**Курсовая работа по дисциплине «Информатика»**

**«Функции WinAPI для анализа информации»**

2019 г

Оглавление

[ВВЕДЕНИЕ 3](#_Toc9417454)

[1 Язык программирования C++ 5](#_Toc9417455)

[2 Среда разработки Visual Studio 7](#_Toc9417456)

[3 Библиотека «Windows Application Programming Interface (API)» 8](#_Toc9417457)

[4 Используемые функции Windows API 10](#_Toc9417458)

[5 Тестирование программы 14](#_Toc9417459)

[ЗАКЛЮЧЕНИЕ 17](#_Toc9417460)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 18](#_Toc9417461)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 19](#_Toc9417462)

# ВВЕДЕНИЕ

В настоящее время, с развитием научно-технического прогресса и информационных технологий, сложность автоматизируемых предметных областей постоянно и неуклонно возрастает. Программист проводит всё своё рабочее время, по сути, в борьбе со сложностью программ, отлаживая свой код в поисках допущенных ошибок. Ради борьбы со сложностью эволюционируют и языки программирования.

После машинных кодов и перфокарт появились текстовые редакторы и язык ассемблера, что позволило писать чуть более сложные и объёмные программы при меньших трудозатратах. Далее появились процедурные языки высокого уровня – C, Паскаль и многие другие, что перевело технологии программирования на новый уровень.

Следующим крупным скачком на пути борьбы со сложностью стало появление объектно-ориентированного подхода и разработка соответствующих языков и компиляторов. Так C эволюционировал в C++.

Как мы знаем, язык программирования не существует в вакууме, сам по себе – для выполнения написанных на нём программ необходима ЭВМ и, особенно если это язык высокого уровня, работающая под управлением некоторой операционной системы (ОС).

Разработка ОС MS DOS началась в октябре 1980 года, когда компания IBM начала исследования рынка на предмет внедрения разрабатываемых ей персональных компьютеров. Впоследствии разработка системы велась совместно команиями Microsoft и IBM. MS-DOS стала основой для первых версий ОС Windows, которые по сути были лишь графической оболочкой, запускавшейся из командной строки. В настоящее время Windows является самой популярной операционной системой, имеющей развитый пользовательский интерфейс, поддерживающей многозадачность, огромное число периферийных устройств, а также работающей не только на персональных компьютерах, но и на множестве мобильных устройств.

Популярность этой ОС обусловлена не в последнюю очередь огромным количеством совместимых с ней популярных программных продуктов, таких как текстовые процессоры, программы обработки видео, звука и изображений, компьютерные игры, и многие другие. Такое изобилие стало возможным благодаря публикации с самых первых версия Windows интерфейса для разработчиков – библиотеки системных функций, с помощью которых можно получать доступ и управлять больинством программных и аппаратных ресурсов системы – окна, файлы, устройства и т.д. Эта библиотека функций называется Windows Application Programming Interface (WinAPI).

Данная работа посвящена изучению современных подходов к разработке программных систем на языке высокого уровня С++ с использованием в качестве среды разработки бесплатно распространяемой системы Visual Studio Community Edition.

Целью работы является закрепление и углубление знаний, полученных при изучении языка C++, а также развитие практических навыков при работе с функциями WinAPI и выборе представления исходных данных, тестировании, отладки программы и оформлении сопроводительной документации.

# Язык программирования C++

Предшественником языка C++ является императивный (процедурный) язык программирования C, который представляет собой следующий шаг эволюции после языка ассемблера и сохраняет достаточно широкие возможности по управлению памятью и процессом выполнения программы.

Язык C включает достаточно аскетичную стандартную библиотеку, упрощающую работу с памятью, строками, файлами и т.д. При разработке на языке C программисты вынуждены постоянно писать код для выделения и очистки памяти под массивы, списки и другие контейнеры. Для каждого контейнера отдельно приходится хранить количество элементов. Для обработки элементов пишутся бесчисленные циклы for и while. Для написания универсальных процедур приходится прибегать к указателям типа void\*, что требует особой внимательности при преобразовании типов. Более того, в реальном коде неизбежно встречаются многократные повторения кусков кода, выполняющих сортировку, поиск и многие другие типовые операции обработки последовательностей.

