Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем (АВС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

«Сбор сведений о ресурсах системы и управление ими»

Выполнила: студентка группы 353501,

Климчук А. А.

Руководитель: ассистент кафедры информатики, Плехова Т. В.

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc437688583)

[1 Анализ предметной области……………………………………………………………...4](#_Toc437688584)

[1.1 Процессы 4](#_Toc437688585)

1.2 Потоки………………………………………………………………………………….4

[1.3 Реестр 5](#_Toc437688586)

1.4 Обзор аналогов………………………………………………………………………...6

1.5 Постановка задачи……………………………………………………………………..7

[2 Используемые технологии 8](#_Toc437688587)

[2.1 C#................ 8](#_Toc437688588)

2.2 WPF……………………………………………………………………………………..8

[2.3 Visual Studio 2015……………………………………………………………………...9](#_Toc437688589)

[3 Программная реализация 11](#_Toc437688590)

[3.1 Возможности платформы .NET 11](#_Toc437688591)

3.2 Windows Management Instrumentation……………………………………………....12

3.3 Log Parser……………………………………………………………………………...13

3.4 Lazy Initialize………………………………………………………………………….13

3.5 Double Checked Locking……………………………………………………………...14

[4 Описание применения 15](#_Toc437688592)

[4.1 Главное окно системы 15](#_Toc437688593)

4.2 Окна с информацией о системе……………………………………………………...15

Заключение…………………………………………………………………………………20

[Список использованных источников 21](#_Toc437688594)

### [Приложение А Класс получения информации о текущем процессе (использование](#_Toc437688595)

### [возможностей платформы .NET)……………………………………….22](#_Toc437688595)

Приложение Б Класс получения информации о процессоре (использование WMI)….22

Приложение В Класс для получения актуальной информации о загрузке процессора и

состоянии RAM (Double checked locking)………………………………24

Приложение Г Метод поиска в реестре с использованием классов Registry и

RegistryKey………………………………………………………………..24

Приложение Д Метод поиска в реестре с использованием Log Parser………………...25

Приложение Е Класс главного окна MainWindow.sc……………………………………26

# ВВЕДЕНИЕ

Операционная система (ОС) представляет собой комплекс взаимосвязанных программ, являющихся интерфейсом между приложениями пользователя и аппаратурой компьютера.

С одной стороны, операционную систему можно рассматривать как виртуальную машину. Виртуальная машина управляется командами более высокого уровня, чем реальная. Для решения своих задач пользователь может обойтись без знания аппаратного устройства компьютера. Например, при работе с диском достаточно представлять его в виде некоторого набора файлов с именами. Последовательность действий при работе с файлами заключается в его открытии, выполнении одной или нескольких операций чтения/записи и в его закрытии.

С другой стороны, операционную систему можно рассматривать как систему управления ресурсами. К числу основных ресурсов относятся процессор, память, диски, сетевые устройства и т.д. Управление ресурсами включает решение следующих общих задач:

* планирование ресурса, т.е. определение, какой задаче и в каком количестве следует выделить данный ресурс;
* удовлетворение запросов на ресурсы;
* разрешение конфликтов между задачами.

Для эффективного управления ресурсами необходимо иметь возможность своевременно получать необходимую информацию о системе. Например, свободной оперативной памяти, дисковом пространстве, загрузке процессора и т.д. Таким образом, *цель* данной курсовой работы: создать программное обеспечение по сбору информации о ресурсах системы и предоставить определенные возможности по управлению ими.

## 1 Анализ предметной области

## Процессы

Программа – статический объект, представляющий собой файл или совокупность файлов с кодами и данными. Для того чтобы программа могла быть запущена на выполнение, операционная система должна создать окружение или среду выполнения задачи, включающую возможности доступа к различным системным ресурсам (память, устройства ввода-вывода, файлы и т.д.). Такое окружение получило название процесса. В целом, процесс можно представить как совокупность данных ядра системы, необходимых для описания образа программы в памяти и управления ее выполнением.

Процессы можно условно разбить на три категории:

* системные;
* фоновые;
* прикладные (пользовательские).

Системные процессы являются частью ядра операционной системы и всегда расположены в оперативной памяти.

Фоновые процессы – это не интерактивные процессы, которые обычно запускаются при инициализации системы и обеспечивают работу различных подсистем. Такие процессы не связаны с пользовательскими сеансами работы и не могут непосредственно управляться пользователями.

Прикладные процессы, как правило, порождаются в рамках пользовательского сеанса. Они могут выполняться как в интерактивном, так и в фоновом режимах.

Управление процессами осуществляется операционной системой в режиме ядра и включает следующие функции:

* создание и завершение процессов;
* планирование и диспетчеризация процессов;
* переключение процессов;
* синхронизация и обмен информацией между процессами;
* организация управляющих блоков процессов.
  1. **Потоки**

Поток является последовательностью команд, обрабатываемых процессором. В рамках одного процесса могут находиться один или несколько потоков. Традиционный подход, при котором каждый процесс представляет собой единый поток выполнения, называется однопоточным. Например, MS-DOS поддерживает один однопоточный пользовательский процесс. Некоторые ОС семейства UNIX поддерживают процессы множества пользователей, но при этом каждый из процессов содержит один поток. Многопоточностью называется способность ОС поддерживать в рамках одного процесса выполнение нескольких потоков. Примерами многопоточных систем являются среда выполнения Java, Windows, Linux.

В многопоточной среде процесс можно рассматривать как структурную единицу объединения ресурсов и структурную единицу защиты. С другой стороны, процесс можно рассматривать как общий поток исполняемых команд, состоящий из нескольких отдельных потоков. У каждого потока есть свой счетчик команд, регистры и стек. Таким образом, в многопоточной среде процессы используются для группирования ресурсов, а потоки являются объектами, поочередно выполняющимися на процессоре, создавая впечатление параллельной работы потоков. Все потоки одного процесса разделяют между собой ресурсы процесса. Они находятся в общем адресном пространстве и имеют доступ к одним и тем же данным.

Использование потоков имеет следующие преимущества с точки зрения производительности.

* Создание нового потока в уже существующем процессе занимает существенно меньше времени, чем создание нового процесса.
* Поток можно завершить быстрее, чем процесс.
* Переключение потоков в рамках одного процесса происходит намного быстрее.
* Использование потока повышает эффективность обмена информацией между процессами. В большинстве операционных систем обмен информацией между процессами происходит с участием ядра, обеспечивающего механизм осуществления обмена и защиту.

## 1.3 Реестр

Реестр (Registry) — специальная системная база данных, в которой приложения и операционная система могут сохранять информацию о конфигурации.

Системный реестр служит для осуществления ряда функций:

* хранение конфигураций оборудования и об устройствах «Plug-and-Play»;
* хранение списка драйверов и их параметров;
* описания программных интерфейсов (например, интерфейсы COM-серверов);
* таблица ассоциаций файлов данных;
* хранение конфигурации и значений параметров программ;
* обслуживание различных административных программ, например, панель управления (Control Panel).

Реестр имеет иерархическую древовидную структуру. Узлы дерева называются ключами (key). Каждый ключ может содержать любое количество подключей (sub-key) и значений (values), причем и те, и другие организованы в виде неупорядоченных списков, элементам которых присвоены индексы начиная с нуля. Различие между ключами и подключами в действительности условно, фактически все ключи являются подключами различного уровня нескольких предопределенных ключе. Значения ключей могут быть строковыми, двоичными и числовыми. Tакже ключ может быть ссылкой на другой ключ. Каждый ключ идентифицируется его именем, уникальным относительно вышестоящего ключа, открытые ключи идентифицируются описателем (номером), тип HKEY (Handle). Значения идентифицируются также именами и индексами в списке.

Данные в системном реестре хранятся в двоичном виде. Для работы с ними приложения должны использовать специальные системные функции. Различают hive (букв. «улей») ключей, то есть их двоичный образ в структурах в памяти, называемых собственно реестром, и файлы данных реестра.

Для работы с ключом приложение должно открыть его. При открытии ключа необходимо указать открытый ранее ключ в качестве вышестоящего. Система всегда предоставляет пять предопределенных ключа верхнего уровня, которые считаются открытыми всегда и могут использоваться как точки входа в реестр:

* HKEY\_LOCAL\_MACHINE — описание известных на этот момент конфигураций компьютера;
* HKEY\_CLASSES\_ROOT — описание текущей конфигурации машины;
* HKEY\_USERS — описание всех имеющихся пользо­вателей;
* HKEY\_CURRENT\_USER — описание текущего пользователя, ссылка на одного из пользователей HKEY\_USERS;
* HKEY\_CURRENT\_CONFIG — текущий подключ Config, ссылка ключа HKEY\_LOCAL\_MACHINE.

На пользовательском уровне для работы с реестром служит утилита regedit.

* 1. **Обзор аналогов**

В настоящее время в операционных системах семейства Microsoft Windows используется диспетчер задач – стандартная утилита. С ее помощью можно в режиме реального времени отслеживать выполняющиеся приложения и запущенные приложения, оценивать загруженность системных ресурсов компьютера и использования сети.

Диспетчер задач содержит следующие вкладки.

* Приложения. Позволяет переключиться в нужное приложение либо завершить его.
* Процессы. Разнообразные данные обо всех запущенных в системе процессах можно завершать, менять приоритет, задавать соответствие процессорам.
* Службы. Сведения обо всех службах Windows.
* Быстродействие. Графики загрузки процессора, использование оперативной памяти.
* Сеть. Графики загрузки сетевых подключений.
* Пользователи. Манипулирование активными пользователями.

Для каждого процесса выводятся его параметры: имя образа, PID, имя пользователя, загрузка эти процессом центрального процессора и объем занимаемой им оперативной памяти. В ряде случаев можно завершить выполняющийся процесс.

