Оптимизация приложений на платформе .NET

с использованием языка C#

Саша Голдштейн, Дима Зурбалев, Идо Флатов

Apress®

ДМК Пресс издательство

УДК 004.438.NET
ББК 32.973.26-018.2
Г79

Г79 Голдштейн С., Зурбалев Д., Флатов И. и др.

Оптимизация приложений на платформе .NET. – Пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2014. – 524 с.: ил.

ISBN 978-5-94074-944-8

Увеличение производительности алгоритмов и приложений является чрезвычайно важным аспектом разработки и может дать вам преимущество перед конкурентами, а вашим пользователям обеспечить низкую стоимость владения и удовольствие от использования быстрых и отзывчивых приложений. Данная книга описывает внутренние особенности ОС Windows, среды выполнения CLR и аппаратного обеспечения, влияющие на производительность приложений, а также дает вам знания и инструменты для измерения производительности вашего кода в изоляции от внешних факторов.

Книга наполнена примерами кода на C# и рекомендациями, которые помогут вам выжать максимум возможного из вашего приложения – низкое потребление памяти, согласованную нагрузку на процессор и минимальное количество операций ввода/вывода с сетью и диском.

Издание предназначено для программистов, знакомых с языком C# и платформой .NET.

УДК 004.438.NET
ББК 32.973.26-018.2

ISBN 978-143-024-458-5 (англ.)
ISBN 978-5-94074-944-8 (рус.)

ОГЛАВЛЕНИЕ

ГЛАВА 8.
Небезопасный код и взаимодействие с ним ... 364

ГЛАВА 9.
Оптимизация алгоритмов ... 400

ГЛАВА 2. Измерение производительности

Эта книга посвящена вопросам повышения производительности приложений для .NET. Следует понимать, что нельзя улучшить какие-либо характеристики, не измерив их предварительно. Именно поэтому первая наиболее существенная глава посвящена инструментам и приемам измерения производительности. Построение догадок и преждевременных выводов об узких местах в приложении – это самое худшее, что может сделать разработчик. Как было показано в главе 1, существует множество любопытных характеристик производительности, которые могут играть важную роль в оценке производительности приложения в целом, а в этой главе мы узнаем, как измерять их.

Подходы к измерению производительности

Измерение производительности приложений может выполняться разными способами, во многом зависящих от контекста, сложности приложения, типа требуемой информации и точности получаемых результатов.

Один из подходов к тестированию небольших программ или библиотечных методов называется *тестированием по принципу «стеклянного ящика»*. Этот подход основан на исследовании исходного кода, анализе его сложности, изменении и добавлении кода, выполняющего измерения. Этот подход, который иногда называется *микрохронометраж* (microbenchmarking), будет обсуждаться ближе к концу этой главы; он особенно ценен – и часто незаменим – когда требуется высочайшая точность результатов хронометража неболь-

ших фрагментов кода, где каждая машинная инструкция на счету, но требует больших трудозатрат при оценке больших приложений. Кроме того, не зная наперед, какие фрагменты программ следует тестировать, выявление узких мест без применения автоматизированных инструментов может оказаться чрезвычайно трудной задачей.

Для больших программ чаще используется подход, называемый *тестированием по принципу «черного ящика»*, когда параметры производительности определяются человеком и затем измеряются с применением инструментов. Применяя этот подход, разработчик не должен строить какие-либо гипотезы об узких местах в приложении. В этой главе мы познакомимся с большим количеством инструментов, автоматически анализирующих производительность приложения и предоставляющих результаты измерений в простом и понятном виде. В числе этих инструментов будут упомянуты *счетчики производительности* (performance counters), *механизм трассировки событий для Windows* (Event Tracing for Windows, ETW) и коммерческие *профилировщики*.

В процессе чтения этой главы помните, что инструменты измерения производительности сами могут отрицательно влиять на производительность. Лишь немногие из них способны дать точную информацию, не добавляя свои накладные расходы. Перемещаясь от одного инструмента к другому, всегда помните, что точность инструмента часто искажается накладными расходами, которые они вносят при применении к приложению.

Встроенные инструменты Windows

Прежде, чем обратиться к коммерческим инструментам, требующим предварительной установки, познакомимся сначала с инструментами, которые может предложить Windows «из коробки». Счетчики производительности (performance counters) являются составной частью Windows вот уже почти два десятилетия. Не так давно (в 2006 г.) в Windows Vista появился еще один инструмент хронометража – механизм трассировки событий для Windows (Event Tracing for Windows). Оба входят в состав всех разновидностей Windows и могут использоваться для оценки производительности с минимальными накладными расходами.

