

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности
Кафедра информационных систем и программирования

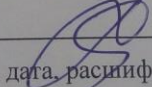
ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

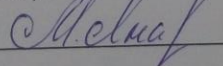
по дисциплине Системное программирование
на тему: Разработка службы логирования подключенных устройств в ОС Windows

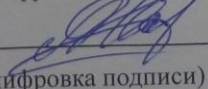
Выполнил студент группы 15- КБ-ПИ1 Ручка Артем Алексеевич

Допущен к защите: 25.12.18

Руководитель, нормоконтролер  Мурлин А.Г.
(подпись, дата, расшифровка подписи)

Защищён 25.12.18 Оценка отлично
(подпись, дата)

Члены комиссии  к.т.н., доцент Янаева М.В.
(подпись, дата, расшифровка подписи)

 с.п.к., Ковтун А.А.
(подпись, дата, расшифровка подписи)

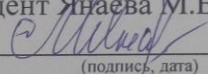
Краснодар
2018

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности
Кафедра информационных систем и программирования

УТВЕРЖДАЮ

Зав. кафедрой ИСП, к. техн. наук,
доцент Янаева М.В.

 2018 г.
(подпись, дата)

ЗАДАНИЕ

на курсовое проектирование

Студенту Ручка Артему Алексеевичу группы 15-КБ-ПИ1 4 курса
факультета Институт компьютерных систем и информационной безопасности
специальности 09.03.03 – Прикладная информатика

Тема работы: Разработка службы логирования подключенных устройств в ОС Windows

Содержание задания: Разработка службы Windows

Объем работы: 26 стр.

а) пояснительная записка к работе;

б) программы 1.

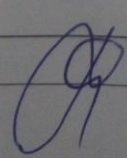
Рекомендуемая литература: Бек, Л. Введение в системное программирование / Л. Бек. - М.: Мир, 2016.-448 с.


Срок выполнения работы: _____ с «10» сентября по «22» декабря 2018 г.

Срок защиты: _____ до «29» декабря 2018 г.

Дата выдачи задания: _____ «10» сентября 2018 г.

Дата сдачи проекта на кафедру: _____ «25» декабря 2018 г.

Руководитель работы  Мурлин А.Г.
(подпись)

Задание принял студент  Ручка А.А.

Реферат

Пояснительная записка курсового проекта - с. 26, рис. 2, источников 5, прил. 2.

СИСТЕМНАЯ СЛУЖБА, СЛУЖБА WINDOWS, WMI, CIM, VISUAL STUDIO 2015, WBEM, WQL, SCM, ЛОГИРОВАНИЕ, WINDOWS FORMS

Объектом разработки является системная служба Windows.

Цель работы заключается в программной реализации службы Windows, которая отслеживает подключаемые usb-устройства и записывает в файл данные о времени их подключения или отключения.

К основным полученным результатам относятся системная служба и пользовательское приложение, взаимодействующее со службой.

Содержание

Введение.....	5
1 Нормативные ссылки	6
2 Инструментарий управления Windows	7
2.1 Классы, объекты, свойства и методы WMI	8
2.2 Язык запросов WQL.....	9
3 Общая информация о системной службе	10
3.1 Системная служба Windows.....	10
3.2 Общая структура системной службы.....	11
4 Разработка программного обеспечения	14
4.1 Разработка службы Windows	14
4.2 Разработка пользовательского приложения	17
Заключение	21
Список используемых источников.....	22
Приложение А	23
Приложение Б	24

Введение

Системное программное обеспечение предназначено для управления работой компьютера, распределения его ресурсов, поддержки диалога с пользователями, оказания им помощи в обслуживании компьютера, а также для частичной автоматизации разработки новых программ.

Программы, работающие на компьютере можно разделить на несколько категорий:

- прикладные программы, непосредственно обеспечивающие выполнение необходимых пользователям работ: редактирование текстов, рисование картинок, обработка информационных массивов;
- системные программы, выполняющие различные вспомогательные функции, например, создание копий используемой информации, выдачу справочной информации о компьютере, проверка работоспособности устройств компьютера.

Цель курсового проекта состоит в написании системной службы Windows, которая должна отслеживать подключаемые USB-устройства и записывать в файл время их подключения и отключения.

