Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

# Диспетчер задач

Студент:
гр. 053504
Сергеев Е.Г

Руководитель:
Ассистент
кафедры информатики
Давыдчик А.В

Минск 2023

# Содержание

# 1 Введение

Основная идея данного курсового проекта – показать возможность взаимодействия с операционной системой (в данном случае ОС Windows) при помощи языка программирования C#. В реализации программного средства используется среда разработки Visual Studio 2019 и платформа .NET 5.0.

Немного о языке программирования C#:

C# (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота[6] как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Delphi, Модула, Smalltalk и, в особенности, Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

И о среде разработки:

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ.

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 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).

# 2 Определение основного функционала приложения

В данном приложении была реализована возможность отслеживать текущее состояние компьютера по таким критерием, как нагрузка на процессор, так же узнать информацию о системе, такой как текущие процессы, потоки, информация о видеокарте, ОС, процессоре, локальном диске и накопителе.

# 3 Обзор используемых технологий

В введении было отмечено, что использовался язык C# и Visual Studio 2019. Для отображения функционала программы была использована технология WPF, сокращение от Windows Presentation Foundation. Данная технология широко применяется в разработке оконных приложений и является одной из наиболее простых и удобных.

Технология WPF использует многоуровневую архитектуру. На вершине ваше приложение взаимодействует с высокоуровневым набором служб, которые полностью написаны на управляемом коде C#. Действительная работа по трансляции объектов .NET в текстуры и треугольники Direct3D происходит "за кулисами", с использованием низкоуровневого неуправляемого компонента по имени **milcore.dll**. Библиотека milcore.dll реализована в неуправляемом коде потому, что ей требуется тесная интеграция с Direct3D, и вдобавок для нее чрезвычайно важна производительность.

# 4 Проектирование приложения

Для реализации основного функционала программы была использована событийно-ориентированная концепция.

При этом на форме находятся компоненты доступные пользователю и при любом взаимодействию с ними вызываются встроенные события технологии WPF с помощью которых происходит программная обработка и ответ на запрос пользователя.

# 5 Программная реализация

Класс DiskInformation, который содержит методы для получения информации о дисках, подключенных к системе.

Класс имеет пять свойств — Capacity, DriveLetter, DriveType, FileSystem и FreeSpace, — которые представляют собой емкость, название диска, тип диска, файловую систему и свободное пространство на диске.

Метод GetDrivesInfo возвращает список объектов DiskInformation, представляющих информацию обо всех дисках, подключенных к системе.

Он начинается с инициализации пустого списка info, а затем использует метод DriveInfo.GetDrives() для получения массива всех дисков, подключенных к системе.

Далее он создает новый объект DiskInformation, задает для его свойств соответствующие значения, полученные из объекта DriveInfo, и добавляет его в информационный список.

Класс HDDInformation извлекает информацию о жестких дисках (HDD) в системе с помощью инструментария управления Windows (WMI). Класс содержит информацию частного списка объектов HDDInformation, в которых будет храниться информация о жестком диске.

Класс имеет два свойства, NameProperty и Value, которые будут содержать имя и значение каждого атрибута жесткого диска. Он также имеет метод GetHDDInfo, который возвращает список объектов HDDInformation.

Метод GetHDDInfo сначала получает список всех моделей жестких дисков, установленных в системе, с помощью метода GetModelsInfo. Затем он просматривает список моделей и извлекает информацию о жестком диске для каждой модели с помощью запроса WMI. Информация извлекается с помощью класса ManagementObject, представляющего объект WMI.

Затем метод вызывает метод AddInList для каждого атрибута, который он хочет получить, передавая имя атрибута и объект ManagementObject. Метод AddInList пытается получить значение атрибута, используя параметр value и индексатор объекта ManagementObject. В случае успеха он создает новый объект HDDInformation с именем и значением атрибута и добавляет его в информационный список.

Класс OSInformation получить информацию об операционной системе (ОС), работающей на текущем компьютере, с помощью WMI.