Основными недостатками диспетчера задач Windows являются

* невозможность просмотреть иерархию запущенных процессов, их связь между собой;
* нет информации о процессах;
* нет информации об открытых дескрипторах и загруженных библиотеках DLL;
* нельзя завершить некоторые зависшие процессы, особенно системные.

Как альтернатива диспетчеру задач Windows рекомендуется Process Explorer. Process Explorer предоставляет следующие возможности:

* показывает всю необходимую информацию о всех запущенных процессах и используемых ресурсам по каждому из них;
* располагает информацией о том, какие DLL используются каждым процессом и с какими ключами они запускались;
* имеет хорошую систему поиска по всей доступной информации;
* есть поиск в интернете по выбранным подозрительным процессам на предмет вирусности;
* показывает подробную информацию по использованию ресурсов системы в целом (оперативная память, процессор, диски, видеокарта);
* закрывает «непослушные» процессы.
  1. **Постановка задачи**

Приведенные аналоги для сбора информации о ресурсах системы узконаправленны. Основное их предназначение – это предоставить пользователю информацию о запущенных процессах, по возможности всех, показать подробную информацию по использованию ресурсов системы и т.д.

В данной курсовой работе будет реализовано приложение по сбору полной информации о системе Windows, включая видеокарту, процессор, операционную систему, установленные ПО и т.д.

Основными задачами программного обеспечения являются:

* получение списка запущенных процессов;
* получение полной информации по каждому отдельному процессу и предоставление возможности управления ими;
* получение системной информации о видеокарте, операционной системе, процессоре, установленном программном обеспечении, оперативной памяти, сервисах системы, дисковом пространстве;
* предоставлении актуальной информации о загрузке процессора и доступной RAM в каждый момент времени;
* предоставление возможности работы с реестром.

Следовательно, разрабатываемый программный продукт отличается от уже имеющихся аналогов своей многофункциональностью и должен отвечать предоставленным минимальным требованиям.

# Используемые технологии

Приложение «Сбор информации о ресурсах системы и управление ими» реализовано на языке C# в среде программирования Visual Studio 2015 с использованием технологии Windows Presentation Foundation (WPF).

### C#

98% преимуществ C# относятся к преимуществам .NET, а не к преимуществам самого языка, к которым относятся сборщик мусора, нормальная поддержка исключений, обширная стандартная кодобаза, атрибуты и отражение, делегаты (хотя, на самом деле, можно было обойтись и без них) и события, промежуточный код, декларативные механизмы обеспечения безопасности выполнения кода, межъязыковое взаимодействие, свойства, индексы, достаточно полная и непротиворечивая система типов и т.д.

C# разрабатывался как язык программирования прикладного уровня для CLR, и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

Среди достоинств самого C# можно выделить анонимные делегаты и лямбда-выражения, продолжения (continuations), встроенная в язык реализация некторых популярных примитивных паттернов проектирования (например, итераторов), б'ольшая степень выразительности, синтаксический сахар для типичных задач (в том числе linq), вывод типов из инициализации, нормальная поддержка модульности, частично определяемые типы и ещё много чего по мелочи. Приятно радует сведение к минимуму неявных преобразований типов и перегрузок унарных операторов.   
 Есть у C# и недостатки по отношению к C++, но в целом применение C# почти всегда гораздо дешевле при большем спектре решаемых с его помощью задач, нежели применение C++.

* 1. **WPF**

Windows Presentation Foundation (WPF) – это технология для построения клиентских приложений Windows, являющаяся частью платформы .NET. WPF разработана как альтернатива технологии Windows Forms. Основные особенности технологии WPF:

1. Собственные методы построения и рендеринга элементов.В Windows Forms классы для элементов управления делегируют функции отображения системным библиотекам, таким как user32.dll. В WPF любой элемент управления полностью строится самой WPF.

2. Независимость от разрешения устройства вывода.Для указания размеров в WPF используется собственная единица измерения, равная 1/96 дюйма. Кроме этого, технология WPF ориентирована на использование не пиксельных, а векторных примитивов.

3. Декларативный пользовательский интерфейс.В WPF визуальное содержимое окна можно полностью описать на языке XAML. Это язык разметки, основанный на XML. Применение XAML является предпочтительным, но не обязательным – приложения WPF можно создавать, используя только код.

4. Веб-подобная модель компоновки.WPF поддерживает гибкий визуальный поток, размещающий элементы управления на основе их содержимого. В результате получается пользовательский интерфейс, который может быть адаптирован для отображения динамичного содержимого.

5. Анимация.В WPF анимация – неотъемлемая часть программного каркаса. Анимация определяется декларативными дескрипторами, и WPF запускает её в действие автоматически.

6. Приложения на основе страниц.В WPF можно строить приложения с кнопками навигации, которые позволяют перемещаться по коллекции страниц. Кроме этого, специальный тип приложения WPF – XBAP – может быть запущен внутри браузера.

7. Стили и шаблоны.Стили стандартизируют форматирование и позволяют повторно использовать его по всему приложению. Шаблоны дают возможность изменить способ отображения любых элементов управления, даже таких основополагающих, как кнопки или поля ввода.

### Visual Studio 2015

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

В Visual Studio 2015 добавлены новые возможности:

* Single Sign In;
* новые возможности ASP.NET;
* Xamarin Integration Improvements;
* Visual Studio Tools для Apache Cordova;
* CodeLens;
* архитектурные инструменты;
* NuGet;
* XAML UI Debugging;
* Visual Studio Emulator for Android.

# Программная реализация

В данном разделе будет рассмотрена архитектура приложения, а также будут приведены примеры некоторых пользовательских функций и хранимых процедур.

## Возможности платформы .NET

При реализации системы по сбору информации о процессах/потоках были использованы возможности платформы .NET.

Статический класс Process из пространства имен System.Diagnostic предоставляет доступ к локальным и удаленным процессам и позволяет запускать и останавливать локальные системные процессы. В работе использовались методы GetProcesses(), GetProcessById() для получения процесса, Kill() – для завершения работы процесса. Экземплярные свойства ProseccName, Handle, SessionId, WorkingSet и т.д. предоставляют возможность получить полную информацию о процессе.

Класс DriveInfo из пространства имен System.IO предоставляет доступ к сведениям на диске. Для получения коллекции всех дисков используется статический метод GetDrives(). Для получения подробной информации о каждом диске используются свойства DriveType, RootDirectory, TotalSize, DriveFormat, TotalFreeSpace.

В обширной библиотеке .NET поставляются классы Registry и RegistryKey из пространства имен Microsoft Win32, которые позволяют получать доступ к реестру. Registry предоставляет собой класс, который позволяет получить эксклюзивный доступ к ключам реестра для выполнения простых операций. Другим предназначением класса Registry является предоставление экземпляров RegistryKey, представляющих ключи наивысшего уровня, т.е. разделы, которые позволяют осуществлять навигацию по реестру. Предоставляются эти экземпляры через семь статических свойств со следующими именами: ClassesRoot, CurrentConfig, CurrentUser, DynData, LocalMachine, PerformanceData и Users.

Экземпляр RegistryKey представляет ключ реестра. В этом классе есть методы для просмотра дочерних ключей, для создания новых ключей, а также для чтения и изменения значений в существующих ключах, т.е. можно выполнять все, что обычно требуется делать с ключами реестра, в том числе устанавливать для них уровни безопасности.

Для получения доступа к каждому сервису в системе используется запрос вида Registry.LocalMachine.OpenSubKey(

"SYSTEM\\CurrentControlSet\\services\\" + serv.ServiceName), гду serv – список всех сервисов. Для получения информации о сервисе используются его свойства DisplayName, Status, а также метод GetValue() для чтения соответствующей информации у ключа реестра.

Для предоставления информации об установленном в системе программном обеспечении также используются возможности платформы .NET для работы с реестром. Для этого происходит открытие ключа по пути "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall".

В системе по сбору информации предоставлена возможность работы с реестром: поиск в реестре по регулярному выражению, чтение значения по ключу, запись значения по ключу, удаление ключа и поддерева ключей, подсчет количества подключей у текущего ключа и значений. Это реализовано с использованием классов Registry и RegistryKey.

* 1. **Windows Management Instrumentation**

Windows Management Instrumentation (WMI) - это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows.

Технология WMI — это расширенная и адаптированная под Windows реализация стандарта WBEM, принятого многими компаниями, в основе которого лежит идея создания универсального интерфейса мониторинга и управления различными системами и компонентами распределенной информационной среды предприятия с использованием объектно-ориентированных идеологий и протоколов HTML и XML.

В основе структуры данных в WBEM лежит Common Information Model (CIM), реализующая объектно-ориентированный подход к представлению компонентов системы. CIM является расширяемой моделью, что позволяет программам, системам и драйверам добавлять в неё свои классы, объекты, методы и свойства.

WMI, основанный на CIM, также является открытой унифицированной системой интерфейсов доступа к любым параметрам операционной системы, устройствам и приложениям, которые функционируют в ней.

Важной особенностью WMI является то, что хранящиеся в нём объекты соответствуют динамическим ресурсам, то есть параметры этих ресурсов постоянно меняются, поэтому параметры таких объектов не хранятся постоянно, а создаются по запросу потребителя данных. Хранилище свойств объектов WMI называется репозиторием и расположено в системной папке операционной системы Windows: %SystemRoot%\System32\WBEM\Repository.

Так как WMI построен по объектно-ориентированному принципу, то все данные операционной системы представлены в виде объектов и их свойств и методов.

Все классы группируются в пространства имен, которые иерархически упорядочены и логически связаны друг с другом по определенной технологии или области управления. В WMI имеется одно корневое пространство имен Root, которое в свою очередь имеет 4 подпространства: CIMv2, Default, Security и WMI.

Классы имеют свойства и методы и находятся в иерархической зависимости друг от друга, то есть классы-потомки могут наследовать или переопределять свойства классов-родителей, а также добавлять свои свойства.