Пояснительная записка должна отражать следующие этапы работы: разработку службы Windows, разработку пользовательского приложения, описание работы программы.

1 Нормативные ссылки

В данной пояснительной записке использованы следующие нормативные ссылки:

1. ГОСТ Р 1.5-2004 Стандарты национальные Российской Федерации. Правила построения, изложения, оформления и обозначения.
2. ГОСТ 2.104-68 ЕСКД. Основные надписи
3. ГОСТ 7.80-2000 СИБИД. Библиографическая запись. Заголовок. Общие требования и правила составления.
4. ГОСТ 7.82-2001 СИБИД. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления.
5. ГОСТ 7.9-95 СИБИД. Реферат и аннотация. Общие требования.
6. ГОСТ 19.001-77 ЕСПД. Общие положения.
7. ГОСТ 19.104-78 ЕСПД. Основные надписи.
8. ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам.
9. ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению.
10. ГОСТ 19.402-78 ЕСПД. Описание программы.
11. ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.

2 Инструментарий управления Windows

Для выполнения данного курсового проекта был использован инструментарий управления Windows(аббревиатура WMI расшифровывается как Windows Management Instrumentation - инструментарий управления Windows).

Из названия ясно, для чего создана и применяется эта технология, стоит лишь добавить, что она давно перешагнула рамки управления только операционной системой Windows и позволяет контролировать множество других совместимых с ней приложений. WMI – это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением ОС Windows. Технология WMI – это расширенная и адаптированная под Windows реализация стандарта WBEM(Web-Based Enterprise Management), 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 WBEM лежит Common Information Model(общая информационная модель или CIM), реализующая объектно-ориентированный подход к представлению компонентов системы. CIM является расширяемой моделью, что позволяет программам, системам и драйверам добавлять в нее 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.

2.1 Классы, объекты, свойства и методы WMI

Поскольку WMI построена по объектно-ориентированному принципу, то все данные об операционной системе, ее свойствах, управляемых приложениях и обнаруженном оборудовании представлены в виде объектов. Каждый тип объекта описан классом, в состав которого входят свойства и методы. Определения классов описаны в MOF-файлах, а объекты этих классов с заполненными свойствами и доступными методами при их вызове возвращаются WMI-провайдерами. Управляет созданием и удалением объектов, а также вызовом их методов служба CIM Object Manager.

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.

2.2 Язык запросов WQL

Для обращения к объектам WMI используется специфический язык запросов WMI Query Language(WQL), который является одним из разновидностей SQL. Основное его отличие от 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.

Ниже приведен пример некоторых типичных WQL-запросов:

```
SELECT * FROM Win32_LogicalDisk WHERE FileSystem IS NULL
```

```
SELECT * FROM Win32_LogicalDisk WHERE FileSystem IS NOT  
NULL
```

```
SELECT * FROM Win32_LogicalDisk WHERE FileSystem = "NTFS"
```

```
SELECT * FROM Win32_DiskDrive WHERE Partitions < 2 OR  
SectorsPerTrack > 100
```

```
SELECT * FROM Win32_LogicalDisk WHERE (Name = "C:" OR  
Name = "D:") AND FreeSpace > 2000000 AND FileSystem = "NTFS"
```

```
SELECT * FROM Win32_NTLogEvent WHERE Logfile =  
'Application'
```

```
SELECT * FROM Meta_Class WHERE __Class LIKE %Win32%
```

```
SELECT * FROM __InstanceCreationEvent WHERE TargetInstance  
ISA "Win32_NTLogEvent" GROUP WITHIN 600 BY  
TargetInstance.SourceName HAVING NumberOfEvents > 25
```

3 Общая информация о системной службе

3.1 Системная служба Windows

Сервис, или служба Windows, - это фоновое приложение, которое может запускаться различными способами, в том числе автоматически при старте Windows, или стартовать в том случае, если окажется нужным другому подобному приложению. Отличительной особенностью сервиса является отсутствие средств непосредственного диалога с пользователем и, соответственно, отсутствие необходимости для пользователя предпринимать какие-либо действия для поддержания сервиса в работоспособном состоянии. Большинство сервисов могут быть остановлены по запросу пользователя, однако есть и такие, которые не могут быть остановлены по причине того, что они необходимы для нормальной работы некоторых компонентов операционной системы.

