# Содержание

[Содержание 2](#_Toc483914324)

[Используемая система и версия ядра 3](#_Toc483914325)

[WINDOWS 4](#_Toc483914326)

[Методы профилирования 5](#_Toc483914327)

[Профилирование средствами Microsoft Visual Studio 2010 7](#_Toc483914328)

[Профилирование в программе DR. Memory 10](#_Toc483914329)

[LINUX 11](#_Toc483914330)

[Утилита strace 12](#_Toc483914331)

[Утилита ltrace 13](#_Toc483914332)

[Инструмент Valgrind 14](#_Toc483914333)

[Источники 16](#_Toc483914334)

# Используемая система и версия ядра

a) Windows

|  |
| --- |
| Процессор: Intel(R) Core(TM) i5-2450 CPU @2.50GHz 2.50GHz  ОЗУ: 5,00 Гб  Тип системы: Windows 10 Pro x64. |

б) Linux

|  |
| --- |
| osboxes@osboxes:~/Documents$ **lsb\_release -a**  No LSB modules are available.  Distributor ID: Ubuntu  Description: Ubuntu 16.04.2 LTS  Release: 16.04  Codename: xenial  osboxes@osboxes:~/Documents$ **cat /proc/version**  Linux version 4.8.0-53-generic (buildd@lgw01-56) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4) ) #56~16.04.1-Ubuntu SMP Tue May 16 01:18:56 UTC 2017 |

# WINDOWS

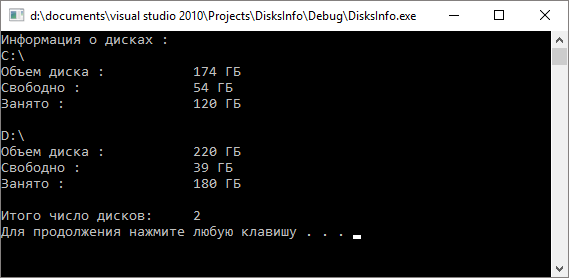
В этой и последующих лабораторных работах в качестве анализируемого приложения используется программа по расчету общего, свободного и занятого объема логических дисков на устройстве. Реализация для ОС Windows представления ниже.

Приложение использует функции WinAPI ***GetLogicalDriveStrings()*** для определения работающих логических дисков на компьютере и вывода их символьных значений, ***GetDriveType()*** для определения типов дисков, ***GetDiskFreeSpaceEx()*** для получения информации о количестве места на диске (в байтах).

Листинг . Программа для Windows

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #pragma warning(disable : 4996)  #include <stdio.h>  #include <iostream>  #include <Windows.h>  #include <locale>  using namespace std;  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  cout << "Информация о дисках : " << endl;  int k = 0;  int sz = GetLogicalDriveStrings(NULL, 0);  TCHAR\* szLogicalDrives = new TCHAR[sz];  GetLogicalDriveStrings(sz, szLogicalDrives);  while (\*szLogicalDrives)  {  TCHAR szDisk[80];  lstrcpy(szDisk, szLogicalDrives);  UINT uDriveType = GetDriveType(szDisk);  if (uDriveType == DRIVE\_FIXED) // если тип диска - жесткий диск  {  k = k + 1;  wcout << szDisk << endl;  \_int64 Free, Total;  Free = 0;  Total = 0;  GetDiskFreeSpaceEx(szDisk, NULL,(PULARGE\_INTEGER)&Total,(PULARGE\_INTEGER) &Free);  cout << "Объем диска: \t\t" << Total/(1024\*1024\*1024) << " ГБ" << endl;  cout << "Свободно: \t\t" << Free/(1024\*1024\*1024) << " ГБ" << endl;  cout << "Занято: \t\t" << (Total-Free)/(1024\*1024\*1024) <<" ГБ\n" << endl;  }  while (\*szLogicalDrives) szLogicalDrives++;  szLogicalDrives++;  }  std::wcout << L"Итого число дисков:\t" << k << std::endl;  system("pause");  return 0;  } |

Результат выполнения программы:



## Методы профилирования

Профилирование позволяет оценить производительность программы и решить основные проблемы:

* интенсивное использование ЦП;
* неэффективное использование подсистемы ввода/вывода;
* уровневое взаимодействие;
* выделение и использование памяти;
* излишняя/неправильная синхронизация, недостаточное использование ядер процессора.