Свойства классов используются для однозначной идентификации экземпляра класса и для описания состояния используемого ресурса. Обычно все свойства классов доступны только для чтения, хотя некоторые из них можно модифицировать определенным методом. Методы классов позволяют выполнить действия над управляемым ресурсом.

Для обращения к объектам WMI используется специфический язык запросов WMI Query Language (WQL), который является одной из разновидностей SQL. Основное его отличие от ANSI SQL — это невозможность изменения данных, то есть с помощью WQL возможна лишь выборка данных с помощью команды SELECT. Помимо ограничений на работу с объектами, WQL не поддерживает такие операторы как DISTINCT, JOIN, ORDER, GROUP, математические функции. Конструкции IS и NOT IS применяются только в сочетании с константой NULL.

В работе для получения информации о ресурсах системы использовались следующие WMI queries:

* жесткий диск

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_DiskDrive WHERE Model = '" + model + "'");

* операционная системы

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_OperatingSystem");

* процессор

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_Processor");

* оперативная память

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_PhysicalMemory");

* видеокарта

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_VideoController").

* 1. **Log Parser**

Log Parser мощный, универсальный инструмент, который предоставляет универсальный доступ в виде запросов к лог, XML и CSV файлам, а также к ключевым источникам данным операционной системы Windows таким как журнал событий, реестр, файловая система и т.д.

С использованием возможностей Log Parser в программе реализована поиск в реестре по значению (см. приложение).

* 1. **Lazy initialize**

Отложенная инициализацияобъекта означает, что его создание откладывается до первого использования. Отложенная инициализация в основном используется, чтобы повысить быстродействие, избежать ресурсоемких вычислений и уменьшить требования к памяти программы.

В языке C# поддержку отложенной инициализации обеспечивает класс Lazy<T>. После создания объекта Lazy, экземпляр **типа** не создается до первого доступа к свойству Value переменной Lazy. При первом доступе заключенный в оболочку тип создается, возвращается и сохраняется для любого использования в будущем.

В проекте возможности отложенной инициализации использованы для получения списка всех сервисов в системе.

private Lazy<ServiceController[]> services = new Lazy<ServiceController[]>(() => { return ServiceController.GetServices(); }).

* 1. **Double Checked Locking**

Блокировка с двойной проверкой (Double checked locking) — параллельный шаблон проектирования, предназначающийся для уменьшения накладных расходов, связанных с получением блокировки. Сначала проверяется условие блокировки без какой-либо синхронизации; поток делает попытку получить блокировку, только если результат проверки говорит о том, что ни один другой поток не владеет блокировкой.

В проекте данный паттерн используется при обновлении информации р загрузке процессора каждую секунду (см. приложение).

# Описание применения

Приложение содержит ряд окон для предоставления возможности предоставления информации о системе. Рассмотрим работу с каждым из них.

## Главное окно системы

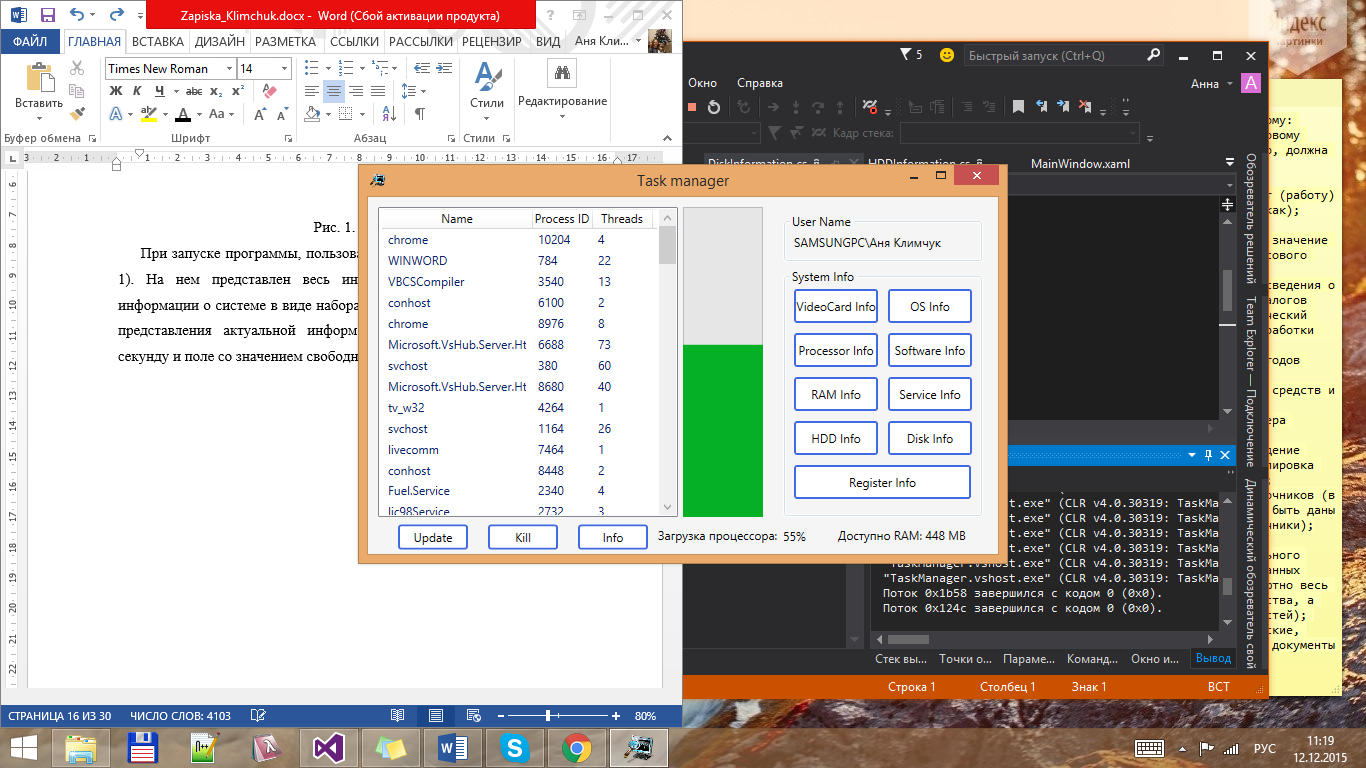


Рис. 1. Главное окно

При запуске программы, пользователя представляется главное окно (рис. 1). На нем представлен весь интерфейс для дальнейшего получения информации о системе в виде набора кнопок. Также имеется ProgressBar для представления актуальной информации о загрузке процессора каждую секунду и поле со значением свободной оперативной памяти.

* 1. **Окна с информацией о системе**

Для обработки запросов пользователя в виде нажатия на кнопки для получения информации о соответствующем ресурсе представлен ряд окон.