Обычно в виде сервисов реализуются различные серверы (например, серверы баз данных), службы мониторинга и прочие приложения, которые могут работать совершенно независимо от действий пользователя.

Для управления сервисами в Windows используется инструмент, называемый Service Control Manager (менеджер управления службами или SCM). Это приложение занимается запуском, остановкой, удалением и добавлением сервисов через специальные API, предусмотренные в системе специально для взаимодействия с сервисами. Существуют механизмы удалённого запуска и управления сервисами, что позволяет системным администраторам оперативно управлять сервисами на клиентских машинах.

3.2 Общая структура системной службы

Все вызовы и обращения к системной службе проходят через менеджер управления службами. Когда служба запускается автоматически при старте системы или вручную, то SCM обращается к методу Main в классе Program:

```
static class Program
{
    static void Main()
    {
        ServiceBase[] ServicesToRun;
        ServicesToRun = new ServiceBase[]
        {
            new Service1()
        };
        ServiceBase.Run(ServicesToRun);
    }
}
```

Метод Main() по умолчанию определен таким образом, чтобы запускать сразу несколько служб, которые определены в массиве ServicesToRun. Однако по умолчанию проект содержит только одну службу Service1. Сам запуск производится с помощью метода Run: ServiceBase.Run(ServicesToRun).

Сама запускаемая служба представлена узлом Service1.cs. Однако на самом деле это не простой файл кода. Если мы откроем этот узел, то увидим в нем файл дизайнера службы Service1.Designer.cs и класс Service1.

Класс Service1 собственно представляет службу. По умолчанию он имеет следующий код:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;

namespace FileWatcherService
{
    public partial class Service1 : ServiceBase
    {
        public Service1()
        {
            InitializeComponent();
        }

        protected override void OnStart(string[] args)
        {
        }

        protected override void OnStop()
        {
        }
    }
}

```

Класс службы должен наследоваться от базового класса `ServiceBase`. Этот класс определяет ряд методов, важнейшие из которых метод `OnStart()`, который запускает действия, выполняемые службой, и метод `OnStop()`, останавливающий службу.

После того, как SCM вызовет метод `Main` и зарегистрирует службу, происходит непосредственный ее вызов через запуск метода `OnStart`.

Когда в консоли служб или через командную строку мы посылаем команду на остановку службы, то SCM обращается к методу `OnStop` для ее остановки.

Кроме этих двух методов в классе службы можно переопределить еще несколько методов базового класса `ServiceBase`:

- `OnPause`: вызывается при приостановке службы;
- `OnContinue`: вызывается при возобновлении работы службы после ее приостановки;
- `OnShutdown`: вызывается при завершении работы Windows;
- `OnPowerEvent`: вызывается при изменении режима электропитания;
- `OnCustomCommand`: вызывается при получении службой пользовательской команды от Менеджера Управления Службами (`Service Control Manager / SCM`).

4 Разработка программного обеспечения

В данном курсовом проекте была разработана системная служба Windows, которая записывает в лог-файл имя устройства и время его подключения/отключения от персонального компьютера, а также было создано приложение для просмотра информации из этого файла.

4.1 Разработка службы Windows

Опишем основные методы реализуемые в разработанной службе.

Метод OnStart(), который выполняется при запуске службы, написан следующим образом:

```
file = new StreamWriter(new FileStream("C:\\DebugKPSP\\KPSP.log",  
                                     System.IO.FileMode.Append));  
this.file.WriteLine("-----");  
this.file.WriteLine("Служба запущена " + System.DateTime.Now);  
this.file.WriteLine("-----");
```

Если файл KPSP.log отсутствует в папке C:\DebugKPSP\, то он создается и в него записываются данные о времени запуска службы. В противном случае данные записываются в конец файла.

Также в этом методе запускается выполнение фоновой операции BackgroundWorker:

```
backgroundWorker1.RunWorkerAsync();
```

Основной код элемента BackgroundWorker описывается в методе backgroundWorker1_DoWork(object sender, DoWorkEventArgs e):

```
WqlEventQuery insertQuery = new WqlEventQuery("SELECT *  
FROM __InstanceCreationEvent WITHIN 2 WHERE TargetInstance ISA  
'Win32_USBHub");
```