**1. Sampling**

Метод Выборки – собирает статистические данные о работе приложения (во время профилирования). Этот метод **легковесный** и поэтому, в результате его работы очень маленькая погрешность в полученных данных.

Каждый определенный интервал времени собирается информация о стеке вызовов(call stack). На основе этих данные производится подсчет производительности. Используется для первоначального профилирования и для определения проблем, связанных с использованием ЦП.

**2. Instrumentation**

Метод собирает детализированную информацию о времени работы каждой вызванной функции. Используется для замера производительности операций ввода/вывода.

Метод внедряет свой код в двоичный файл, который фиксирует информацию о синхронизации (времени) для каждой функции в файл, и для каждой функции, которые вызываются в этой.

Отчет содержит 4 значения для предоставления затраченного времени:

* **Elapsed Inclusive** — общее время, затраченное на выполнение функции
* **Application Inclusive** — время, затраченное на выполнение функции, за исключением времени обращений к операционной системе.
* **Elapsed Exclusive** — время, затраченное на выполнение кода в теле. Время, которое тратят функции, вызванные целевой функцией.
* **Application Exclusive** — время, затраченное на выполнение кода в теле. Исключается время, которое тратится выполнения вызовов операционной системы и время, затраченное на выполнение функций, вызванные целевой функцией.

**3. Concurrency**

Метод собирает информацию о многопоточных приложениях. Метод собирает подробную информацию о стеке вызовов, каждый раз, когда конкурирующие потоки вынуждены ждать доступа к ресурсу.

**4. .NET Memory**

Профайлер собирает информацию о типе, размере, а также количество объектов, которые были созданы в распределении или были уничтожены сборщиком мусора. Профилирование памяти почти не влияет на производительность приложения в целом.

**5. Tier Interaction**

Метод добавляет информацию в файл для профилирования о синхронных вызовах ADO.NET между страницей ASP.NET или другими приложениями и SQL сервера. Данные включают число и время вызовов, а также максимальное и минимальное время.

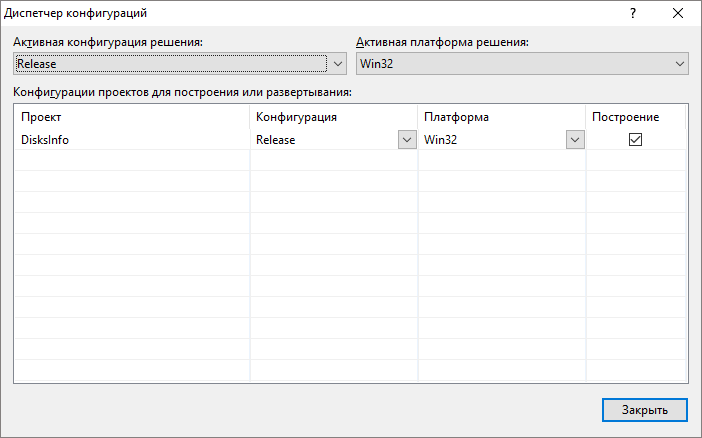
## Профилирование средствами Microsoft Visual Studio 2010

**Visual Studio Profiling Tool** позволяет разработчикам измерять, оценивать производительность приложения и кода. Эти инструменты полностью встроены в IDE, чтобы предоставить разработчику беспрерывный контроль.

В данной работе рассматривается профилирование приложения, используя **Sampling** и **Instrumentation** методы профилирования, чтобы выявить проблемы в производительности приложения.