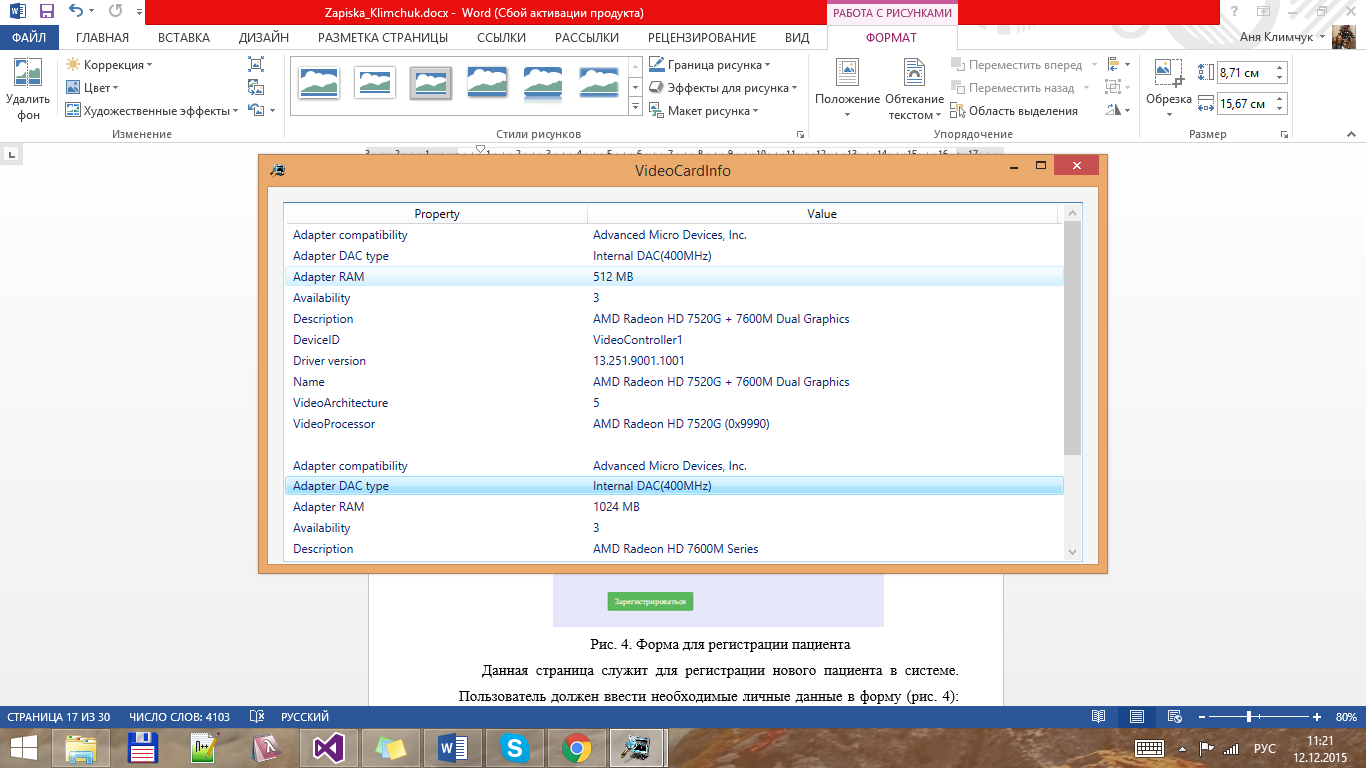


Рис. 3. Окно с информацией о видеокарте

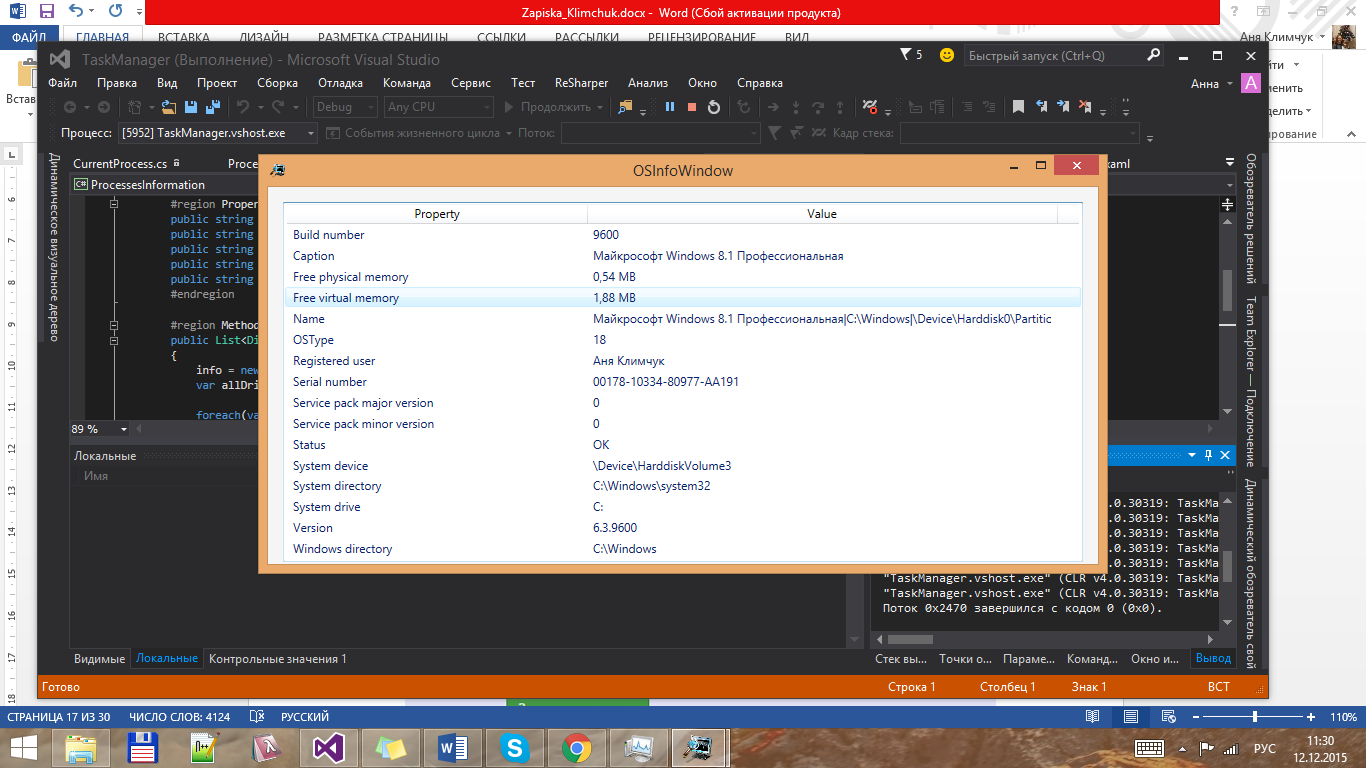


Рис. 4. Окно с информацией об операционной системе

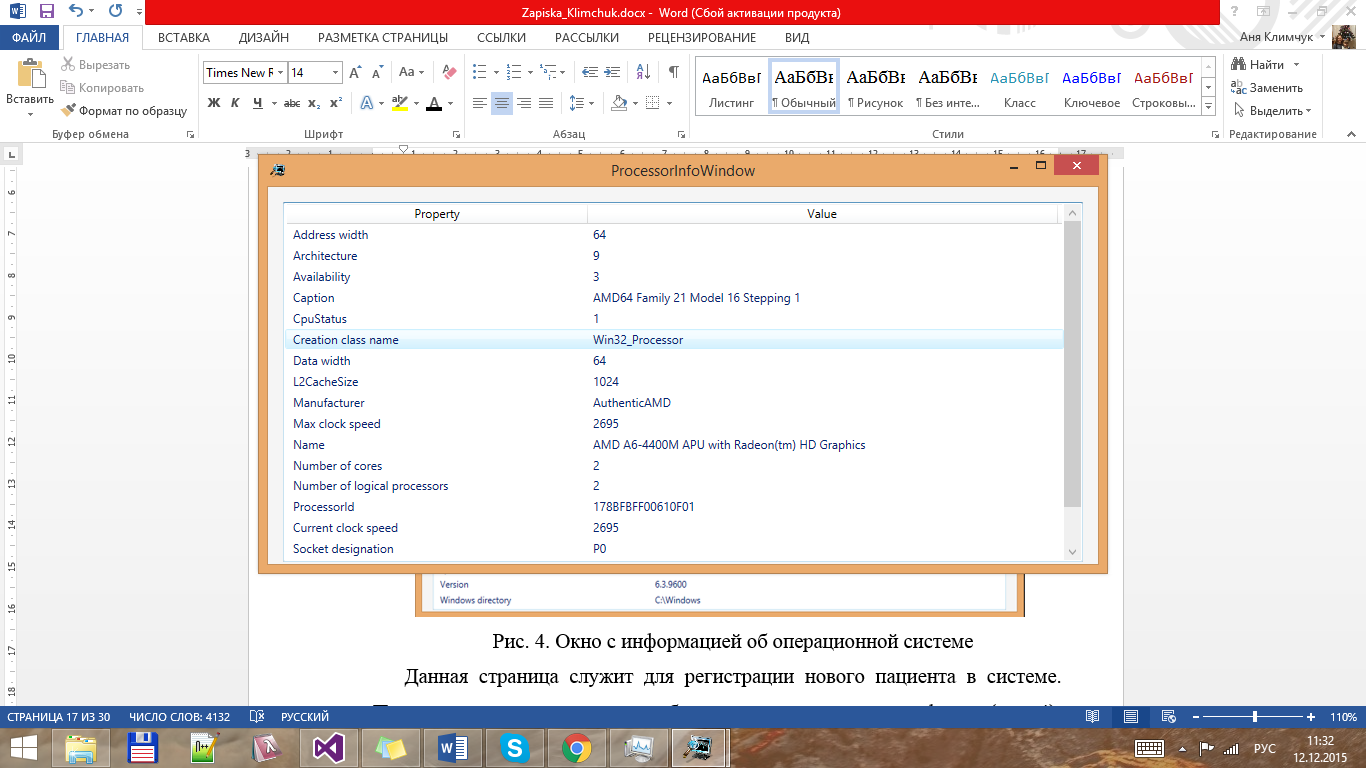


Рис. 5. Окно с информацией о процессоре

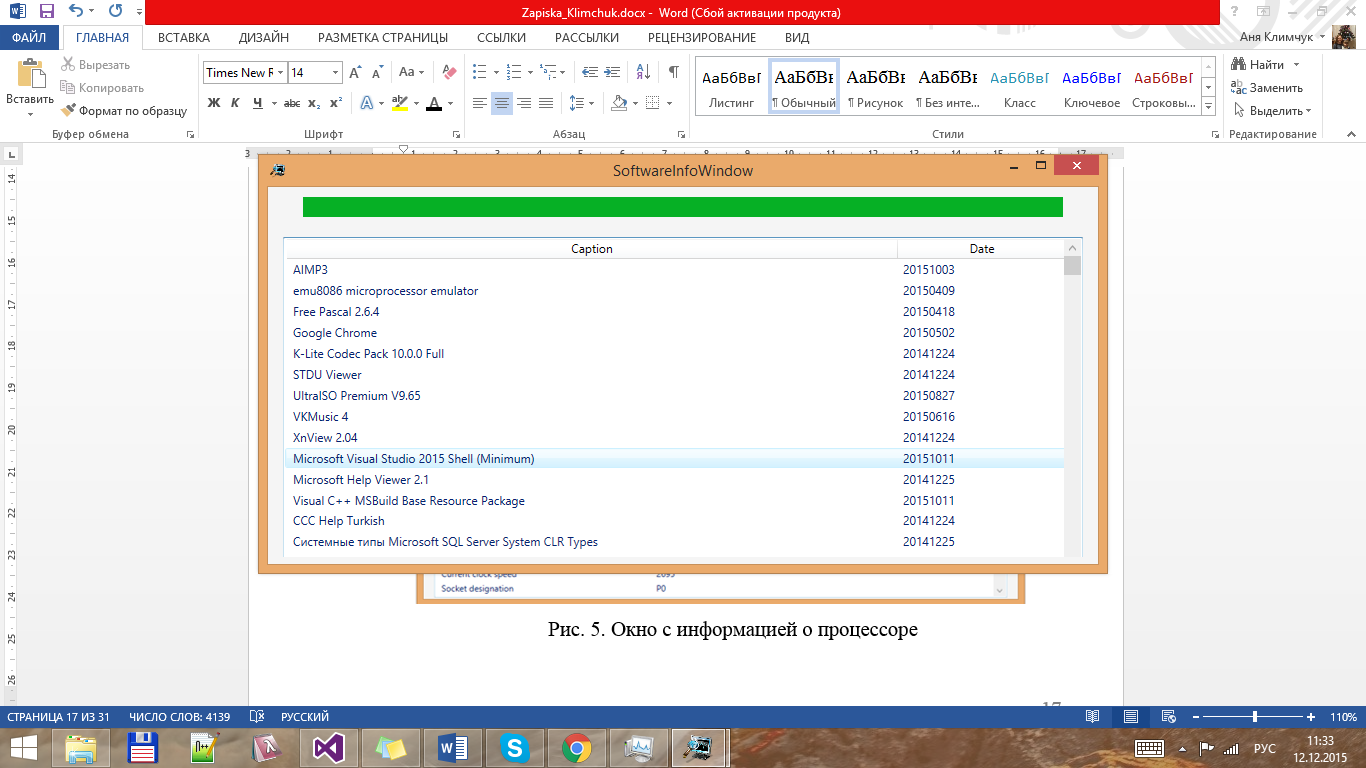