```
ManagementEventWatcher insertWatcher = new  
ManagementEventWatcher(insertQuery);
```

С помощью данных строк создается экземпляр класса WqlEventQuery для события, которое вызывается при подключении USB-устройства, и, с помощью запроса на языке WQL, мы получаем данные о подключенных USB-устройствах.

Данная строка регистрирует новое событие и осуществляет его обработку с помощью метода DeviceInsertedEvent():

```
insertWatcher.EventArrived += new  
EventArrivedEventHandler(DeviceInsertedEvent);
```

Аналогично происходит обработка события, при котором USB-устройство извлекается из компьютера:

```
WqlEventQuery removeQuery = new WqlEventQuery("SELECT *  
FROM __InstanceDeletionEvent WITHIN 2 WHERE  
TargetInstance ISA 'Win32_USBHub'");
```

```
ManagementEventWatcher removeWatcher = new  
ManagementEventWatcher(removeQuery);
```

```
removeWatcher.EventArrived += new  
EventArrivedEventHandler(DeviceRemovedEvent);
```

```
removeWatcher.Start();
```

Метод DeviceInsertedEvent(object sender, EventArrivedEventArgs e) выполняется при появлении события insertWatcher:

```
private void DeviceInsertedEvent(object sender,  
EventArrivedEventArgs e)  
{  
    ManagementBaseObject instance =  
        (ManagementBaseObject)e.NewEvent["TargetInstance"];  
    foreach (var property in instance.Properties)
```

```

{
    if (property.Name == "Caption")
        this.file.WriteLineAsync("Устройство: " + property.Value + "
                                подключено " + DateTime.Now);
}
this.file.Flush();
}

```

Данный метод ищет недавно подключенное устройство и записывает в файл его название и время подключения.

Метод DeviceInsertedEvent(object sender, EventArgs e) выполняется при появлении события removeWatcher, которое возникает при извлечении USB-устройства:

```

private void DeviceRemovedEvent(object sender,
EventArgs e)
{
    ManagementBaseObject instance =
        (ManagementBaseObject)e.NewEvent["TargetInstance"];
    foreach (var property in instance.Properties)
    {
        if (property.Name == "Caption")
            this.file.WriteLineAsync("Устройство: " + property.Value + "
                                    отключено " + DateTime.Now);
    }
    this.file.Flush();
}

```

Данный метод записывает в файл название и время отключения извлеченного устройства.

Метод OnStop() выполняется при остановке службы. Он записывает в файл данные, о том в какое время служба завершает работу:

```
this.file.WriteLine("-----");  
this.file.WriteLine("Служба остановлена " + DateTime.Now);  
this.file.WriteLine("-----");  
this.file.Flush();  
this.file.Close();
```

4.2 Разработка пользовательского приложения

Для управления службой и просмотра лог-файла было создано клиентское приложение.

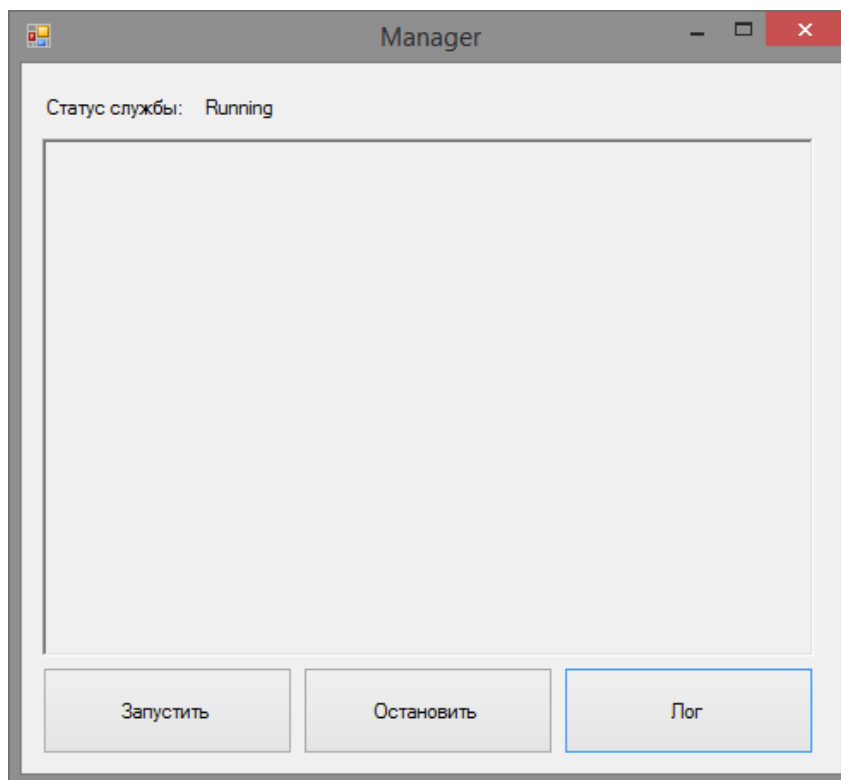


Рисунок 1 - Основное окно пользовательского приложения

С помощью пользовательского приложения возможно запускать и останавливать разработанную службу, следить за статусом службы, а также возможно просматривать лог-файл, созданный во время работы службы.

Код кнопки «Запустить»:

```
private void exec_button1_Click(object sender, EventArgs e)
{
    try
    {
        if (serviceController.Status.ToString() == "Stopped")
        {
            serviceController.Start();
            serviceController.WaitForStatus(ServiceControllerStatus.Running);
            label2.Text = serviceController.Status.ToString();
        }
        else MessageBox.Show("Служба уже запущена");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

При нажатии данной кнопки проверяется статус службы и, если она остановлена, подается команда на запуск службы. Если служба уже запущена появляется предупреждение о том, что служба уже работает.

Код кнопки «Остановить»:

```
private void stop_button2_Click(object sender, EventArgs e)
{
    try
    {
        if (serviceController.Status.ToString() == "Running")
        {

```

```

        serviceController.Stop();
        serviceController.WaitForStatus(ServiceControllerStatus.Stopped);
        label2.Text = serviceController.Status.ToString();
    }
    else MessageBox.Show("Служба уже остановлена");
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

При нажатии данной кнопки проверяется статус службы и, если она запущена, подается команда на остановку работы службы. Если служба уже остановлена появляется предупреждение о том, что служба уже остановлена.

Код кнопки «Лог»:

```

private void log_button3_Click(object sender, EventArgs e)
{
    File.Copy("C:\\DebugKPSP\\KPSP.log", "C:\\DebugKPSP\\KPSP-
copy.log", true);
    StreamReader r = new StreamReader("C:\\DebugKPSP\\KPSP-
copy.log");
    richTextBox1.Text = r.ReadToEnd().ToString();
    r.Close();
    File.Delete("C:\\DebugKPSP\\KPSP-copy.log");
}

```

При нажатии данной кнопки программа копирует файл KPSP.log и считывает информацию из него. Это сделано для того чтобы не нарушать

работу службы, так как файл KPSP.log может быть занят в данный момент и при попытке считывания из него информации может возникнуть ошибка. Полученная информация заносится в richTextBox, расположенный на главной форме пользовательского приложения.

При нажатии кнопки Лог в текстовое поле на форме выводятся данные из файла KPSP.log:

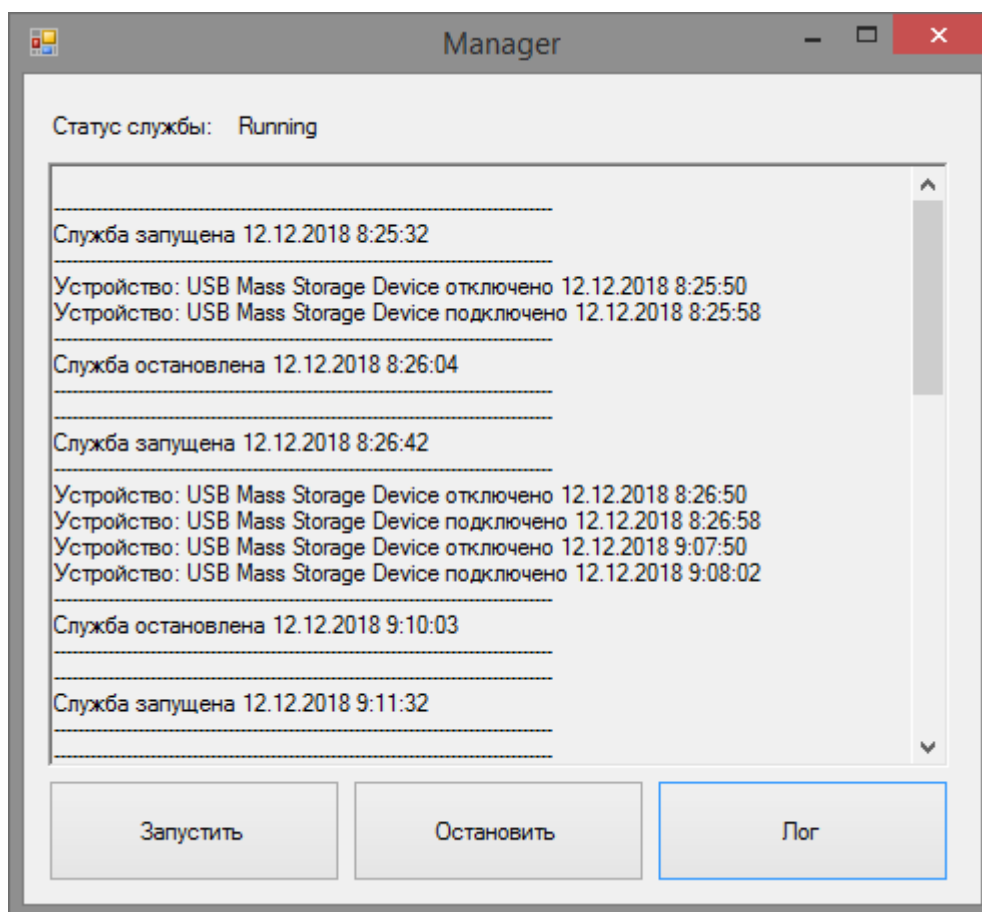


Рисунок 2 – Информация из лог-файла

Заключение

В результате данного курсового проекта были выполнены все поставленные задачи, повышен навык работы с платформой .NET Framework, языком программирования C#.

В результате выполнения курсового проекта была выполнена основная цель работы, которая заключалась в создании службы Windows, которая записывает информацию о времени подключения и отключения USB- устройствах в лог-файл, и клиентское приложение, с помощью которого возможно управлять запуском и остановкой службы, а также просматривать создаваемый лог-файл.

Также была освоена работа с инструментарием управления Windows(WMI), мы научились работать с WQL-запросами, и был приобретен опыт разработки службы Windows в среде разработки Visual Studio 2015 на языке программирования C#.

Список используемых источников

1. Александреску, А. Язык программирования D / А. Александреску. — М.: Символ, 2013. — 536 с.
2. Дорогов, В.Г. Основы программирования на языке C: Учебное пособие / В.Г. Дорогов, Е.Г. Дорогова; Под общ. ред. проф. Л.Г. Гагарина. — М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. — 224 с.
3. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен; Пер. с англ. Ю.Н. Артеменко. — М.: Вильямс, 2013. — 1312 с.
4. Кетков, Ю.Л. Введение в системное программирование на языке ассемблера ЕС ЭВМ / Ю.Л. Кетков, В.С. Максимов, А.Н. Рябов. - М.: Наука, 2018. - 264 с.
5. Макаров, А. В. Common Intermediate Language и системное программирование Microsoft .NET / А.В. Макаров, С.Ю. Скоробогатов, А.М. Чеповский. - М.: Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2014. - 328 с.

Проверка на оригинальность:

Отчет о проверке на заимствования №1



Автор: mgonluc@gmail.com / ID: 5147160