Счетчики производительности

Счетчики производительности – это встроенный механизм Windows, позволяющий оценивать производительность и состояние системы в целом. С помощью счетчиков производительности пользователи и администраторы могут исследовать работу различных компонентов, включая ядро Windows, драйверы, базы данных и CLR. Как дополнительное преимущество, счетчики производительности для подавляющего большинства компонентов системы уже включены по умолчанию, поэтому вам не придется вносить поправки на дополнительные накладные расходы, связанные с их использованием.

Прочитать информацию из счетчиков производительности в локальной или удаленной системе чрезвычайно просто. Встроенный инструмент *Performance Monitor* (Системный монитор) (*perfmon.exe*) может отображать все счетчики производительности, доступные в системе и сохранять информацию в файле для последующего изучения, а также автоматически генерировать оповещения, когда значение того или иного счетчика производительности превышает некоторый установленный порог. Инструмент Performance Monitor (Системный монитор) может также подключаться к удаленным системам, если вы обладаете привилегиями администратора и возможностью подключения к удаленной системе по локальной сети.

Информация о производительности имеет иерархическую организацию, как описывается ниже.

Категории счетчиков производительности (или *объектов производительности*) представляют наборы отдельных счетчиков для определенных компонентов системы. В качестве примеров категорий можно привести: **.NET CLR Memory** (Память CLR .NET), **Processor Information** (Процессор), **TCPv4** и **PhysicalDisk** (Физический диск)[1].

- *Счетчики производительности* – это отдельные числовые свойства в категориях. Обычно принято указывать категорию и название счетчика производительности, разделяя их обратным слешем, например, **Process\Private Bytes** (Процесс\Байт исключительного пользования). Счетчики производительности могут иметь разные типы, включая простые числовые значения (**Process\Thread Count** (Процесс\Счетчик потоков)), скорости следования событий (**Print Queue\Bytes Printed/sec** (Очередь печати\Печатаемых байт/сек)), проценты

1 Здесь и далее перевод пунктов взят из русифицированной версии Windows.

(**PhysicalDisk\%Idle Time** (Физический диск\% активности диска)) и средние значения (**ServiceModelOperation 3.0.0.0\ Calls Duration** (Показатели работы служб 3.0.0.0\Продолжительность вызова)).

- *Экземпляры категорий счетчиков производительности* (вхождения) используются с целью создания разных наборов счетчиков для разных экземпляров компонентов системы. Например, в системе может иметься несколько процессоров, поэтому для каждого из них имеется свой экземпляр в категории **Processor Information** (Сведения о процессоре), а также общий экземпляр `_Total`). Одни категории счетчиков производительности могут иметь несколько экземпляров (таковых большинство), другие – единственный экземпляр (например, категория **Memory** (Память)).

Исследовав полный список счетчиков производительности, предоставляемых типичной системой Windows, где выполняются приложения для .NET, легко понять, что многие проблемы производительности могут быть выявлены без использования других инструментов. По крайней мере, счетчики производительности позволяют получить общее представление о причинах низкой производительности, а исследование файлов журналов поможет понять, насколько поведение системы отличается от нормального.

Ниже перечислены ситуации, когда системный администратор или разработчик, занимающийся проблемами производительности, сможет получить представление о том, где находится узкое место в приложении, прежде чем применить более мощные инструменты.

- Если в приложении присутствуют утечки памяти, счетчики производительности помогут выяснить, какие операции выделения памяти – низкоуровневые или управляемые – являются источником этих утечек. Для этого достаточно сопоставить счетчик **Process\Private Bytes** (Процесс\Байт исключительного пользования) со счетчиком **.NET CLR Memory\# Bytes in All Heaps** (Память CLR .NET\Байт во всех кучах). Первый подсчитывает объем всей памяти, выделенной для процесса (включая кучу сборщика мусора), а второй – только объем управляемой памяти. (См. рис. 2.1.)
- Если приложение ASP.NET начинает проявлять необычное поведение, счетчики из категории **ASP.NET Applications** (Приложения ASP.NET) позволят уточнить, что именно пошло не так. Например, счетчики **Requests/Sec** (Запросов/сек),

Requests Timed Out (Запросов с истекшим временем ожидания), **Request Wait Time** (Запросов в очереди приложений) и **Requests Executing** (Выполняется запросов) помогут выявить состояния пиковых нагрузок. Счетчик **Errors Total/Sec** (Общее число ошибок/сек) покажет, не столкнулось ли приложение с необычно большим количеством исключений. А различные счетчики, имеющие отношение к механизму кеширования покажут, насколько эффективно работает этот механизм.