Перед началом работы необходимо произвести подготовку, выполнив следующие шаги:

1. Запустить среду от имени администратора;
2. Установить активную сборку конфигурации (active build configuration) в значение «release» (в меню **Построение** выбрать **Диспетчер конфигурации** и в поле **Активная конфигурация решения** выбрать **Release**). В Debug версию встраивается дополнительная информация для отладки приложения, и она плохо скажется на точности результатов профилирования;

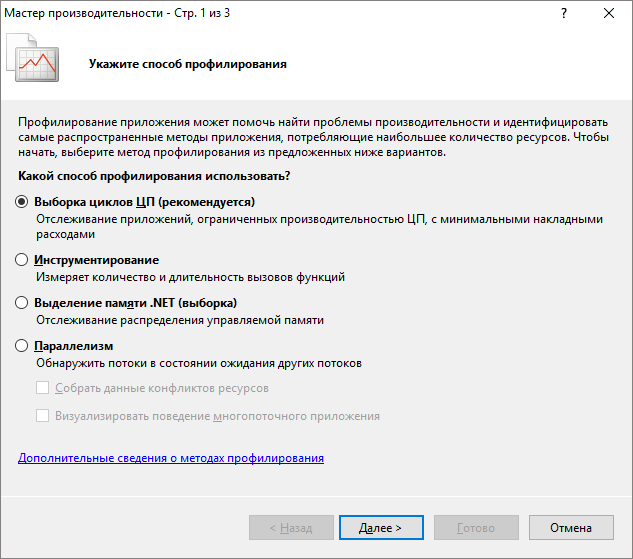


**Создание и запуск сессии профилирования**

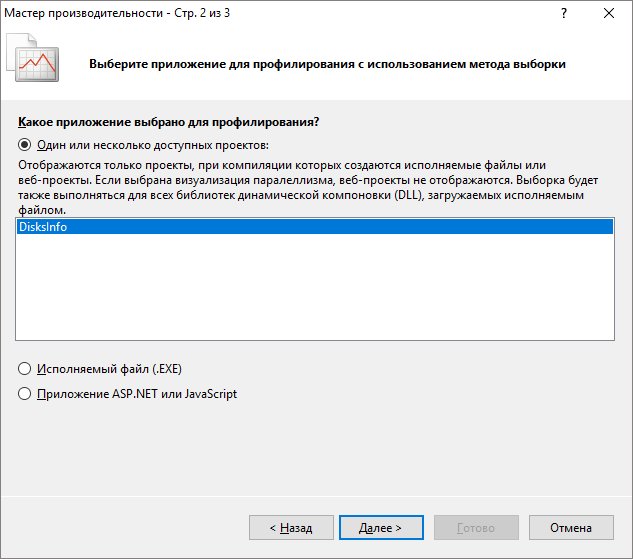
Для того, чтобы включить профилирования для текущего проекта или .exe-файла следует выполнить следующие шаги:

1. Открыть проект;
2. В пункте меню **Анализ** выбрать **Запустить мастер производительности…**;

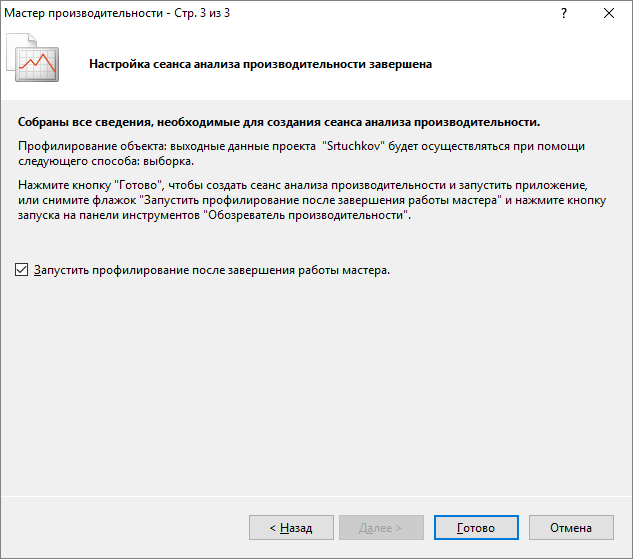
Откроется окно **Мастера производительности:**



1. Выбрать один из методов профилирования (например, **Выборка циклов ЦП**). Нажать кнопку **Далее**;



1. Указать открытый проект, как цель анализа, и нажать **Далее**;



1. Оставить галочку «**Запустить профилирование после завершения работы мастера**» и нажать **Готово**.