Проверяющий: mgonluc@gmail.com / ID: 5147160)
Отчет предоставлен сервисом «Антиплагиат»: <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 29
Начало загрузки: 23.12.2018 20:11:43
Длительность загрузки: 00:00:00
Имя исходного файла: Курсовая работа СП
Ручка
Размер текста: 502 кБ
Символов в тексте: 26259
Слов в тексте: 2922
Число предложений: 369

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 23.12.2018 20:11:44
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска:



Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общепотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	11.56%	11.56%	Программа, выдающая полную информацию о наличии...	http://knowledge.allbest.ru	раньше 2011	Модуль поиска Интернет	8	8
[02]	0%	11.56%	Программа, выдающая полную информацию о наличии...	http://knowledge.allbest.ru	27 Apr 2013	Модуль поиска Интернет	0	8
[03]	6.76%	7.01%	Не знаете, что такое WMI? Тогда мы идем к Вам! windo...	http://microsin.net	04 Янв 2016	Модуль поиска Интернет	11	13

Еще источников: 7
Еще заимствований: 13,17%

Приложение Б

Листинг системной службы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Management;

namespace KursSP
{
    public partial class KursSP : ServiceBase
    {
        public KursSP()
        {
            InitializeComponent();
        }
        private StreamWriter file;
        protected override void OnStart(string[] args)
        {
            file = new StreamWriter(new FileStream("C:\\\\DebugKPSP\\KPSP.log",
                                                    System.IO.FileMode.Append));
            this.file.WriteLine("-----");
            this.file.WriteLine("Служба запущена " + System.DateTime.Now);
            this.file.WriteLine("-----");
            this.file.Flush();
            backgroundWorker1.RunWorkerAsync();
        }

        protected override void OnStop()
        {
            this.file.WriteLine("-----");
            this.file.WriteLine("Служба остановлена " + DateTime.Now);
            this.file.WriteLine("-----");
            this.file.Flush();
            this.file.Close();
        }

        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            WqlEventQuery insertQuery = new WqlEventQuery("SELECT * FROM
__InstanceCreationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_USBHub'");

            ManagementEventWatcher insertWatcher = new
ManagementEventWatcher(insertQuery);
            insertWatcher.EventArrived += new
EventArrivedEventHandler(DeviceInsertedEvent);
            insertWatcher.Start();
        }
    }
}
```



```

        WqlEventQuery removeQuery = new WqlEventQuery("SELECT * FROM
__InstanceDeletionEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_USBHub'");
        ManagementEventWatcher removeWatcher = new
ManagementEventWatcher(removeQuery);
        removeWatcher.EventArrived += new
EventArrivedEventHandler(DeviceRemovedEvent);
        removeWatcher.Start();

        // Do something while waiting for events
        System.Threading.Thread.Sleep(20000000);
    }
    private void DeviceInsertedEvent(object sender, EventArrivedEventArgs e)
    {
        ManagementBaseObject instance =
        (ManagementBaseObject)e.NewEvent["TargetInstance"];
        foreach (var property in instance.Properties)
        {
            if (property.Name == "Caption")
                this.file.WriteLineAsync("Устройство: " + property.Value + "
подключено " + DateTime.Now);
        }
        this.file.Flush();
    }

    private void DeviceRemovedEvent(object sender, EventArrivedEventArgs e)
    {
        ManagementBaseObject instance =
        (ManagementBaseObject)e.NewEvent["TargetInstance"];
        foreach (var property in instance.Properties)
        {
            if (property.Name == "Caption")
                this.file.WriteLineAsync("Устройство: " + property.Value + "
отключено " + DateTime.Now);
        }
        this.file.Flush();
    }
}
}