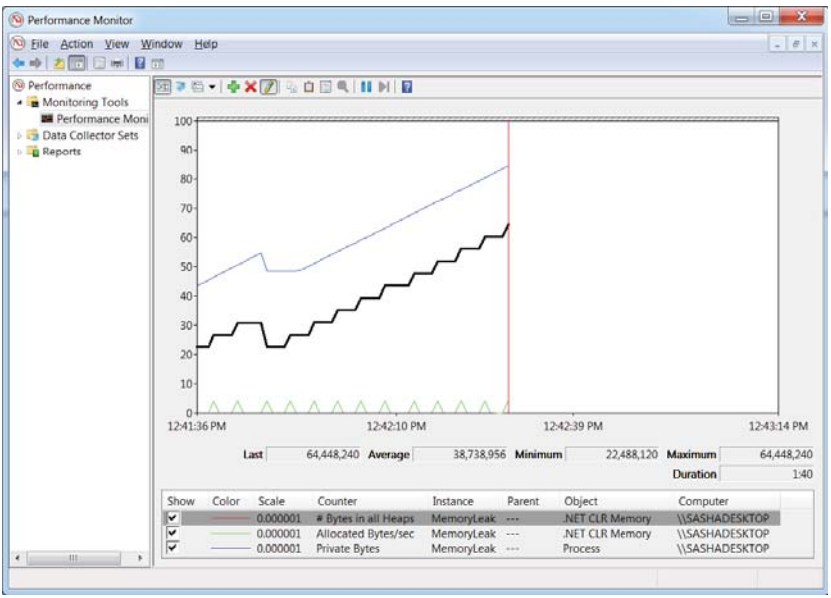

Рис. 2.1. Главное окно программы Performance Monitor (Системный монитор) с тремя счетчиками для определенного процесса. Верхняя линия на графике – значения счетчика Process\Private Bytes (Процесс\Байт исключительного пользования), средняя линия – значения счетчика .NET CLR Memory\# Bytes in all Heaps (Память CLR .NET\Байт во всех кучах), и нижняя линия – значения счетчика .NET CLR Memory\Allocated Bytes/sec (Память CLR .NET\Выделено байт/сек). На основании этого графика можно заключить, что в приложении имеется утечка памяти в куче сборщика мусора.

- Если WCF-служба, опирающаяся на взаимодействия с базами данных и распределенные транзакции, оказывается не в состоянии справиться с текущей нагрузкой, уточнить источник проблем поможет категория **Service Model Service**

(Показатели работы служб) – счетчики **Calls Outstanding** (Текущих вызовов), **Calls Per Second** (Вызовов/сек) и **Calls Failed Per Second** (Неудачных вызовов/сек) помогут идентифицировать состояние тяжелой нагрузки, счетчик **Transactions Flowed Per Second** (Транзакций в данной операции/сек) покажет частоту транзакций, выполняемых службой. А счетчики из категорий, имеющих отношение к SQL Server, таких как **MSSQL$INSTANCENAME:Transactions** и **MSSQL$INSTANCENAME:Locks** укажут на проблемы выполнения транзакций, чрезмерное количество блокировок и даже взаимоблокировок.

Мониторинг использования памяти с применением счетчиков производительности

В этом коротком эксперименте предлагается провести мониторинг использования памяти демонстрационным приложением и с помощью программы Performance Monitor (Системный монитор) выявить наличие утечек памяти, как описывалось выше.

1. Запустите программу Performance Monitor (Системный монитор) – это можно сделать, отыскав пункт **Performance Monitor** (Системный монитор) в меню **Start** (Пуск) или запустив программу *perfmon.exe* непосредственно.
2. Запустите приложение *MemoryLeak.exe* из папки с примерами для этой главы.
3. Щелкните на пункте **Performance Monitor** (Системный монитор) в панели слева и затем щелкните на кнопке с изображением зеленого плюса (+).
4. В категории **.NET CLR Memory** (Память CLR .NET) выберите счетчики **# Bytes in all Heaps** (Байт во всех кучах) и **Allocated Bytes/sec** (Выделено байт/сек), в списке справа выберите экземпляр **MemoryLeak** и щелкните на кнопке **Add >>** (Добавить >>).
5. В категории **Process** (Процесс) выберите счетчик **Private Bytes** (Байт исключительного пользования), в списке справа выберите экземпляр **MemoryLeak** и щелкните на кнопке **Add >>** (Добавить).
6. Щелкните на кнопке **OK**, чтобы подтвердить свой выбор и понаблюдайте за изменениями на графике.
7. Вам может понадобиться щелкнуть правой кнопкой мыши на строке со счетчиком в таблице, находящейся внизу окна и выбрать пункт контекстного меню **Scale selected counters** (Масштабировать выделенные счетчики), чтобы линии появились на графике.