К счастью для большинства программистов, разработчики стандарта языка C++ – в частности, его главный идеолог и основатель Бьёрн Страуструп – собрали вокруг себя команду профессионалов, которые определили основной круг типовых задач, решаемых в повседневном программировании, а также разработали начальную реализацию стандартной библиотеки. В итоге, STL содержит средства для управления контейнерами – массивы, списки, стеки, очереди и т.д. – и значительное число типовых алгоритмов для работы с ними.

Язык C++ значительно расширяет возможности своего предшественника и включает в себя сразу три парадигмы – процедурное, объектно-ориентированное и обобщенное программирование. При этом соблюдается обратная совместимость с предшественником. По сути, современный язык C++ – это три диалекта, весьма существенно отличающиеся друг от друга.

В последние годы, с внедрением стандартов 2011, 2014 и 2017 годов, наметилась тенденция к появлению четвёртого диалекта. Речь идёт о так называемом функциональном программировании, в котором, в противовес императивной парадигме, процесс вычисления трактуется как последовательное вычисление значений функций в математическом понимании, а не смена состояний в результате череды процедурных вызовов.

На C и C++ написано огромное количество промышленного программного обеспечения – пожалуй, больше, чем на всех остальных. Можно для примера выделить несколько известных продуктов, имеющих успех на рынке программного обеспечения. Приведём так же возможные причины, по которым C++ кажется наилучшим выбором разработчиков этих программ.

**Google Chrome** – кроссплатформенный браузер, первый по скорости и производительности. Если кто-то захочет заменить язык программирования, на котором он написан, то придется использовать тот язык, который портируется на многие платформы. Речь идёт не только о Mac OSX, Windows или Linux, а также Android (в частности, Google TV — это Android с портированным Chrome).

**Adobe Creative Suite** – системы, работающие с видео, аудио и графикой очень требовательны к производительности. При необходимости портировать продукты Adobe пришлось бы использовать язык, при помощи которого можно создавать программы, использующие наименьшее количество ресурсов компьютера при выполнения простых задач создания/редактирования, и при этом оставляющие достаточно ресурсов для выполнения других задач системы. И, конечно же, необходимо, чтобы программа продолжала работать как на OS X, так и на Windows.

**Microsoft Office** – широко распространенная, расширяемая, встраиваемая и самая полнофункциональная на сегодняшний день офисная система. При портировании Microsoft Office нужно чтобы в результате не получился хаос и программа быстро реагировала на действия пользовател. Пользоватети и программисты должны быть уверены, что можно будет написать мощные расширения, которые смотрелись бы как одно целое с программой.

**World of Warcraft** – здесь имеет смысл рассматривать не только клиентское приложения для настольных ПК, которое перерабатывает огромное количество событий, следует множеству правил, отрисовывает графику и делает еще много всего связанного с игрой. Не стоит забывать также о серверной части, обслуживающей миллионы запросов от игроков по всему миру и предъявляющей таким образом жесточайшие требования не только к аппаратной платформе, но к производительности используемого программного обеспечения.

# Среда разработки Visual Studio

Microsoft Visual Studio Community Edition — [свободно](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) распространяемая [интегрированная среда разработки (англ. IDE, Integrated Development Environment) приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для семейства языков программирования, в том числе [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))/[C++](https://ru.wikipedia.org/wiki/C%2B%2B). Пакет установки доступен по адресу: https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16.

Среда разработки включает редактор кода с поддержкой технологии IntelliSense – механизма автоматического дополнения кода, в зависимости от структуры программы и текущего контекста. Также имеются встроенные редактор форм и веб-редактор, помогающие разработчику быстро создавать графический пользовательский интерфейс.

Microsoft Visual Studio предоставляет множество других удобных функций для редактирования исходного кода программы – подстветка ключевых слов, идетификаторов и комментариев, навигация по структуре программы, поиск по файлам проекта. Кроме того, имеется очень тесная связь с поддерживаемыми компиляторами – в нашем случае, Microsoft C++ Compiler – можно очень детально настраивать опции компиляции, используя при этом удобный диалоговый интерефейс. Кроме C++, поддержваются также языки Visual Basic, Visual C# и Visual F#. Комплект поставки, как правило, включает также в том или ином объёме СУБД MS SQL Server.

Следует также упомянуть об удобных средствах интерактивной отладки, встроенной в редактор кода и включающей множество полезных функций – например, наглядную навигацию по ходу выполнения программы, просмотр значений переменных или содержимого стека выполнения. Встроенный в интегрированную среду разработки отладчик может работать на уровне как исходного, так и машинного кода.