Метод GetOSInfo, который возвращает список объектов OSInformation. Этот метод запрашивает WMI с помощью класса Win32_OperatingSystem и извлекает различные свойства ОС, такие как номер сборки, заголовок, свободная физическая память, свободная виртуальная память, имя, тип ОС,

зарегистрированный пользователь, серийный номер, основная версия пакета обновления, служба упаковать дополнительную версию, статус, системное устройство, системный каталог, системный диск, версию и каталог Windows.

Полученные свойства добавляются в информационный список, и этот список возвращается в конце метода. Значения свойств свободной физической памяти и свободной виртуальной памяти преобразуются из байтов в мегабайты и округляются до двух знаков после запятой перед добавлением в информационный список.

Класс SystemProcess получает информацию о запущенных процессах в системе, а также может завершить процесс  по его идентификатору. Для взаимодействия с системными процессами использовалось пространство имён System.Diagnostics.

Класс имеет три свойства: ProcessName (имя процесса), ProcessID (идентификатор процесса), Threads (количество потоков, запущенных в процессе).Так же в классе есть два метода: GetProcesses и KillProcessById.

GetProcesses возвращает список объектов SystemProcess, содержащих информацию обо всех запущенных процессах в системе.

KillProcessById принимает целочисленный параметр, который является идентификатором процесса, который необходимо завершить. Если операция завершается неудачно, генерируется исключение.

Класс ProcessorInformation извлекает информацию о процессорах в текущей системе с помощью WMI. Этот класс имеет два свойства: NameProperty хранит имя информации о свойствах процессора, например (Имя, Количество ядер, Максимальная тактовая чистота и др.). Value хранит значение этого свойства.

Метод GetProcessorInfo использует ManagementObjectSearcher для поиска и получения информации о процессорах в текущей системе с помощью класса WMI Win32_Processor. Затем он перебирает каждый объект процессора, возвращенный запросом, создает новый объект ProcessorInformation для каждого соответствующего свойства (определенного в методе) и добавляет его в список сведений. Наконец, метод GetProcessorInfo возвращает список сведений.

Класс ServiceInformation предоставляет информации обо всех службах Windows, установленных на локальном компьютере. Он имеет следующие свойства: Name (имя), Status (текущий статус службы), Description (описание службы )

Метод GetServiceInfo извлекает информацию обо всех службах на компьютере и возвращает список объектов ServiceInformation, содержащих упомянутые выше свойства.

Метод GetUserName(), класса UserName, возвращает имя текущего пользователя, вошедшего в систему, в формате «домен\имя пользователя».

Класс VideoCardInformation получает информацию о видеокартах, установленных на компьютере, с помощью WMI. Класс имеет два свойства: NameProperty (имя извлекаемого свойства ) Value (значение извлекаемого свойства).

Метод GetVideoCardInfo() извлекает информацию обо всех видеокартах, установленных на компьютере, с помощью WMI-запроса. Для каждой найденной видеокарты создается список объектов VideoCardInformation, содержащий имя и его значение. Затем этот список добавляется в информационный список, который представляет собой список списков, где каждый список представляет отдельную видеокарту.

Метод GetCurrentProcessInformationById, класса CurrentProcess, извлекает информацию о конкретном процессе, по его идентификатору. Информация хранится в списке объектов CurrentProcess, где каждый объект представляет собой пару свойство-значение. войства включают имя процесса, его дескриптор, размер рабочего набора, размер виртуальной памяти, базовый приоритет, идентификатор сеанса и др.

Класс MainWindow, который содержит обработчики событий для нажатия кнопок и таймер, обновляющий индикатор загрузки ЦП. Так же этот класс создаёт экземпляры различных других классов для сбора информации о системе, такой как текущие процессы, потоки, информация о видеокарте, ОС, процессоре, локальном диске и накопителях.

Так же класс MainWindow определяет несколько обработчиков событий для нажатия кнопок, которые открывают новые окна, отображающие дополнительную системную информацию. Эти окна включают VideoCardInfo, ProcessInfoWindow, OSInfoWindow, ProcessorInfoWindow, DiscInfoWindow, HDDInfoWindow и ServiceInfoWindow.