Вы должны увидеть, что линии, соответствующие счетчикам **Private Bytes** (Байт исключительного пользования) и **# Bytes in all Heaps** (Байт во всех кучах) изменяются синхронно (как на рис. 2.1). Это указывает на утечки памяти в управляемой куче. Мы еще вернемся к данному примеру в главе 4 и раскроем причину утечки.

Совет. *В типичной системе Windows существуют, буквально, тысячи счетчиков производительности. И ни один, даже самый опытный разработчик, не в состоянии запомнить назначение их всех. Поэтому в диалоге «Add Counters» (Добавить счетчики) есть возможность отметить флажок «Show description» (Отображать описание). Когда флажок установлен, в нижней части окна будет отображаться дополнительное описание, которое сообщит, например, что счетчик «System\Processor Queue Length» (Система\Длина очереди процессора) – это количество потоков выполнения, ожидающих своей очереди, или, что счетчик «.NET CLR Locks And Threads\Contention Rate/sec» (Блокировки и потоки .NET CLR\Частота конфликтов/сек) – это количество неудачных попыток (в секунду) предпринятых потоками выполнения, чтобы получить управляемую блокировку.*

Журналы и оповещения производительности

Добавить сохранение в журнал значений счетчиков производительности очень просто, и есть даже возможность передать системным администраторам XML-шаблон, чтобы они с его помощью могли добавить автоматическую запись счетчиков без необходимости делать это вручную. После записи данных, журнал можно открыть на любом компьютере и проиграть его, как если это были оперативные данные. (Есть даже некоторые встроенные наборы счетчиков, которые не требуется настраивать вручную для записи в журнал.)

Инструмент Performance Monitor (Системный монитор) позволяет также определять настройки оповещений – выполнения определенных заданий при превышении указанными счетчиками установленных пороговых значений. Данную возможность можно использовать для создания упрощенной инфраструктуры мониторинга, способной отправлять электронные письма или сообщения системному администратору при нарушении ограничений производительности. Например, оповещение можно настроить так, что оно автоматически будет перезапускать процесс при достижении опасного предела используемого объема памяти, или когда система исчерпает все свободное пространство на диске. Мы настоятельно рекомендуем поэкспериментировать с системным монитором, чтобы поближе познакомиться с предлагаемыми им возможностями.

Настройка записи значений счетчиков в журнал

Чтобы настроить запись значений счетчиков в журнал, откройте Performance Monitor (Системный монитор) и выполните описываемые ниже действия. (Здесь предполагается, что вы пользуетесь Windows 7 или Windows Server 2008 R2. В предыдущих версиях операционной системы системный монитор имел несколько иной интерфейс – если вы пользуетесь такими версиями, обращайтесь к документации за более подробными инструкциями.)

1. В дереве слева разверните ветку **Data Collector Sets** (Группы сборщиков данных).
2. Щелкните правой кнопкой мыши на пункте **User Defined** (Определяемые пользователем) и выберите пункт **New → Data Collector Set** (Создать → Группа сборщиков данных) контекстного меню.
3. Введите имя группы, выберите радиокнопку **Create manually (Advanced)** (Создать вручную (для опытных)) и щелкните на кнопке **Next** (Далее).
4. Выберите радиокнопку **Create data logs** (Создать журналы данных), отметьте флажок **Performance counter** (Счетчик производительности) и щелкните на кнопке **Next** (Далее).
5. Щелкните на кнопке **Add** (Добавить) и добавьте счетчики производительности (в открывшемся стандартном диалоге **Add Counters** (Добавить счетчики)). Закончив добавление, настройте значение в поле **Sample Interval** (Интервал выборки) (по умолчанию замеры производятся один раз в 15 секунд) и щелкните на кнопке **Next** (Далее).
6. Укажите каталог, где будут сохраняться журналы и щелкните на кнопке **Next** (Далее).
7. Выберите радиокнопку **Open properties for this data collector set** (Открыть свойства группы сборщиков данных) и щелкните на кнопке **Finish** (Готово).
8. В открывшемся диалоге выполните настройки на разных вкладках – здесь можно определить расписание для автоматического запуска, условия останова (например, после сбора определенного объема данных) и задание, которое следует запустить после прекращения сбора данных (например, выгрузить результаты в централизованное хранилище). Завершив настройки, щелкните на кнопке **OK**.
9. Разверните ветку дерева **User Defined** (Определяемые пользователем), щелкните правой кнопкой мыши на вновь созданной группе сборщиков данных и выберите пункт контекстного меню **Start** (Пуск).
10. В результате начнется накопление данных в журнале, хранящемся в выбранном вами каталоге. Сбор данных можно остановить в любой момент, щелкнув на группе правой кнопкой мыши и выбрав пункт контекстного меню **Stop** (Стоп).