Кроме встроенных инструментов, поставляемых с установочным комплектом, Visual Studio посволяет создавать свои или подключать сторонние расширения, дополняющие общую функциональность интегрированной среды – например, модули взаимодействия с различными системами контроля версий.

# Библиотека «Windows Application Programming Interface (API)»

Как известно, любая программа должна получать из внешнего мира данные для обработки, а также выдавать во внешний мир результаты своей работы, т.е. взаимодействовать с пользователями посредством различных периферийных устройств – клавиатура, мышь, принтер, сеть, порты ввода-вывода и другие.

Текст самой программы пишется на каком-либо языке программирования, транслируется в машинный код и запускается на выполнение – таким образом, программист задаёт некотороую последовательность выполнения обработки данных.

Выполнение машинного кода на процессоре, а также работой периферийных устройств управляет операционная система – как правило, на уровне своего ядра, с помощью специального программного обеспечения, драйверов.

Если текст программы, отвечающий за выполнение обработки данных, с некотороми допущениями можно транслировать в код для разных операционных систем с помощью соответствующих компиляторов, то большинство операций по взаимодействию с периферийными устройствами и даже с локальной файловой системой глубоко специфичны для каждой операционной системы. Часть этой специфики убирается благодаря наличию CRT – библиотеки времени выполнения C/C++, которая икапсулирует некоторые аспекты, такие как, например, управление виртуальной памятью процесса или простейшие файловые операции. Однако для получения более полного контроля над имеющимися в распоряжении системными ресурсами требуется доступ к программному интерфейсу, предоставляемому конкретной операционной системой. В Windows таким интерфейсом является библиотека функций, носящая название Windows Application Programming Interface (WinAPI). Кроме вышеупомянутых ресурсов, этот интерфейс также предоставляет полное управление графическим оконным интерфейсом операционной системы – изображение окон и расположенных на них элементов управления, реакция на действия пользователя и многое другое.

Библиотека функций WinAPI представляет собой набор заголовочных файлов, содержащих сигнатуры всех доступных функций и определения структур данных, а также совокупность всех объектных библиотек, с которыми необходимо компоновать исполняемые файлы пользовательских приложений. Реализация этих функций находится в ядре самой операционной системы – в динамических библиотеках, например, kernel.dll. Для использования большинства функций WinAPI достаточно включить в текст программы заголовочный файл windows.h, а также скомпоновать исполняемый файл с объектными библиотеками kernel32.lib и user32.lib.

Компания Microsoft предоставляет можную информационно-справочную поддержку через сервис Microsoft Developer Network (MSDN), содежащий описание всех функций, а также методик и подходов программирования под Windows с использованием API. Раньше справочная система поставлялась на установочных дисках вместе с Visual Studio, но прогресс не стоит на месте, и постепенно на смену твёрдым носителям пришла Всемирная паутина. В данный момент самая актуальная информация по Windows для ползователей и разработчиков доступна по адресу <https://docs.microsoft.com/ru-ru/windows/>

# Используемые функции Windows API

**GetSystemInfo**

Получает текущую информацию о системе – архитектура и тип процессора, параметры виртуальной памяти, количество процессоров. Информация возвращается в виде заполненной структуры SYSTEM\_INFO.

Возвращаемое значение: нет (void).

Параметры:

lpSystemInfo (LPSYSTEM\_INFO) – указатель на область памяти со структурой SYSTEM\_INFO.

Поля структуры SYSTEM\_INFO:

* dwOemId (DWORD) – устаревшее поле, оставлено для совместимости; приложениям рекомендуется использовать поле wProcessorArchitecture;
* wProcessorArchitecture (WORD) – архитектура процессора, может принимать следующие значения:
  + PROCESSOR\_ARCHITECTURE\_AMD64 - x64 (AMD или Intel);
  + PROCESSOR\_ARCHITECTURE\_ARM - ARM
  + PROCESSOR\_ARCHITECTURE\_ARM64 - ARM64
  + PROCESSOR\_ARCHITECTURE\_IA64 – процессор на базе Intel Itanium
  + PROCESSOR\_ARCHITECTURE\_INTEL - x86
  + PROCESSOR\_ARCHITECTURE\_UNKNOWN – неизвестная архитектура
* wReserved (WORD) – поле для выравнивания структуры в памяти;
* dwPageSize (DWORD) – размер, а также гранулярность размещения страниц виртуальной памяти;
* lpMinimumApplicationAddress (LPVOID) – указатель на минимальный адрес памяти, доступный для использования приложениями и динамически загружаемыми библиотеками;
* lpMaximumApplicationAddress (LPVOID) – указатель на максимальный адрес памяти, доступный для использования приложениями и динамически загружаемыми библиотеками;
* dwActiveProcessorMask (DWORD\_PTR) – двоичная маска из 32 бит, описывающая процессоры, установленных в системе; нулевому биту соответствует первый процессор, а последнемуму – 32й процессор;
* dwNumberOfProcessors (DWORD) – количество процессоров (процессорных ядер) в системе;
* dwProcessorType (DWORD) – устаревшее поле, оставленное для обратной совместимости; рекомендуется вместо него использовать поля wProcessorArchitecture , wProcessorLevel и wProcessorRevision;
* dwAllocationGranularity (DWORD) – гранулярность начального адреса блоков виртуальной памяти, выделяемых функцией VirtualAlloc;
* wProcessorLevel (WORD) – зависящий от конкретной архитектуры «уровень» процессора; рекомендуется использовать это поле только для отображения информации пользователю, но не для автоматического анализа для принятия решения в программном коде;
* wProcessorRevision (WORD) – зависящая от конкретной архитектуры «ревизия» процессора, это поле может принимать следующие значения:
  + PROCESSOR\_INTEL\_386 (386)
  + PROCESSOR\_INTEL\_486 (486)
  + PROCESSOR\_INTEL\_PENTIUM (586)
  + PROCESSOR\_INTEL\_IA64 (2200)
  + PROCESSOR\_AMD\_X8664 (8664)
  + PROCESSOR\_ARM (Reserved)

**GetComputerName**

Получает имя компьютра. Система считывает это имя из реестра при запуске.

Возвращаемое значение: признак успешного завершения (BOOL). Если произошла ошибка, то функция вернёт нулевое значение, в случае успеха – значение, отличное от нуля.

Параметры:

lpBuffer (LPSTR) – указатель на выделенную область памяти, куда будет записана строка с именем компьютера. Рекомендуется выделить память, достаточную для размещения (MAX\_COMPUTERNAME\_LENGTH + 1) символов.

nSize (LPDWORD) – указатель на беззнаковое целое. До вызова функции это поле должно содержать размер выделенного буфера, после вызова – количество реально скопированных символов, за исключением завершающего признака конца строки. Если размер буфера недостаточно велик для того чтобы вместить имя компьютера, функция вернёт признак неудачного завершения. В этом случае функция GetLastError() должна возвращать код ошибки ERROR\_BUFFER\_OVERFLOW, а поле по адресу nSize будет содержать требуемый размер буфера.

**GetUserName**

Получает имя текущего пользователя

Возвращаемое значение: признак успешного завершения (BOOL). Если произошла ошибка, то функция вернёт нулевое значение, в случае успеха – значение, отличное от ноля.

Параметры:

lpBuffer (LPSTR) – указатель на выделенную область памяти, куда будет записана строка с именем текущего пользователя. Если памяти недостаточно для того чтобы вместить имя пользователя, функция вернёт признак неудачного завершения, а код ошибки, возвращаемый функцией GetLastError, будет равен ERROR\_INSUFFICIENT\_BUFFER. Максимально возможное имя пользователя определяется определённой в файле Lmcons.h константой UNLEN, поэтому для того чтобы точно разместить имя пользователя в буфере, достаточно выделить в памяти буфер размером UNLEN + 1.

pcbBuffer (LPDWORD) – указатель на беззнаковое целое. До вызова функции это поле должно содержать размер выделенного буфера, после вызова – количество реально скопированных символов, включая завершающий признак конца строки.

**GetSystemDirectory**

Получает путь к каталогу с установленными системными файлами Windows – динамически подключаемыми библиотеками и драйверами устройств.

Возвращаемое значение: количество символов (UINT), скопированных в выделенный буфер памяти. Если произошла ошибка, это значение будет нулевым.

Параметры:

lpBuffer (LPSTR) – указатель на выделенную область памяти, куда будет записана строка, содержащая полный путь к системному каталогу. Этот путь не включает завершающий символ «\». Например, если системные файлы установлены в каталоге Windows\System32 на диске C, то память по этому указателю будет содержать ASCII-строку «C:\Windows\System32».

uSize (UINT) –размер выделенного буфера.