После этого будет выполнен запуск программы, а по ее завершению проведен результат выполнения профилирования. Профайлер сгенерирует отчет.

На моем рабочем устройстве не удалось завершить профилирование из-за следующих ошибок:

|  |
| --- |
| Ошибка VSP1398: Монитору не удалось запустить драйвер производительности VS. Загрузка драйвера была заблокирована Попробуйте использовать параметры /Admin:Driver,Start и /Admin:Security команды VSPerfCmd из среды с повышенными правами. Не удается продолжить профилирование.  Ошибка VSP1341: Не удается запустить обработчик журнала событий. Завершение работы.  Профилирование завершено.  PRF0010: Запуск прерван - не удалось запустить файл vsperfmon.exe  Ошибка VSP1712: недопустимый файл: d:\documents\visual studio 2010\Projects\DisksInfo\DisksInfo170530(2).vspPRF0025: Данные не собраны. |

Были предприняты попытки запустить инструменты профилирования на версии Visual Studio 2015. Завершить профилирование также не удалось:

|  |
| --- |
| Profiling started.  Profiling process ID 11756 (DisksInfo).  Starting data collection. The output file is d:\documents\visual studio 2015\Projects\DisksInfo\DisksInfo170530.vspx  Profiler stopping.  File not found:d:\documents\visual studio 2015\Projects\DisksInfo\DisksInfo170530.vspx |

## Профилирование в программе DR. Memory

DR. Memory – средство для оценивания работы программ с точки зрения работы с памятью. Проводятся проверки на утечки памяти, неправильную адресацию, выход за пределы памяти и другие ошибки, связанные с памятью.

*Примечание*: Dr. Memory существенно увеличивает время исполнения программы, т.к. производится анализ всех системных вызовов.

Приложение было скачано с официального сайта разработчика: <http://drmemory.org/>

# LINUX

Программа по предоставлению информации о дисках была также реализована для систем Linux. Код анализируемого приложения представлен в Листинге 2.

Листинг 2. Программа на Linux

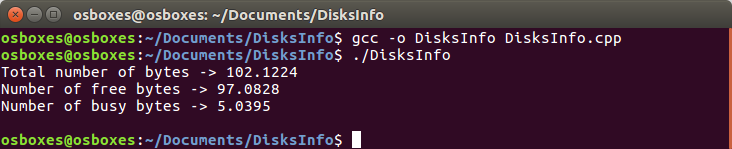
|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  #include <sys/vfs.h>  int main ()  {  struct statfs fs;  unsigned long long free, size, blocks;  statfs("/home/osboxes/Desktop/fstat/file\_name.txt", &fs);  free = fs.f\_bfree;  size = fs.f\_bsize;  blocks=fs.f\_blocks;  double blocks\_gb=blocks\*size/1000000000.;  double busy\_gb=(blocks-free)\*size/1000000000.;  double free\_gb=free\*size/1000000000.;  printf("Total number of bytes -> %.4f \n", blocks\_gb);  printf("Number of free bytes -> %.4f \n", free\_gb);  printf("Number of busy bytes -> %.4f \n", busy\_gb);  int c = getchar ();  return 0;  } |

Для получения информации о распределении объема диска используется системный вызов ***statfs***, который записывает в структуру statfs информацию о файловой системе. Используются следующие поля структуры:

* f\_bfree – количество свободных блоков в ФС;
* f\_bsize – оптимальный размер блока;
* f\_blocks – общее число блоков в ФС.