Когда после завершения сбора данных вам потребуется исследовать их с помощью системного монитора, выполните следующие действия:

1. Разверните ветку дерева **User Defined** (Определяемые пользователем).
2. Щелкните правой кнопкой мыши на группе сборщиков данных и выберите пункт контекстного меню **Latest Report** (Последний отчет).
3. В появившемся окне вы сможете добавить или удалить счетчики из списка в журнале, настроить диапазон изменения времени и масштаб изменения данных, щелкнув правой кнопкой мыши на графике и выбрав пункт контекстного меню **Properties** (Свойства).

Наконец, чтобы проанализировать данные на другом компьютере, необходимо скопировать каталог с журналами на этот компьютер, открыть ветку дерева Performance Monitor (Системный монитор) и щелкнуть на второй кнопке слева в панели инструментов (или нажать комбинацию клавиш **Ctrl** + **L**). В появившемся диалоге выберите радиокнопку **Log files** (Файлы журнала) и добавьте файлы с помощью кнопки **Add** (Добавить...).

Собственные счетчики производительности

Системный монитор – чрезвычайно удобный инструмент, однако значения счетчиков производительности можно читать из любого приложения для .NET, с помощью класса `System.Diagnostics.PerformanceCounter`. Более того, можно даже создавать собственные счетчики производительности и добавлять их к множеству уже имеющихся.

Ниже описаны некоторые ситуации, когда может пригодиться создание собственных категорий счетчиков:

- При разработке библиотеки, используемой как часть большой системы. Посредством счетчиков библиотека может сообщать информацию о производительности, что часто намного проще для разработчиков и системных администраторов, чем копаться в файлах журналов или в исходном коде.
- При разработке серверной системы, принимающей нестандартные запросы, обрабатывающей их и возвращающей ответы (например, нестандартного веб-сервера или веб-службы), может потребоваться оценить скорость обработки запросов, количество встречающихся ошибок и другие похожие параметры. (Дополнительные подсказки можно найти в категории счетчиков производительности ASP.NET.)
- При разработке высоконадежной службы Windows, которая выполняется без контроля со стороны человека и обменивается данными с нестандартным оборудованием. С помощью счетчиков производительности служба может сообщать о состоянии этого оборудования, о частоте следования операций обмена с ним и другие параметры.

Следующий фрагмент кода – это все, что необходимо, чтобы экспортировать из приложения категорию счетчиков производительности, имеющую единственный экземпляр, и обновлять их периодически. Предполагается, что класс `AttendanceSystem` хранит информацию о количестве зарегистрировавшихся к настоящему моменту пользователей, и вы хотите экспортировать эту информацию в виде счетчика производительности. (Чтобы скомпилировать этот фрагмент, потребуется добавить пространство имен `System.Diagnostics`.)

```
public static void CreateCategory() {
    if (PerformanceCounterCategory.Exists(“Attendance”)) {
        PerformanceCounterCategory.Delete(“Attendance”);
    }
    CounterCreationDataCollection counters = new CounterCreation-
```

```
DataCollection();
    CounterCreationData employeesAtWork = new CounterCreationData(
       "# Employees at Work", "The number of employees currently checked in.",
        PerformanceCounterType.NumberOfItems32);
    PerformanceCounterCategory.Create(
        "Attendance", "Attendance information for Litware, Inc.",
        PerformanceCounterCategoryType.SingleInstance, counters);
}

public static void StartUpdatingCounters() {
    PerformanceCounter employeesAtWork = new PerformanceCounter(
        "Attendance", "# Employees at Work", readOnly: false);
    updateTimer = new Timer(_ = > {
        employeesAtWork.RawValue = AttendanceSystem.Current.EmployeeCount;
    }, null, TimeSpan.Zero, TimeSpan.FromSeconds(1));
}
```

Как видите, нужно совсем немного усилий, чтобы определить собственные счетчики производительности, и они могут предоставлять весьма важную информацию. Корреляции системного и пользовательского счетчиков производительности часто бывает достаточно, чтобы понять причины, вызывающие проблемы с производительностью или настройками.

> **Примечание.** *Системный монитор можно использовать и для сбора другой информации, не имеющей отношения к счетчикам производительности. Например, его можно применять для сбора информации о системных настройках – значений ключей из реестра, свойств объектов WMI и даже содержимого файлов на диске. Поддерживается также возможность захватывать данные, поставляемые провайдерами механизма ETW (о котором рассказывается далее) для последующего анализа. Используя XML-шаблоны, администраторы могут создавать группы сборщиков данных на других компьютерах и генерировать отчеты, выполнив всего несколько простых операций по настройке.*

Счетчики производительности позволяют получить массу интересной информации о производительности, но они не могут использоваться как высокопроизводительная инфраструктура мониторинга и журналирования. Не существует системных компонентов, способных обновлять счетчики производительности чаще нескольких раз в секунду, а сама программа Performance Monitor (Системны монитор) не позволяет читать значения счетчиков чаще, чем один раз в секунду. Если для анализа потребуется выполнять замеры каких-либо характеристик тысячи раз в секунду, счетчики производительности окажутся непригодными для этого. Теперь обратим внимание на механизм трассировки событий для Windows (Event Tracing for Windows,

ETW), специально спроектированный для высокоскоростного сбора данных самых разных типов (не только числовых).

