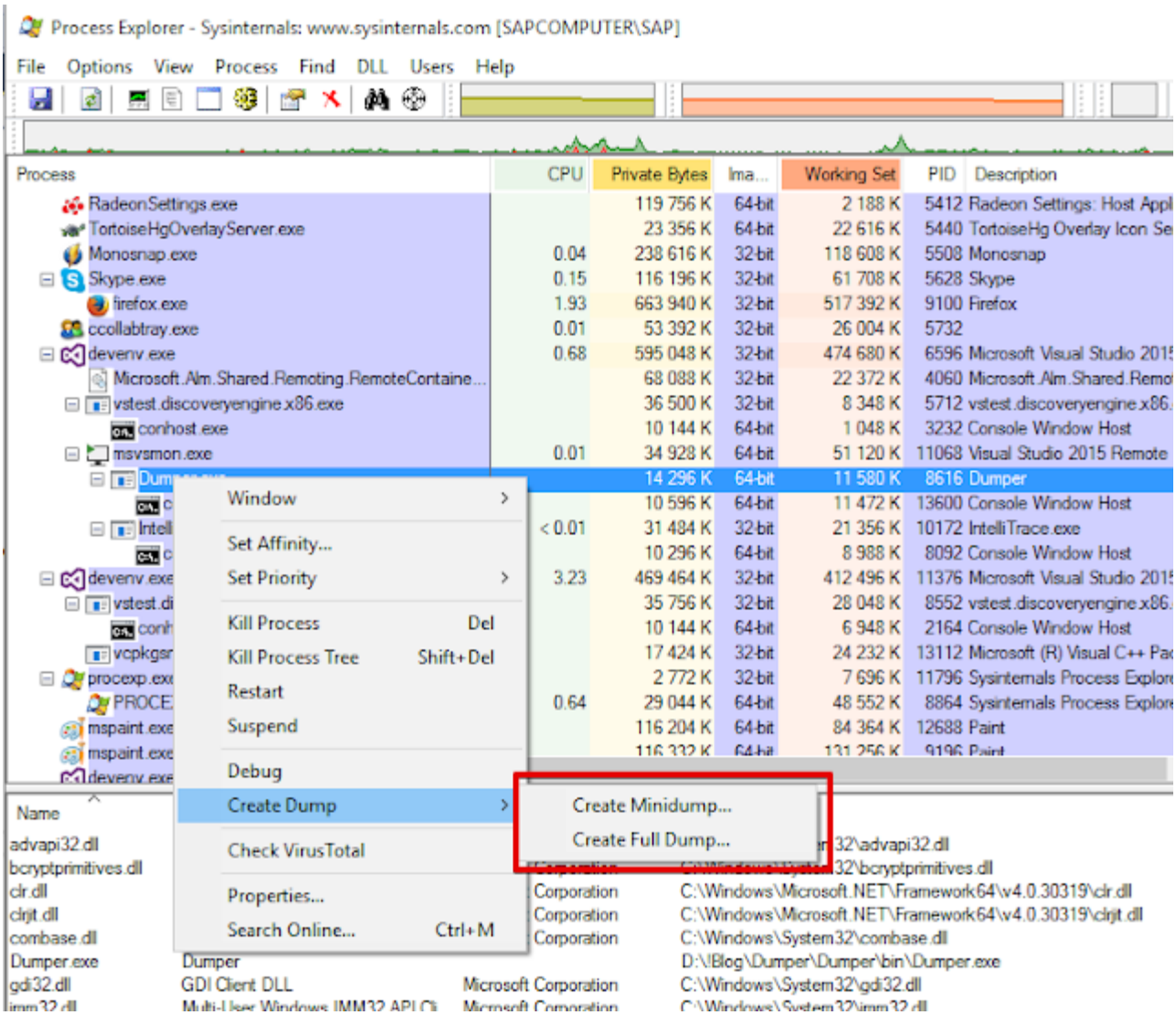
Предположим ситуация: *На тестовом сервере работает ваше приложение, которое неожиданно начинает ужасно сильно нагружать* ***CPU****. Тестировщик подходит к вам и просит разобраться. Можно воспользоваться описанными тут действиями "Поиск причины загрузки CPU приложения в слепую.". Или сделать дамп и на рабочей машине посмотреть, что происходит в работающем процессе.*
Перед созданием будьте готовы к тому, что полный дамп будет занимать в 3-5 раз больше места чем размер процесса в памяти и может занять длительное время.

Создание дампов памяти:

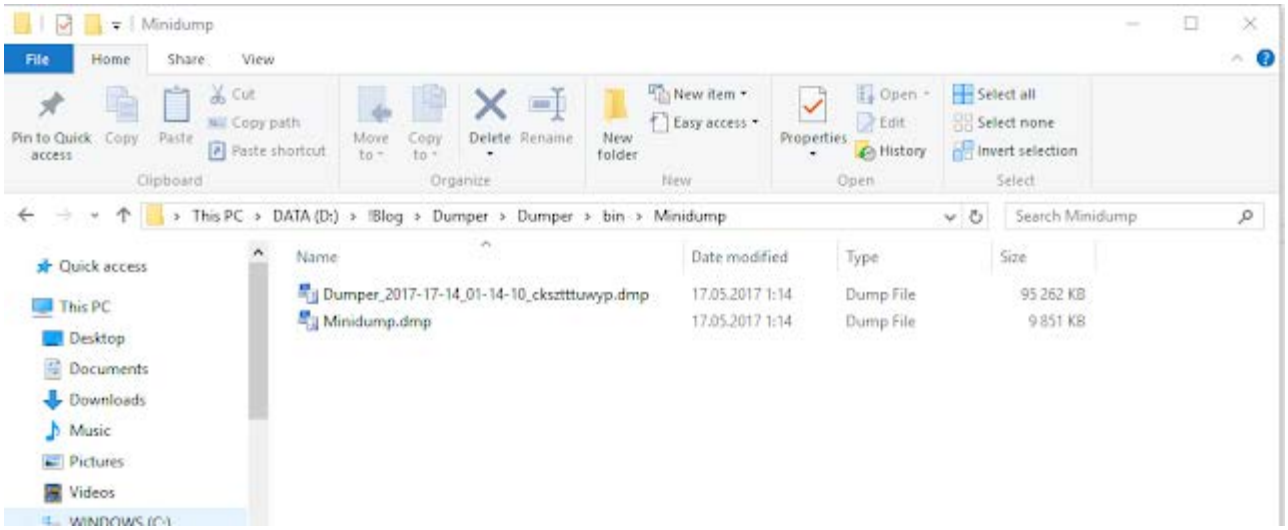
- Родной **Task Manager**. Нажимаем **CTRL+ALT+DEL** переходим на вкладку **"Processes",**



- Самым удобным, на мой взгляд, является [Process Explorer](). В нем можно создать как **"Full dump"** так и **"Minidump"** и это я считаю очень здорово.



Для демонстрации разницы "Полного дампа" и "Минидампа" приведу размеры созданных файлов:



Как видно разница размера почти в 10 раз, но при открытии в студии минидампа можно ожидать, что часть значений глобальных переменных и объектов в памяти может быть недоступна. Поэтому если основная часть глобальных переменных доступна, то нет потребности в полном дампе.