Рис. 6. Окно с информацией об установленном ПО

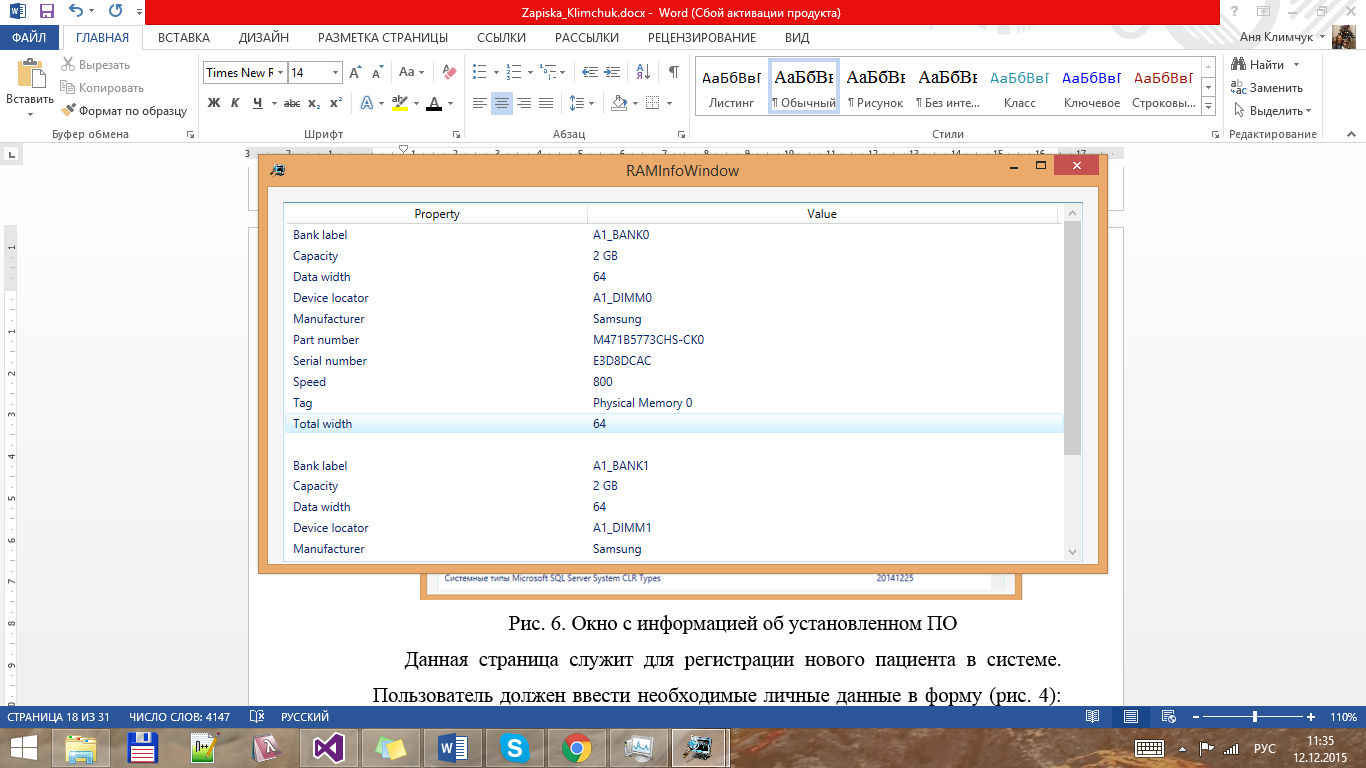


Рис. 7. Окно с информацией об RAM

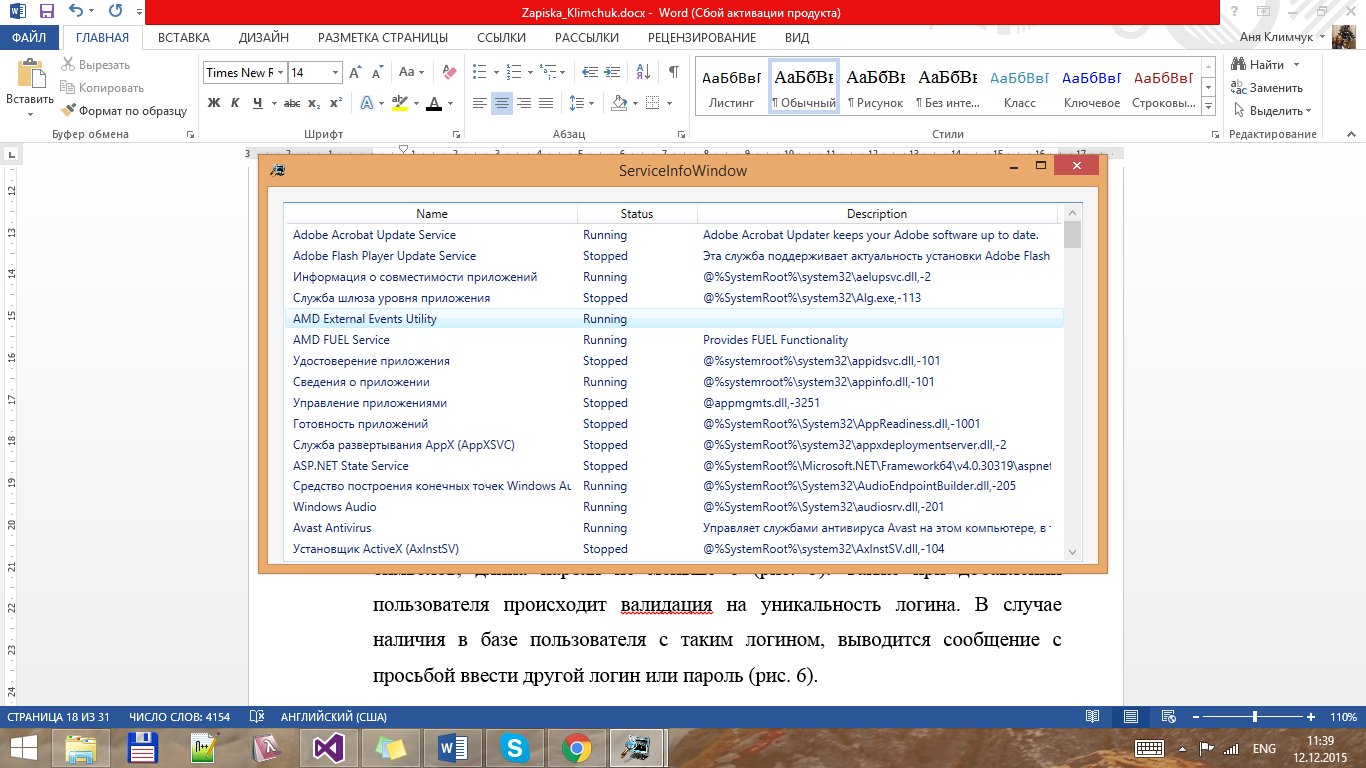


Рис. 8. Окно с информацией о сервисах

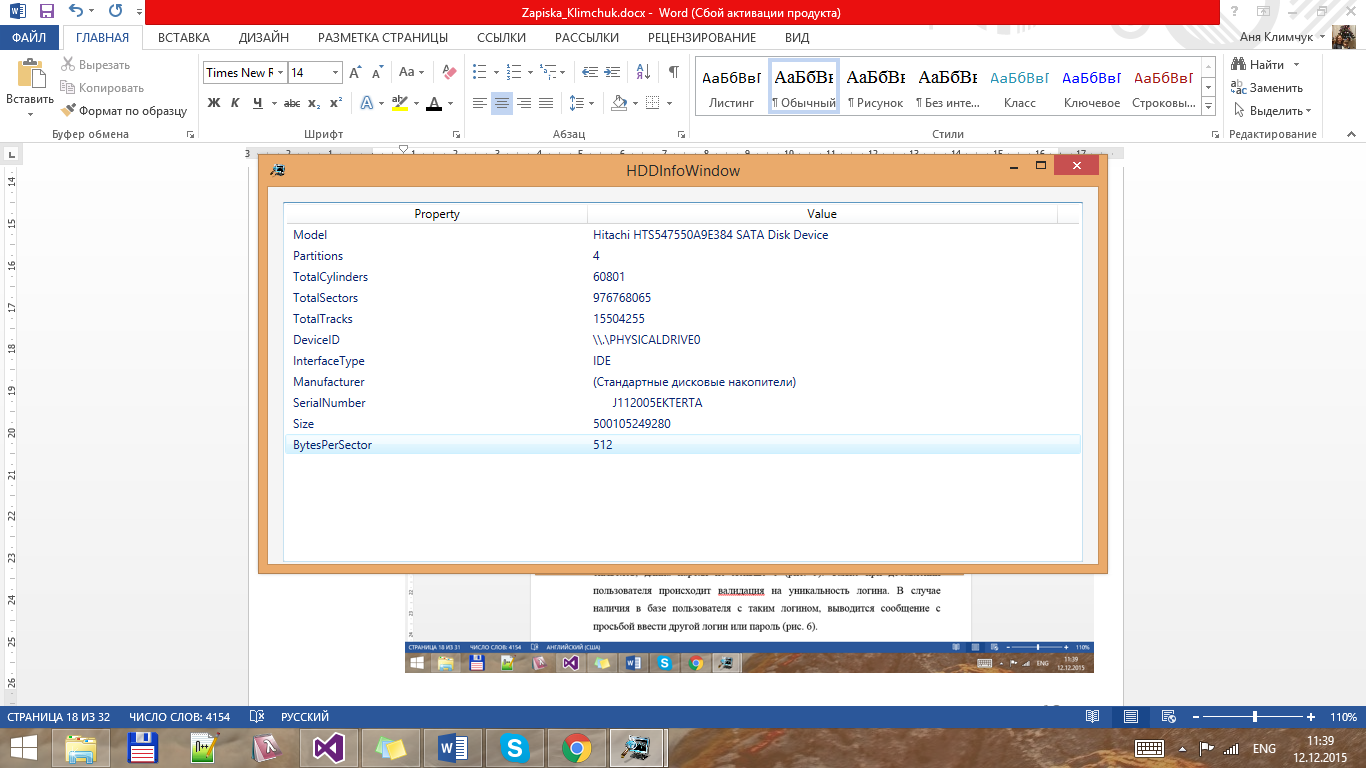


Рис. 9. Окно с информацией о жестком диске