MSDN не рекомендует приложениям и библиотекам создавать файлы в этом каталоге. Кроме того, в большинстве случаем пользовательская учётная запись не будет иметь права на запись в системной директории.

**GetWindowsDirectory**

Получает путь к каталогу с установленной на компьютере операционной системой.

Возвращаемое значение: количество символов (UINT), скопированных в выделенный буфер памяти, не считая завершающего признака конца строки. Если произошла ошибка, это значение будет нулевым.

Параметры:

lpBuffer (LPSTR) – указатель на выделенную область памяти, куда будет записана строка, содержащая полный путь к каталогу с операционной системой. Этот путь не включает завершающий символ «\». Например, если операционная система установлена в стандартный каталог Windows на диске C, то память по этому указателю будет содержать ASCII-строку «C:\Windows».

uSize (UINT) –размер выделенного буфера, адрес которого передаётся в параметре lpBuffer.

Раньше некоторые старые приложения хранили данные в этом каталоге – программ было достаточно немного и этот подход считался общепринятой практикой. Теперь же принято хранить данные приложений либо в системном реестре, либо в конфигурационных файлах, расположенных там же, куда установлена сама программа. Так же, как и для системного каталога, для каталога Windows пользовательским учётным записям запрещена запись в него.

Программы, которые предпочитают хранить какие-то данные для конкретных пользователей, могут использовать переменную окружения HOMEPATH, содержащую строку с домашним каталогом текущего пользователя.

# Тестирование программы

Исходный текст программы приведён в приложении А. Запускать программу на выполнение можно как из среды разработки – например, нажатием клавиши F5, так и из командной строки, по имени исполняемого файла. Получаемый в результате компиляции программы файл находится в каталоге, соответствующем текущей выбранной конфигурации компилятора и целевой платформы. Например, файл, собранный для 64-битной платформы в отладочной конфигурации будет расположен в каталоге SysInfo\x64\Debug.

После запуска программы открывается окно с системной информацией, изображённое на рисунке 1. Тестовый запуск произодился на 64-разрядной системе Windows 7, на ноутбуке с 4-ядерным процессором Intel Core i5-2410M.

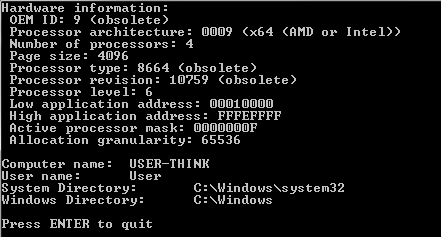


Рисунок . Отчёт на ноутбуке

На каждой системе информация будет отличаться в соответствии с её особенностями – количество ядер процессора, имя компьютера или пользователя, пути к каталогам и т.д.

Ещё один тестовый запуск, результаты которого изображены на рисунке 2, произведён на 64-разрядной системе Windows 7, на стационарном ПК с 4-ядерным процессором Intel Core 2 Quad.

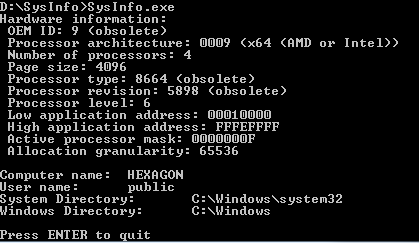


Рисунок . Отчёт на стационарном ПК

Как видно из рисунков, устаревшее поле wProcessorRevision даже для более старого процессора Intel Core 2 Quad содержит не определённое для него в офциальной документации значение 5898, не говоря уже о значении, выдаваемом системой на более современном процессоре ноутбука.

Также из рисунков видно, что операционная система на обоих компьютерах вместе со своими системными файлами установлена, как и на большинстве пользовательских ПК, в стандартные каталоги на диске C.

# ЗАКЛЮЧЕНИЕ

В результате работы создана программа, демонстрирующая работу на языке C++ с функциями библиотеки Windows API в среде Microsoft Visual Studio. Изучено подмножество функций WinAPI, отвечающее за предоставление системной информации.

Приобретены практические навыки разработки приложений, использования языковых конструкций С++, отладки программ и составления сопроводительной документации. На базе практического применения, закреплены базовые знания, полученные ранее, при изучении теории.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Герберт Шилдт Искусство программирования на C++ / Герберт Шилдт. - М.: БХВ-Петербург, 2005. - 496 c.