Механизм трассировки событий для Windows

Механизм трассировки событий для Windows (Event Tracing for Windows, ETW) – это высокопроизводительный фреймворк регистрации событий, встроенный в Windows. По аналогии со счетчиками производительности, многие компоненты системы и инфраструктура поддержки приложений, включая ядро Windows и CLR, определяют *механизмы отправки событий* – информации о внутреннем состоянии компонентов. В отличие от счетчиков производительности, которые всегда активны, механизм ETW можно включать и выключать во время выполнения, чтобы накладные расходы на сбор и отправку информации оказывали влияние на производительность, только когда это действительно необходимо.

Одним из богатейших источников информации является *провайдер ядра* (kernel provider), который генерирует события в моменты запуска процессов и потоков, загрузки DLL, распределения блоков памяти, сетевых операций ввода/вывода и при выполнении трассировки стека. В табл. 2.1 приводится перечень некоторых наиболее интересных событий, сообщаемых ETW-провайдерами ядра и CLR. Механизм ETW можно использовать для исследования общего поведения системы, например, чтобы выяснить, какой из процессов потребляет большую часть вычислительной мощности CPU, проанализировать узкие места в операциях ввода/вывода, получить статистику, касающуюся работы сборщика мусора и использования памяти управляемыми процессами, и во многих других случаях, обсуждаемых далее в этом разделе.

События ETW несут в себе точное время их возникновения, могут содержать дополнительную пользовательскую информацию, а также состояние стека на момент их появления. Информация о состоянии стека может использоваться для выявления источников различных проблем. Например, провайдер CLR может посылать события в начале и в конце каждого цикла сборки мусора. Эти события в комплексе с информацией о состоянии стека вызовов можно использовать для выявления частей программы чаще других вызывающих сборку мусора. (За дополнительной информацией о сборке мусора и событиях, ее вызывающих, обращайтесь к главе 4.)

Таблица 2.1. Неполный список событий ETW в ядре Windows и CLR

Провайдер	Флаг/ключевое слово	Описание	События
Ядро	PROC_THREAD	Запуск и завершение процессов и потоков	–
Ядро	LOADER	Загрузка и выгрузка образов (библиотек DLL, драйверов, выполняемых файлов)	–
Ядро	SYSCALL	Системные вызовы	–
Ядро	DISK_IO	Дисковые операции чтения и записи (включая позиционирование головок)	–
Ядро	HARD_FAULTS	Ошибки обращения к страницам диска (которые были вытеснены из кеша в оперативной памяти)	–
Ядро	PROFILE	Дискретное событие – сохранение информации о состоянии стека для всех процессоров выполняется через каждую 1 мсек	–
CLR	GCKeyword	Статистика и информация о работе механизма сборки мусора	Запуск сборки, конец сборки, запуск процедур завершения, выделение блока памяти ~100 Кбайт
CLR	ContentionKeyword	Конфликт между потоками выполнения при попытке приобрести разделяемую блокировку	Начало конфликта (поток переведен в режим ожидания), конец конфликта
CLR	JITTracingKeyword	Информация о состоянии динамического компилятора (Just in Time, JIT)	Успешная попытка встраивания метода, неудачная попытка встраивания метода
CLR	ExceptionKeyword	Возбужденное исключение	–

Для доступа к этой детализированной информации требуется специализированный инструмент и приложение, способное читать события ETW и выполнять простейший анализ. На момент написания этих строк существовало два инструмента, способных решать обе задачи: Windows Performance Toolkit (WPT, также известный как XPerf), распространяемый в составе Windows SDK, и PerfMonitor (не путайте с Windows Performance Monitor!) – открытый проект, разрабатываемый командой CLR в Microsoft.

Windows Performance Toolkit (WPT)

Windows Performance Toolkit (WPT) – это комплект утилит для управления сеансами ETW, сохранения событий ETW в файлах журналов и их обработки для последующего отображения на экране. Может генерировать графики и диаграммы событий ETW, сводные таблицы, включающие информацию о состоянии стека, и файлы CSV для автоматизированной обработки. Чтобы установить WPT, загрузите дистрибутив Windows SDK на странице http://msdn.microsoft.com/en-us/performance/cc752957.aspx, запустите мастер установки и выберите только **Common Utilities → Windows Performance Toolkit** (Общие утилиты → Windows Performance Toolkit). После установки перейдите в подкаталог *Redist\Windows Performance Toolkit* в каталоге установки SDK и запустите мастер установки для своей аппаратной архитектуры (*Xperf_x86.msi* – для 32-разрядных систем, *Xperf_x64.msi* – для 64-разрядных систем).