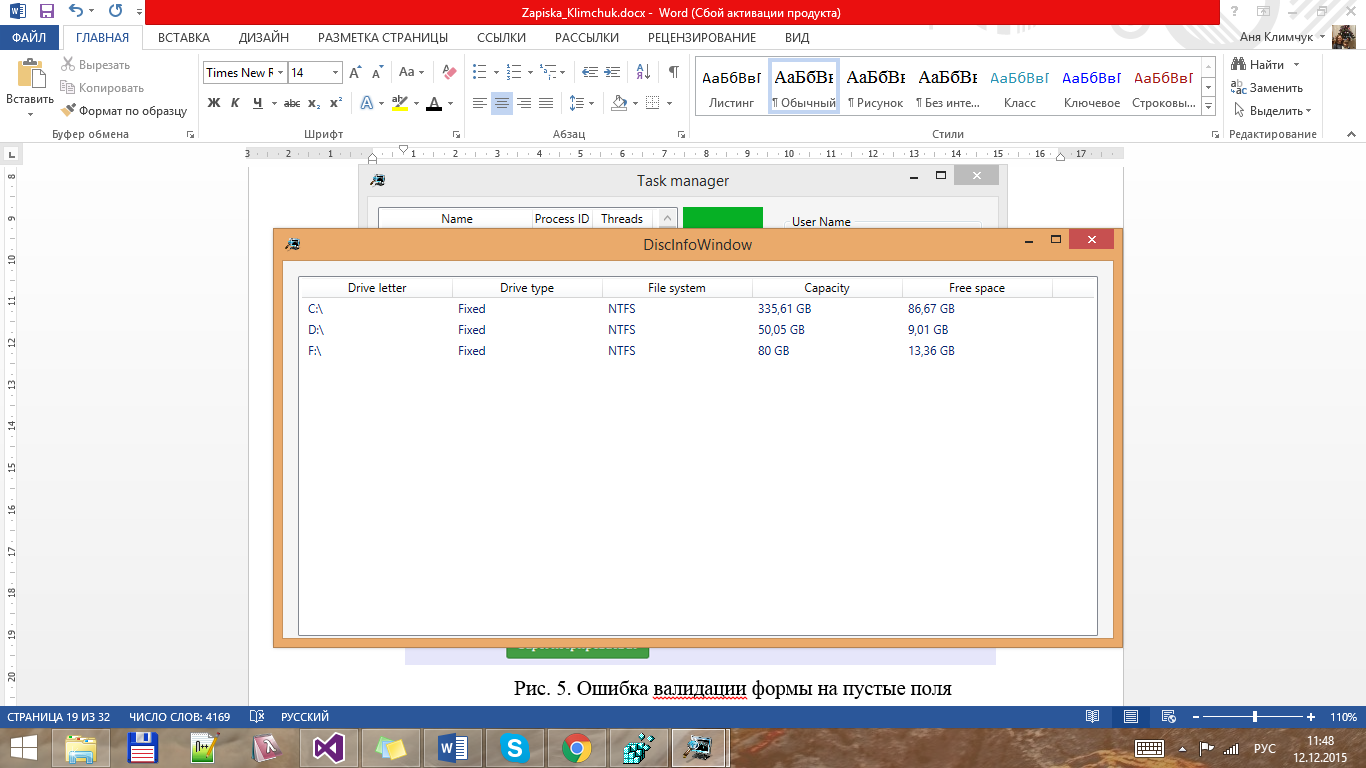


Рис. 10. Окно с информацией о дисках

О каждом процессе в системе можно получить информацию (рис. 11).

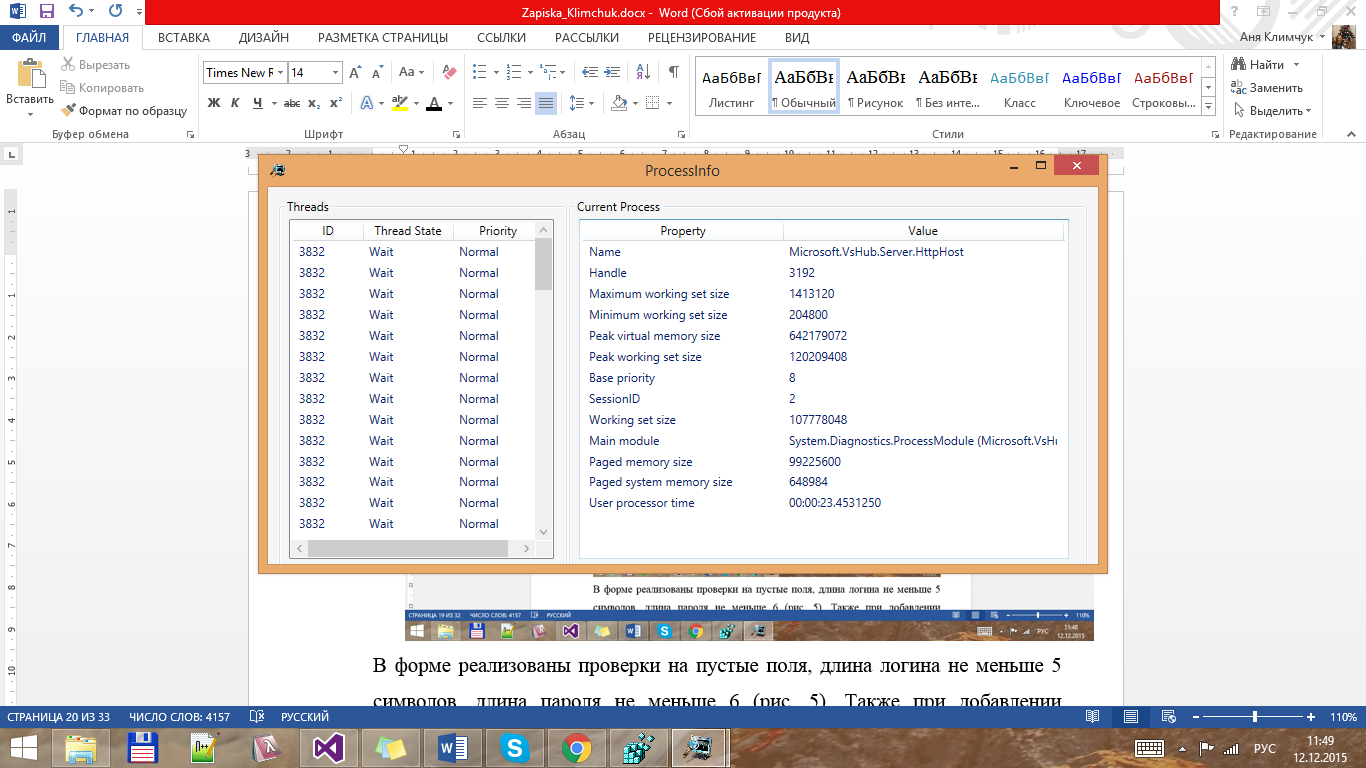


Рис. 11. Окно с информацией о конкретном процессе

В случае запрещенного доступа будет получено окно с предупреждением.