2. Герб Саттер Решение сложных задач на С++ / Герб Саттер. - М.: Вильямс, 2015. - 400 c.

3. Александр Чиртик Программирование на C++. Трюки и эффекты / Александр Чиртик. - М.: "Издательство "Питер", 2010. - 352 c.

4. Давыдов, В. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций / В. Давыдов. - М.: БХВ-Петербург, 2014. - 576 c.

5. Юрий Щупак. Win32 API. Эффективная разработка приложений / Ю. Щупак. - М.: Питер, 206. - 576 c.

6. ГОСТ 2.105-95 Общие требования к текстовым документам.

# ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

**SysInfo.cpp**

#include "pch.h"

#include <windows.h>

#include <tchar.h>

const int BufferSize = 4096;

TCHAR infoBuf[BufferSize];

int main()

{

SYSTEM\_INFO sysInfo;

DWORD bufCharCount = BufferSize;

// Отобразить общую аппаратную информацию

GetSystemInfo(&sysInfo);

\_tprintf(TEXT("Hardware information: \n"));

\_tprintf(TEXT(" OEM ID: %u (obsolete)\n"), sysInfo.dwOemId);

\_tprintf(TEXT(" Processor architecture: %04X ("), sysInfo.wProcessorArchitecture);

switch (sysInfo.wProcessorArchitecture) {

case PROCESSOR\_ARCHITECTURE\_AMD64:

\_tprintf(TEXT("x64 (AMD or Intel)"));

break;

case PROCESSOR\_ARCHITECTURE\_ARM:

\_tprintf(TEXT("ARM"));

break;

case PROCESSOR\_ARCHITECTURE\_ARM64:

\_tprintf(TEXT("ARM64"));

break;

case PROCESSOR\_ARCHITECTURE\_IA64:

\_tprintf(TEXT("Intel Itanium"));

break;

case PROCESSOR\_ARCHITECTURE\_INTEL:

\_tprintf(TEXT("x86"));

break;

case PROCESSOR\_ARCHITECTURE\_UNKNOWN:

\_tprintf(TEXT("Unknown"));

break;

default:

\_tprintf(TEXT("Unsupported value"));

}

\_tprintf(TEXT(")\n"));

\_tprintf(TEXT(" Number of processors: %u\n"), sysInfo.dwNumberOfProcessors);

\_tprintf(TEXT(" Page size: %u\n"), sysInfo.dwPageSize);

\_tprintf(TEXT(" Processor type: %u (obsolete)\n"), sysInfo.dwProcessorType);

\_tprintf(TEXT(" Processor revision: %u (obsolete)\n"), sysInfo.wProcessorRevision);

\_tprintf(TEXT(" Processor level: %u\n"), sysInfo.wProcessorLevel);

\_tprintf(TEXT(" Low application address: %08X\n"), sysInfo.lpMinimumApplicationAddress);

\_tprintf(TEXT(" High application address: %08X\n"), sysInfo.lpMaximumApplicationAddress);

\_tprintf(TEXT(" Active processor mask: %08X\n"), sysInfo.dwActiveProcessorMask);

\_tprintf(TEXT(" Allocation granularity: %u\n"), sysInfo.dwAllocationGranularity);

// Отобразить имя компьютера

if (!GetComputerName(infoBuf, &bufCharCount)) {

\_tprintf(TEXT("\nGetComputerName failed"));

return -1;

}

\_tprintf(TEXT("\nComputer name:\t%s"), infoBuf);

// Отобразить имя пользователя

bufCharCount = BufferSize;

if (!GetUserName(infoBuf, &bufCharCount)) {

\_tprintf(TEXT("\GetUserName failed"));

return -1;

}

\_tprintf(TEXT("\nUser name:\t%s"), infoBuf);

// Отобразить системный каталог

if (!GetSystemDirectory(infoBuf, BufferSize)) {

\_tprintf(TEXT("GetSystemDirectory"));

return -1;

}

\_tprintf(TEXT("\nSystem Directory:\t%s"), infoBuf);

// Отобразить каталог операционной системы

if (!GetWindowsDirectory(infoBuf, BufferSize)) {

\_tprintf(TEXT("GetWindowsDirectory"));

return -1;

}

\_tprintf(TEXT("\nWindows Directory:\t%s"), infoBuf);

\_tprintf(TEXT("\n\nPress ENTER to quit "));

\_fgettc(stdin);

}