```

Листинг пользовательского приложения:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.ServiceProcess;

namespace KPSP_manager
{
    public partial class Form1 : Form
    {
        public Form1()
    }
}

```

```

{
    InitializeComponent();
    label2.Text = serviceController.Status.ToString();
}
ServiceController serviceController = new ServiceController("KursSP");
private void exec_button1_Click(object sender, EventArgs e)
{
    try
    {
        if (serviceController.Status.ToString() == "Stopped")
        {
            serviceController.Start();
            serviceController.WaitForStatus(ServiceControllerStatus.Running);
            label2.Text = serviceController.Status.ToString();
        }
        else MessageBox.Show("Служба уже запущена");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void stop_button2_Click(object sender, EventArgs e)
{
    try
    {
        if (serviceController.Status.ToString() == "Running")
        {
            serviceController.Stop();
            serviceController.WaitForStatus(ServiceControllerStatus.Stopped);
            label2.Text = serviceController.Status.ToString();
        }
        else MessageBox.Show("Служба уже остановлена");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void log_button3_Click(object sender, EventArgs e)
{
    File.Copy("C:\\DebugKPSP\\KPSP.log", "C:\\DebugKPSP\\KPSP-copy.log",
true);
    StreamReader r = new StreamReader("C:\\DebugKPSP\\KPSP-copy.log");
    richTextBox1.Text = r.ReadToEnd().ToString();
    r.Close();
    File.Delete("C:\\DebugKPSP\\KPSP-copy.log");
}
}
}

```