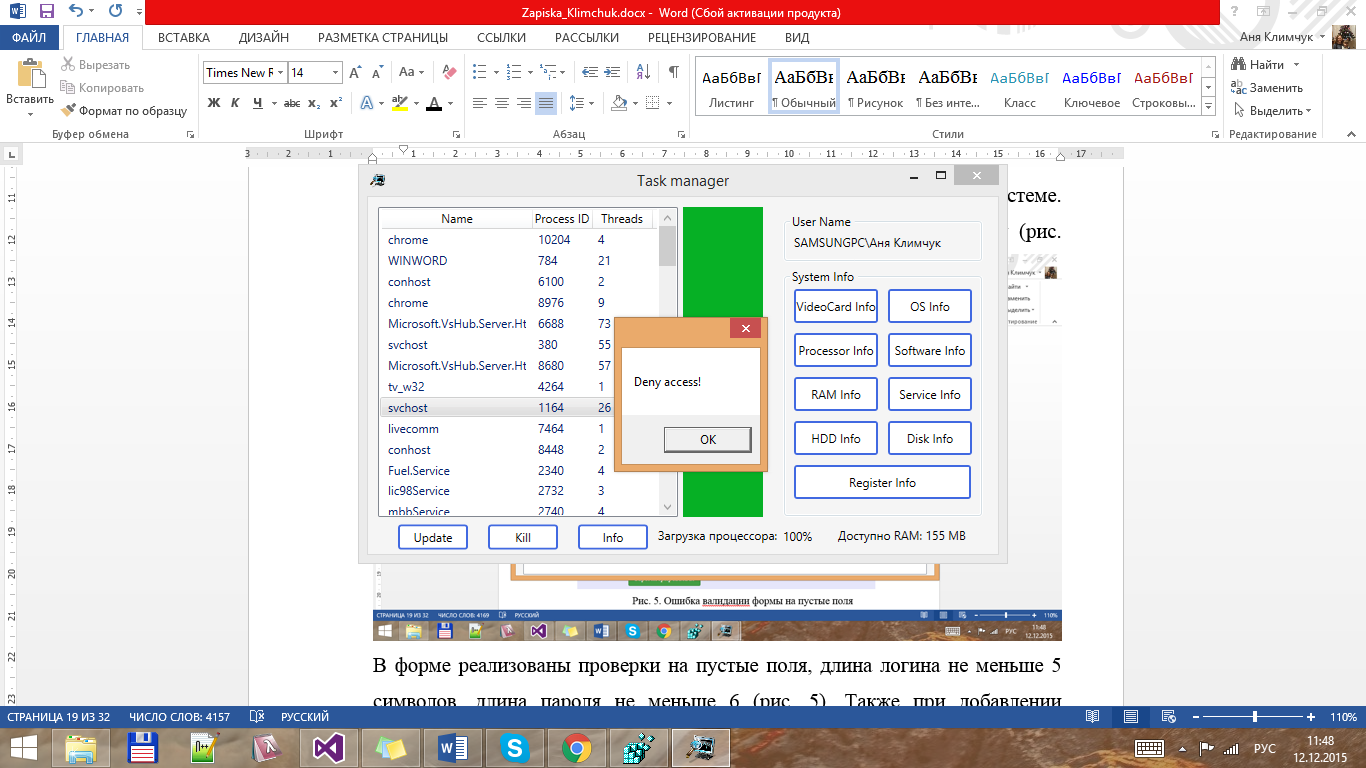


Рис. 12. Окно с предупреждением об отказе в доступе

В системе предоставлена возможность работы с реестром: удаление поддерева ключей, поиск по значению ключа, по регулярному выражению, чтение, удаление и запись значения ключа.

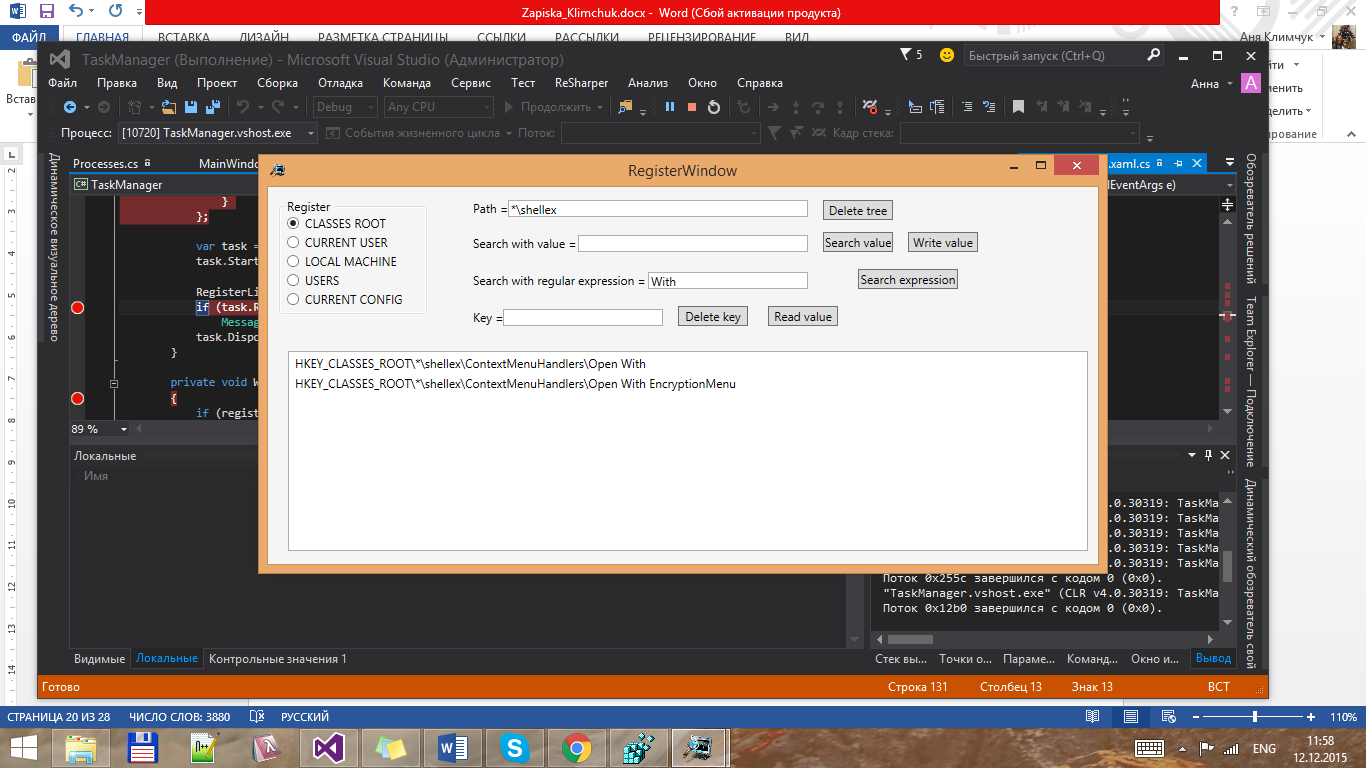


Рис. 13. Окно работы с реестром

**ЗАКЛЮЧЕНИЕ**

В результате работы над курсовым проектом был разработан программный продукт для сбора информации о ресурсах системы, таких как видеокарта, процессор, операционная система, диски, сервисы, оперативная память, установленное ПО. Система позволяет просмотреть все запущенные процессы, полную информацию и предоставляет возможность их завершения.

В процессе написания работы были закреплены навыки работы с технологией WPF, многопоточностью. При создании приложения были применены паттерны проектирования, способы синхронизации потоков, отложенная инициализация, WMI queries, Log Parser.

Разработанное программное средство представляет собой законченный программный продукт, готовый к использованию. Но при желании приложение можно доработать: расширить функциональность программного продукта, предоставить больше возможностей для управления ресурсами системы и т.д.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Волосевич, А. Технология Windows Presentation Foundation / А. Волосевич. – Минск. : БГУИР, 2014. – 108 с.

[2] Microsoft [Электронный ресурс]. – Электронные данные. – Режим доступа : <https://msdn.microsoft.com/>.

[3] Реестр в C# [Электронный ресурс]. – Электронные данные. – Режим доступа : [<http://brtrg.by/blog/post/536>/](https://msdn.microsoft.com/).

[4] Википедия [Интернет ресурс]. – Электронные данные. – Режим доступа: <http://ru.wikipedia.org/wiki/>.

# ПРИЛОЖЕНИЯ

### Приложение А Класс получения информации о текущем процессе (использование возможностей платформы .NET)

public class CurrentProcess

{

#region Fields

private Process process;

private List<CurrentProcess> info;

#endregion

#region Properties

public string Property { get; private set; }

public string Value { get; private set; }

#endregion

#region Methods

public List<CurrentProcess> GetCurrentProcessInformationById(int id)

{

info = new List<CurrentProcess>();

process = Process.GetProcessById(id);

info.Add(new CurrentProcess() { Property = "Name", Value = process.ProcessName.ToString() });

info.Add(new CurrentProcess() { Property = "Handle", Value = process.Handle.ToString() });

info.Add(new CurrentProcess() { Property = "Maximum working set size", Value = process.MaxWorkingSet.ToString() });

info.Add(new CurrentProcess() { Property = "Minimum working set size", Value = process.MinWorkingSet.ToString() });

info.Add(new CurrentProcess() { Property = "Peak virtual memory size", Value = process.PeakVirtualMemorySize.ToString() });

info.Add(new CurrentProcess() { Property = "Peak working set size", Value = process.PeakWorkingSet.ToString() });

info.Add(new CurrentProcess() { Property = "Base priority", Value = process.BasePriority.ToString() });

info.Add(new CurrentProcess() { Property = "SessionID", Value = process.SessionId.ToString() });

info.Add(new CurrentProcess() { Property = "Working set size", Value = process.WorkingSet.ToString() });

info.Add(new CurrentProcess() { Property = "Main module", Value = process.MainModule.ToString() });

info.Add(new CurrentProcess() { Property = "Paged memory size", Value = process.PagedMemorySize.ToString() });

info.Add(new CurrentProcess() { Property = "Paged system memory size", Value = process.PagedSystemMemorySize.ToString() });

info.Add(new CurrentProcess() { Property = "User processor time", Value = process.UserProcessorTime.ToString() });

return info;

}

#endregion

}

**Приложение Б Класс получения информации о процессоре (использование WMI)**

public class ProcessorInformation

{

private List<ProcessorInformation> info;

public string NameProperty { get; private set; }

public string Value { get; private set; }

public List<ProcessorInformation> GetProcessorInfo()

{

info = new List<ProcessorInformation>();

var query = new ManagementObjectSearcher("SELECT \* FROM Win32\_Processor");

foreach (ManagementObject obj in query.Get())

{

info.Add(new ProcessorInformation() { NameProperty = "Address width", Value = obj["AddressWidth"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Architecture", Value = obj["Architecture"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Availability", Value = obj["Availability"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Caption", Value = obj["Caption"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "CpuStatus", Value = obj["CpuStatus"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Creation class name", Value = obj["CreationClassName"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Data width", Value = obj["DataWidth"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "L2CacheSize", Value = obj["L2CacheSize"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Manufacturer", Value = obj["Manufacturer"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Max clock speed", Value = obj["MaxClockSpeed"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Name", Value = obj["Name"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Number of cores", Value = obj["NumberOfCores"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Number of logical processors", Value = obj["NumberOfLogicalProcessors"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "ProcessorId", Value = obj["ProcessorId"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Current clock speed", Value = obj["CurrentClockSpeed"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Socket designation", Value = obj["SocketDesignation"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Current voltage", Value = obj["CurrentVoltage"].ToString() });

info.Add(new ProcessorInformation() { NameProperty = "Status", Value = obj["Status"].ToString() });