Приложение извлекает информацию о процессах и отображает ее в виде списка. Можно просмотреть информацию о потоках и текущем процессе, связанном с этим процессом просто щелкнув на него. Также можно просматривать информацию о видеокарте системы, ОС, процессоре, диске и жестком диске, а также просматривать список всех служб, работающих в системе.

# 6 Заключение

Таким образом было реализовано программное средство для анализа производительности подсистем компьютера и управления процессами запущенными на нём.

Привожу скриншоты работы приложения (рисунок 1):

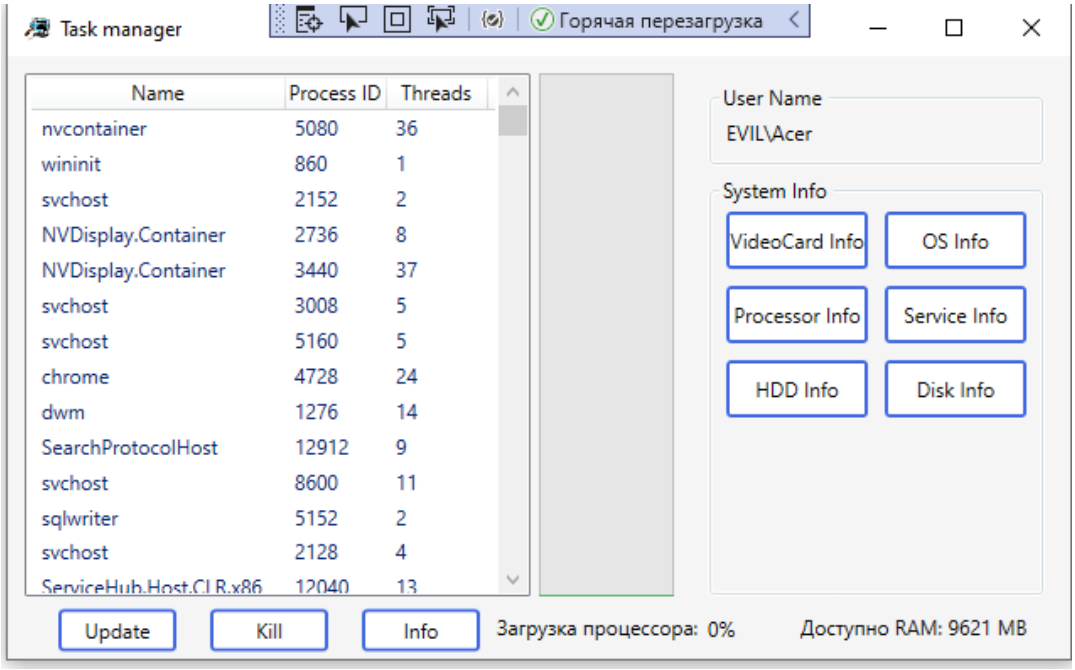


Рисунок 1. Начальное окно приложения с основными элементами приложения.

Список запущенных процессов на данный момент (рисунок 2):

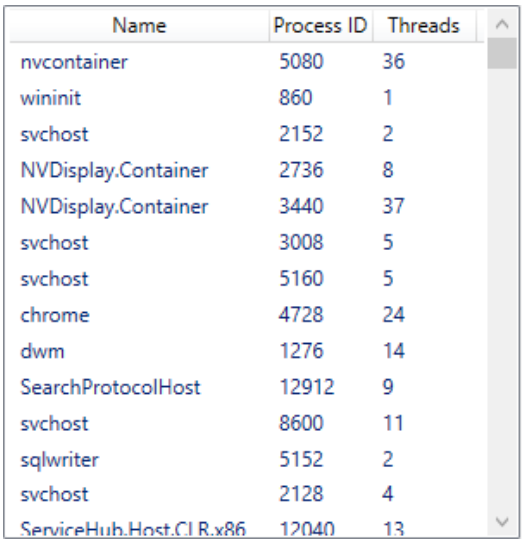


Рисунок 2. Список процессов на пользовательском ПК

Отображение нагрузки на процессор (рисунок 3):