> **Примечание.** *В 64-разрядной версии Windows для поддержки возможности трассировки стека необходимо изменить настройки в реестре, запрещающие выгрузку страниц с кодом из оперативной памяти в файл подкачки (для самого ядра Windows и для всех драйверов). Это может увеличить потребление оперативной памяти системой на несколько мегабайт. Чтобы изменить настройки, найдите в реестре ключ* `HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management`*, установите параметр* `DisablePagingExecutive` *типа* `DWORD` *в значение* `0x1` *и перезагрузите систему.*

Для перехвата и анализа событий ETW используются инструменты *XPerf.exe* и *XPerfView.exe*. Оба должны запускаться с привилегиями администратора. Утилита *XPerf.exe* имеет несколько ключей командной строки, с помощью которых можно указать, какие провайдеры должны включаться, размеры используемых буферов, имя файла для сохранения информации о событиях и множество других параметров. Утилита *XPerfView.exe* анализирует исходную информацию и генерирует графические отчеты на основе информации в файле журнала.

Вместе с событиями может сохраняться также информация о состоянии стека вызовов, что часто помогает выявить дополнительные грани проблем производительности. Однако, чтобы получить информацию о состоянии стека совсем необязательно включать прием событий от какого-то определенного провайдера – флаг `SysProfile` позволяет получать эту информацию от всех процессоров с интервалом 1 мсек. Это упрощенный способ понять суть событий, протекающих в системе на уровне методов. (Мы еще вернемся к этому режиму, далее в этой главе, когда будем знакомится с дискретными профилировщиками.)

Захват и анализ событий ядра с помощью XPerf

В этом разделе предлагается выполнить трассировку событий ядра с помощью *XPerf.exe* и проанализировать полученные результаты с помощью *XPerfView.exe*. Данный эксперимент планировался для проведения в версии Windows Vista или выше. (Для его проведения требуется также настроить две системные переменные окружения: щелкните правой кнопкой мыши на ярлыке **Computer** (Компьютер), выберите пункт контекстного меню **Properties** (Свойства), щелкните на пункте **Advanced system settings** (Дополнительные параметры системы) в панели слева и в открывшемся диалоге – на кнопке **Environment Variables** (Переменные среды) внизу.)

1. Создайте системную переменную окружения `_NT_SYMBOL_PATH` со значением, включающим путь к общедоступному серверу символов и локальному кешу символов, например: `srv*C:\Temp\Symbols*http://msdl.microsoft.com/download/symbols`.
2. Создайте системную переменную окружения `_NT_SYMCACHE_PATH` со значением, включающим путь к локальному каталогу на диске – это должен быть другой каталог, отличный от того, что был указан в качестве локального кеша символов в предыдущем пункте.
3. Запустите с правами администратора программу **Command Prompt** (Командная строка) и перейдите в каталог установки WPT (например, *C:\Program Files\Windows Kits\8.0\Windows Performance Toolkit*).
4. Запустите прием событий из группы Base ядра, содержащие флаги `PROC_THREAD`, `LOADER`, `DISK_IO`, `HARD_FAULTS`, `PROFILE`, `MEMINFO` и `MEMINFO_WS` (см. табл. 2.1). Для этого выполните команду: `xperf -on Base`.
5. Сымитируйте некоторую активность в системе: запустите несколько приложений, попереключайтесь между окнами, попробуйте открыть какие-нибудь файлы – хотя бы несколько секунд. (Все это будет приводить к созданию отслеживаемых событий.)
6. Остановите прием событий и сохраните результаты в файл, выполнив команду: `xperf -d KernelTrace.etl`.
7. Запустите инструмент анализа, выполнив команду: `xperfview KernelTrace.etl`.

8. В появившемся окне вы увидите несколько графиков, по одному для каждого события ETW. Выбор графиков для отображения выполняется в панели слева. Обычно флажки, соответствующие графикам нагрузки на процессоры, находятся в самом верху, а ниже – флажки выбора графиков дисковых операций ввода/вывода, использования памяти и других статистик.
9. Щелкните правой кнопкой мыши на графике нагрузки на процессор и выберите пункт контекстного меню **Load Symbols** (Загрузить символы). Щелкните правой кнопкой мыши на графике еще раз и выберите пункт контекстного меню **Simple Summary Table** (Простая сводная таблица). В результате должна появиться таблица со списком методов во всех процессах, проявлявших активность в процессе сбора информации. (Загрузка символов с сервера Microsoft в первый раз может занять продолжительное время.)