}

return info;

}

}

**Приложение В Класс для получения актуальной информации о загрузке процессора и состоянии RAM (Double checked locking)**

public class ProcessorUsage

{

const float sampleFrequencyMillis = 1000;

private object syncLock = new object();

private PerformanceCounter counter;

private PerformanceCounter ramCounter;

private float result;

private DateTime lastSampleTime;

public ProcessorUsage()

{

this.counter = new PerformanceCounter("Processor", "% Processor Time", "\_Total", true);

this.ramCounter = new PerformanceCounter("Memory", "Available MBytes");

}

public float GetCurrentValue()

{

if ((DateTime.UtcNow - lastSampleTime).TotalMilliseconds > sampleFrequencyMillis)

{

lock (syncLock)

{

if ((DateTime.UtcNow - lastSampleTime).TotalMilliseconds > sampleFrequencyMillis)

{

result = counter.NextValue();

lastSampleTime = DateTime.UtcNow;

}

}

}

return result;

}

public float GetAvailableRAM()

{

return ramCounter.NextValue();

}

}

**Приложение Г Метод поиска в реестре с использованием классов Registry и RegistryKey**

public List<string> GetKeyPath(string searchKey)

{

info = new List<string>();

string path = string.Empty;

try

{

foreach (string keyname in baseRegistryKey.OpenSubKey(subKey).GetSubKeyNames())

{

try

{

using (RegistryKey key = baseRegistryKey.OpenSubKey(subKey).OpenSubKey(keyname, true))

{

foreach (var name in key.GetSubKeyNames())

{

MatchCollection coll = Regex.Matches(name.ToString(), searchKey);

if (coll != null)

{

foreach (var item in coll)

info.Add(baseRegistryKey.ToString() + "\\" + subKey + "\\" + keyname + "\\" + name);

}

}

}

}

catch (System.Security.SecurityException)

{

continue;

}

}

}

catch (NullReferenceException ex)

{

throw ex;

}

return info;

}

**Приложение Д Метод поиска в реестре с использованием Log Parser**

public List<string> Search(string value)

{

info = new List<string>();

RegRecordSet rs = null;

try

{

LogQuery qry = new LogQuery();

RegistryInputFormat registryFormat = new RegistryInputFormat();

string query = "SELECT Path from \\" + baseRegistryKey + "\\" + subKey + " where Value=\'" + value + "\'";

//string query = @"SELECT Path from \HKLM\SOFTWARE\Microsoft where Value='VisualStudio'";

rs = qry.Execute(query, registryFormat);

for (; !rs.atEnd(); rs.moveNext())

info.Add(rs.getRecord().toNativeString("\n"));

}

catch (System.IO.FileNotFoundException ex)

{

throw ex;

}

finally

{

if (rs != null)

rs.close();

}

return info;

}

**Приложение Е Класс главного окна MainWindow.sc**

public partial class MainWindow : Window

{

private SystemProcess systemProcess;

private ThreadsInfo threadsInfo;

private CurrentProcess currentProcess;

private VideoCardInformation videoCard;

private OSInformation os;

private ProcessorInformation processor;

private RAMInformation ram;

private DiskInformation disc;

private HDDInformation hdd;

private ProcessorUsage usage;

private System.Timers.Timer timer;

private ServiceInformation service;

private SoftwareInformation software;

private SoftwareInfoWindow softInfoWindow;

private bool isReady = false;

private List<SystemProcess> processes;

private List<ThreadsInfo> threads;

private List<CurrentProcess> currentProcessInfo;

private List<VideoCardInformation> videoCardInfo;

private List<RAMInformation> ramInfo;

private List<HDDInformation> hddInfo;

private List<SoftwareInformation> softInfo;

public MainWindow()

{

InitializeComponent();

systemProcess = new SystemProcess();

threadsInfo = new ThreadsInfo();

currentProcess = new CurrentProcess();

videoCard = new VideoCardInformation();

os = new OSInformation();

processor = new ProcessorInformation();

ram = new RAMInformation();

disc = new DiskInformation();

hdd = new HDDInformation();

usage = new ProcessorUsage();

service = new ServiceInformation();

software = new SoftwareInformation();

processes = new List<SystemProcess>();

currentProcessInfo = new List<CurrentProcess>();

InformationAboutProcesses();

UserNameLabel.Content = UserName.GetUserName().ToString();

timer = new System.Timers.Timer(500);

timer.Elapsed += new ElapsedEventHandler(UpdateProcessorUsage);

timer.Start();

}

private void UpdateProcessorUsage(object sender, ElapsedEventArgs e)

{

this.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Normal, (Action)(() =>

{

CPUProgressBar.Value = usage.GetCurrentValue();

ProcessorCount.Text = ((int)usage.GetCurrentValue()).ToString() + "%";

RAMCount.Text = ((int)usage.GetAvailableRAM()).ToString() + " MB";

}));

}

private void InformationAboutProcesses()

{

ProcessesList.ItemsSource = null;

ProcessesList.Items.Clear();

ProcessesList.ItemsSource = systemProcess.GetProcesses();

}

private void VideoCard\_Click(object sender, RoutedEventArgs e)

{

videoCardInfo = new List<VideoCardInformation>();

videoCardInfo.AddRange(videoCard.GetVideoCardInfo()[0]);

videoCardInfo.AddRange(videoCard.GetVideoCardInfo()[1]);

var videoCardInfoWindow = new VideoCardInfo();

videoCardInfoWindow.Show();

videoCardInfoWindow.VideoCardInfoList.ItemsSource = videoCardInfo;

}

private void Update\_OnClick(object sender, RoutedEventArgs e)

{

InformationAboutProcesses();

}

private void Kill\_Click(object sender, RoutedEventArgs e)

{

try

{

var item = ProcessesList.SelectedItem;

var process = item as SystemProcess;

systemProcess.KillProcessById(process.ProcessID);

}

catch (NullReferenceException ex)

{

MessageBox.Show("Process to kill is not selected!");

}

catch

{

MessageBox.Show("Deny access!");

}

Thread.Sleep(500);

InformationAboutProcesses();

}

private void Info\_Click(object sender, RoutedEventArgs e)

{

try

{

var item = ProcessesList.SelectedItem;

var process = item as SystemProcess;

threads = threadsInfo.GetThreadsInformationById(process.ProcessID);

currentProcessInfo = currentProcess.GetCurrentProcessInformationById(process.ProcessID);

}

catch (NullReferenceException ex)

{

MessageBox.Show("Process to info is not selected!");

return;

}

catch (Exception ex)

{

MessageBox.Show("Deny access!");

return;

}

var processInfoWindow = new ProcessInfoWindow();

processInfoWindow.Show();

processInfoWindow.ThreadsList.ItemsSource = threads;

processInfoWindow.CurrentProcessList.ItemsSource = currentProcessInfo;

}

private void OS\_Click(object sender, RoutedEventArgs e)

{

var osInfoWindow = new OSInfoWindow();

osInfoWindow.Show();

osInfoWindow.OSInfoList.ItemsSource = os.GetOSInfo();

}

private void Processor\_Click(object sender, RoutedEventArgs e)

{

var processorInfoWindow = new ProcessorInfoWindow();

processorInfoWindow.Show();

processorInfoWindow.ProcessorInfoList.ItemsSource = processor.GetProcessorInfo();

}

private void RAM\_Click(object sender, RoutedEventArgs e)

{

ramInfo = new List<RAMInformation>();

ramInfo.AddRange(ram.GetRAMInfo()[0]);

ramInfo.AddRange(ram.GetRAMInfo()[1]);

var ramInfoWindow = new RAMInfoWindow();

ramInfoWindow.Show();

ramInfoWindow.RAMInfoList.ItemsSource = ramInfo;

}

private void Disk\_Click(object sender, RoutedEventArgs e)

{

var discInfoWindow = new DiscInfoWindow();

discInfoWindow.DiscInfoList.ItemsSource = disc.GetDrivesInfo();

discInfoWindow.Show();

}

private void HDD\_Click(object sender, RoutedEventArgs e)

{

var hddInfoWindow = new HDDInfoWindow();

hddInfoWindow.Show();

hddInfoWindow.HDDInfoList.ItemsSource = hdd.GetHDDInfo();

}

private void Register\_Click(object sender, RoutedEventArgs e)

{

var regysterWindow = new RegisterWindow();

regysterWindow.Show();

}

private void Service\_Click(object sender, RoutedEventArgs e)

{

try

{

var serviceInfoWindow = new ServiceInfoWindow();

serviceInfoWindow.Show();

serviceInfoWindow.ServiceInfoList.ItemsSource = service.GetServiceInfo();

}

catch

{

MessageBox.Show("Can't download all services!");

}

}

private void Software\_Click(object sender, RoutedEventArgs e)

{

softInfoWindow = new SoftwareInfoWindow();

softInfoWindow.Show();

isReady = false;

softInfo = software.GetSoftwareInfo(out isReady);

timer = new System.Timers.Timer(1000);

timer.Elapsed += new ElapsedEventHandler(UpdateProgressBar);

timer.Start();

}

private void UpdateProgressBar(object sender, ElapsedEventArgs e)

{

this.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Normal, (Action)(() =>

{

if (isReady)

{

softInfoWindow.SoftProgressBar.Value = 100;

softInfoWindow.SoftwareInfoList.ItemsSource = softInfo;

}

}));

}

}