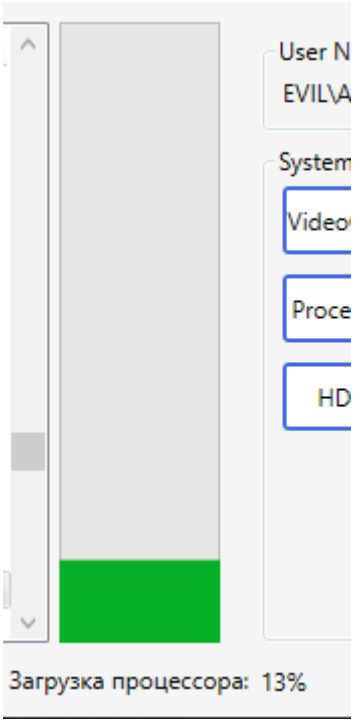


Рисунок 3. Нагрузка процессора на пользовательском ПК

Отображение информации о выбранном процессе (рисунок 4):

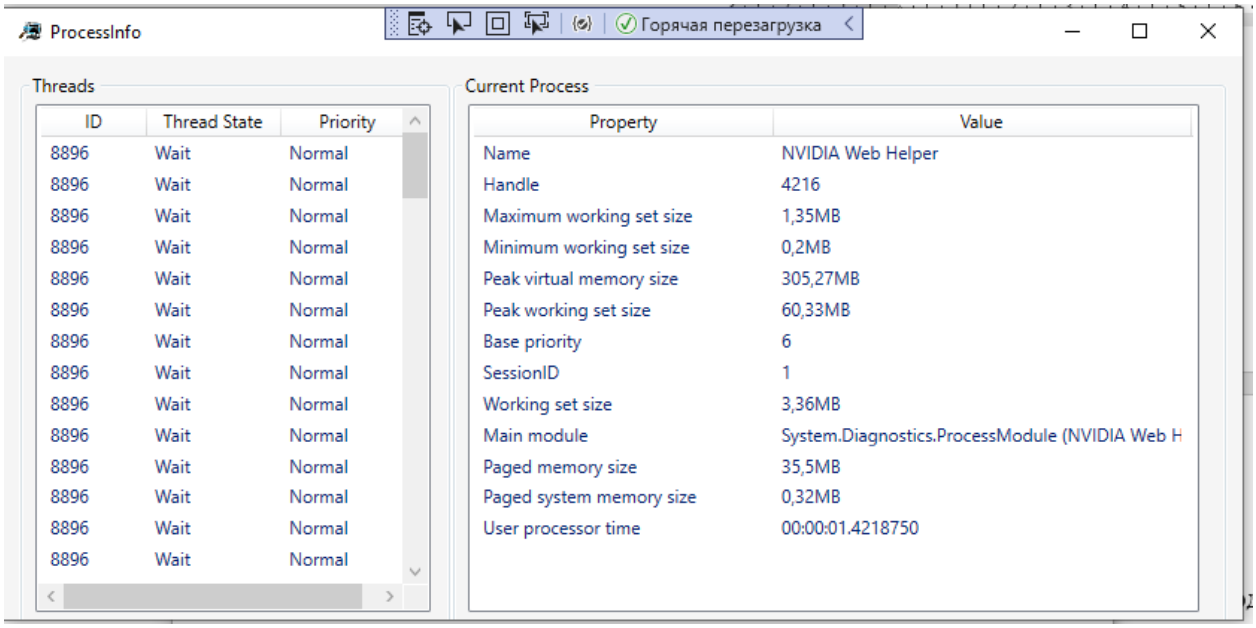


Рисунок 4. Подробная информация о выбранном процессе

Демонстрация завершения процесса
До нажатия кнопки “Kill” (рисунок 5.1):