Инструмент WPT способен на большее, чем было показано в этом эксперименте. Вам следует заняться самостоятельными исследованиями пользовательского интерфейса и попробовать принять и проанализировать другие группы событий ядра или даже события от собственных провайдеров ETW. (Создание собственных провайдеров рассматривается далее в этой главе.)

Инструмент WPT может пригодиться в самых ситуациях, поможет вникнуть в поведение системы и отдельных процессов. Ниже представлено несколько скриншотов и описаний примеров подобных ситуаций:

- WPT может перехватывать все события дисковых операций ввода/вывода в системе и выводить информацию с привязкой к карте физического диска. Это дает возможность выявить наиболее дорогостоящие операции ввода/вывода, в частности операции, требующие значительных перемещений головок жесткого диска. (См. рис. 2.2.)
- WPT может предоставить информацию о состоянии стеков вызовов для всех процессоров в системе. Он группирует стеки вызовов по процессам, модулям и функциям, что позволяет визуально оценить, где система (или конкретное приложение) проводит больше всего времени. Обратите внимание, что управляемые кадры стеков не поддерживаются – к этой проблеме мы еще вернемся ниже, когда будем знакомиться с инструментом PerfMonitor. (См. рис. 2.3.)
- WPT может отображать сводные графики событий разных типов, чтобы проще было выявить корреляцию, например, между операциями ввода/вывода, использованием памяти, нагрузкой на процессор и другими характеристиками. (См. рис. 2.4.)

- WPT может отображать сводную информацию о состоянии стеков вызовов (когда утилита приема событий запускалась с ключом `-stackwalk`) – это дает возможность получить полную информацию о стеках вызовов на момент создания определенны событий. (См. рис. 2.5.)

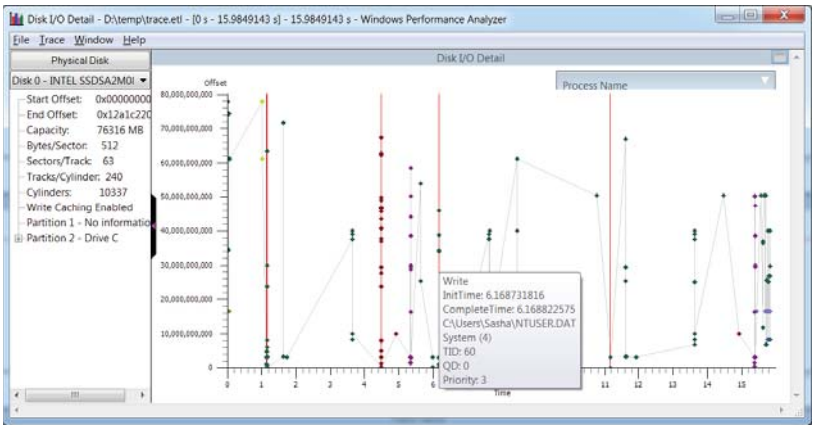

Рис. 2.2. Дисковые операции ввода/вывода, нанесенные на карту физического диска. Информация об операциях ввода/вывода и дополнительных деталях предоставляется в виде всплывающих подсказок.

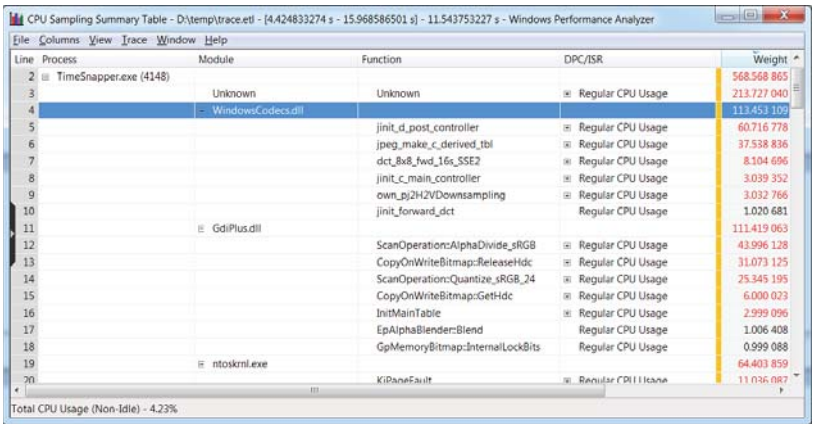

Рис. 2.3. Детальная информация о кадрах стека вызовов для единственного процесса (TimeSnapper.exe). Колонка Weight (Вес) показывает (примерно), как долго продолжалось выполнение в этом кадре.