Программа выведет на консоль следующую информацию:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **gcc -o DisksInfo DisksInfo.cpp**  osboxes@osboxes:~/Documents/DisksInfo$ **./DisksInfo** |

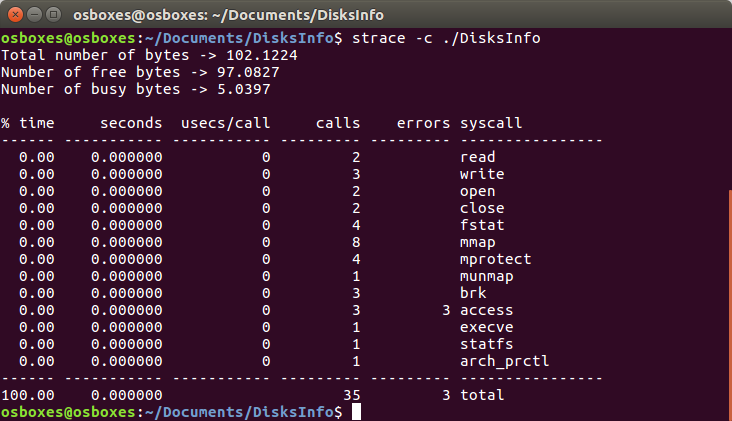


## Утилита strace

**Strace** - утилита для диагностики и отладки программ для ОС, использующих ядро Linux. Она позволяет отслеживать и (начиная c версии strace 4.15) вмешиваться в процесс взаимодействия программы и ядра, включая происходящие системные вызовы, возникающие сигналы и изменения состояния процесса. Для своей работы strace использует механизм ptrace.

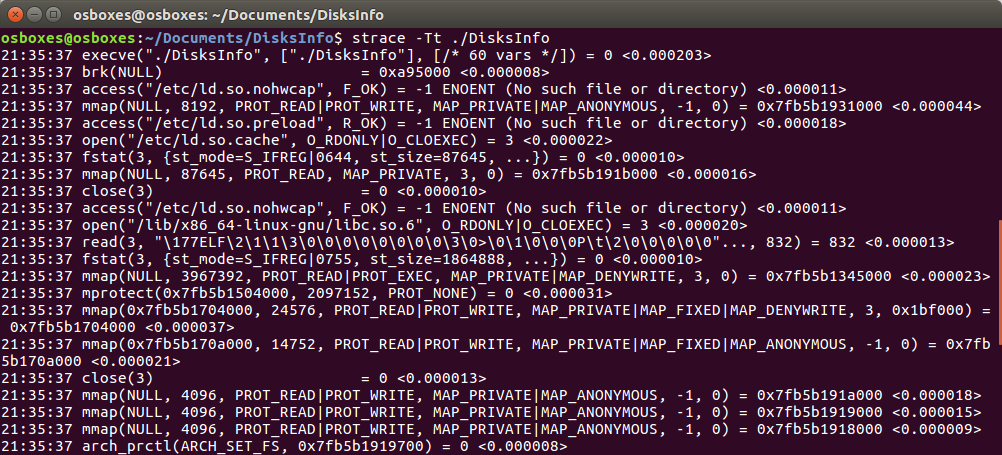
|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **strace -c ./DisksInfo** |

**флаг –c** – позволяет вывести информацию о количестве используемых методов:



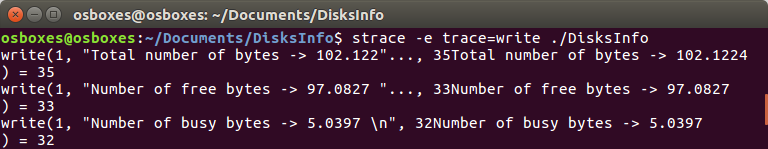
**флаг -Tt** – выводит время исполнения, с точностью до секунды:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **strace -Tt ./DisksInfo** |



**флаг -e** **trace=NAME** – позволяет вывести информацию только об 1 системном вызове (с именем NAME)

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **strace -e trace=write ./DisksInfo** |



## Утилита ltrace

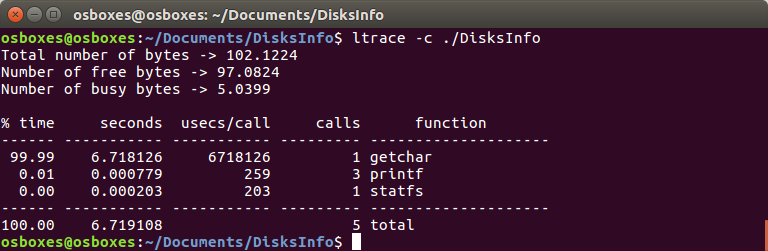
**Ltrace** – регистрирует вызовы динамических библиотек с целью отладки.