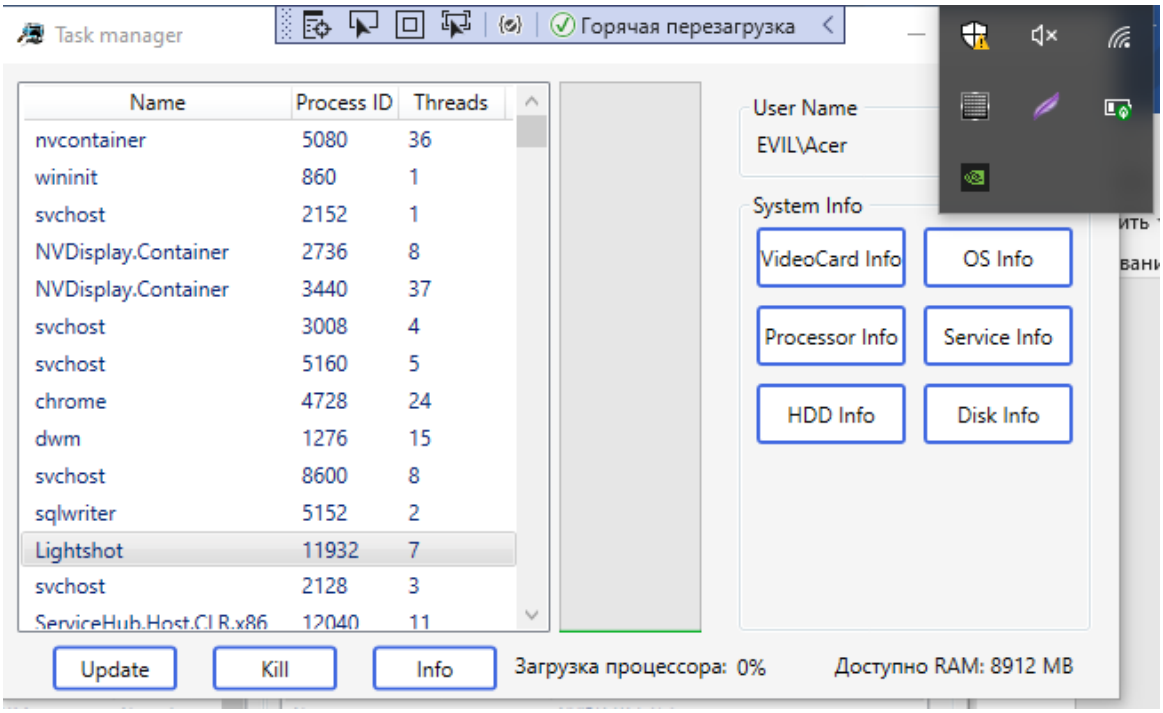


Рисунок 5.1 Демонстрация завершения процесса

После нажатия “Kill” (рисунок 5.2):