Программа ltrace предназначена **для отладки динамически собранных программ**. Отлаживаемый код запускается под управлением ltrace, при этом вызовы динамических библиотек, а также получаемые процессом сигналы перехватываются и регистрируются. Возможна также регистрация системных вызовов со стороны отлаживаемой программы.

Для отладки программы её не нужно перекомпилировать, поэтому возможно использование ltrace с программами, исходный текст которых не доступен.

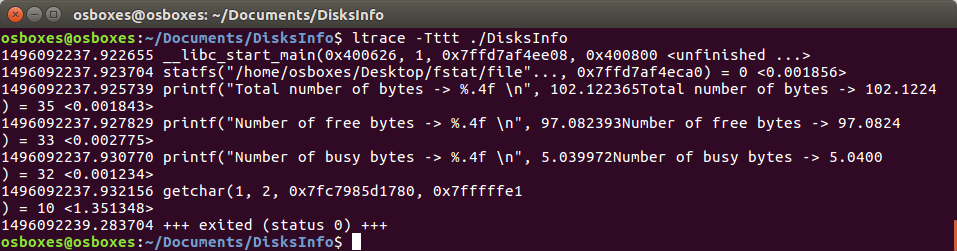
**флаг -c** – выводит системные вызовы динамически подключаемых библиотек:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **ltrace -c ./DisksInfo** |



**флаг -Tttt** – отображает точное время (с точностью до микросекунды) о времени совершения вызова:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **ltrace -Tttt ./DisksInfo** |



## Инструмент Valgrind

Valgrind является многоцелевым инструментом профилирования кода и отладки памяти для Linux. Это позволяет запускать программу в собственной среде Valgrind, что контролирует использование памяти, например, вызовы *malloc* и *free* (или *new* и *delete* в C++). Если вы используете неинициализированную память, записываете за пределами концов массива, или не освобождаете указатель, Valgrind может это обнаружить.

Для начала работы с Valgrind его необходимо проинсталлировать:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **sudo apt install valgrind** |

Без дополнительных аргументов, Valgrind выведет обзор вызовов *free* и *malloc*:

|  |
| --- |
| osboxes@osboxes:~/Documents/DisksInfo$ **valgrind ./DisksInfo**  ==34801== Memcheck, a memory error detector  ==34801== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.  ==34801== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info  ==34801== Command: ./DisksInfo  ==34801==  Total number of bytes -> 102.1224  Number of free bytes -> 96.4832  Number of busy bytes -> 5.6391  ==34801==  ==34801== HEAP SUMMARY:  ==34801== in use at exit: 0 bytes in 0 blocks  ==34801== total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated  ==34801==  ==34801== All heap blocks were freed -- no leaks are possible  ==34801==  ==34801== For counts of detected and suppressed errors, rerun with: -v  ==34801== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)  osboxes@osboxes:~/Documents/DisksInfo$ |

Одинаковое количество *allocs* и *frees* говорит о том, что в данной программе нет утечек памяти.

Помимо утечек памяти, Valgrind позволяет обнаруживать недопустимое использование указателей, использование неинициализированных переменных, неправильно выбранные методы освобождения памяти, а также многие другие случаи неправильного использования памяти.

# Источники

1. Информация по функциям WinAPI:  
   <http://rusproject.narod.ru/winapi/g/getdrivetype.html>
2. Профилирование приложений в Visual Studio 2010:  
   <https://habrahabr.ru/post/98361/>
3. Системные вызовы statfs, fstatfs:  
   <http://www.opennet.ru/man.shtml?topic=statfs&category=2&russian=0>
4. Утилита strace – Linux man page:  
   <https://linux.die.net/man/1/strace>
5. Страница загрузки Valgrind  
   URL: <http://www.valgrind.org/downloads/current.html>
6. Использование Valgrind для поиска утечек и недопустимого использования памяти:  
   http://cppstudio.com/post/4348/