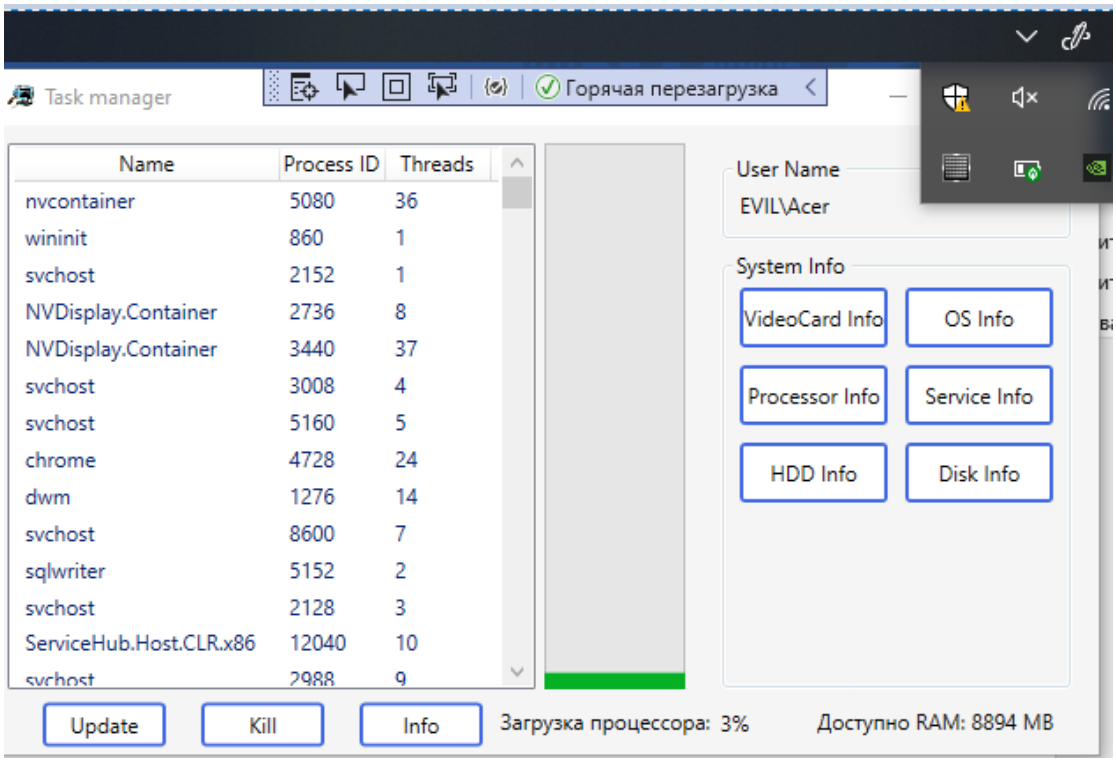


Рисунок 5.2 Демонстрация завершения процесса

Если это является системным процессов, то при завершении вызывается исключение (рисунок 6):

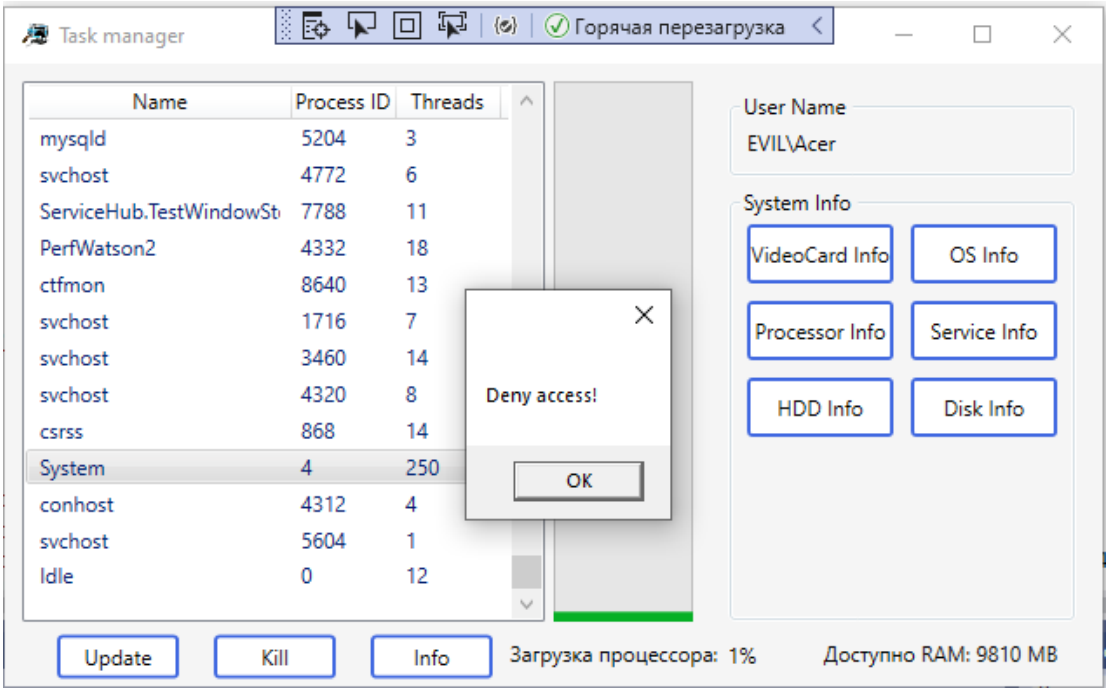


Рисунок 6. Демонстрация исключения

Кнопка “OS Info” выводит информацию о ОС (рисунок 7):

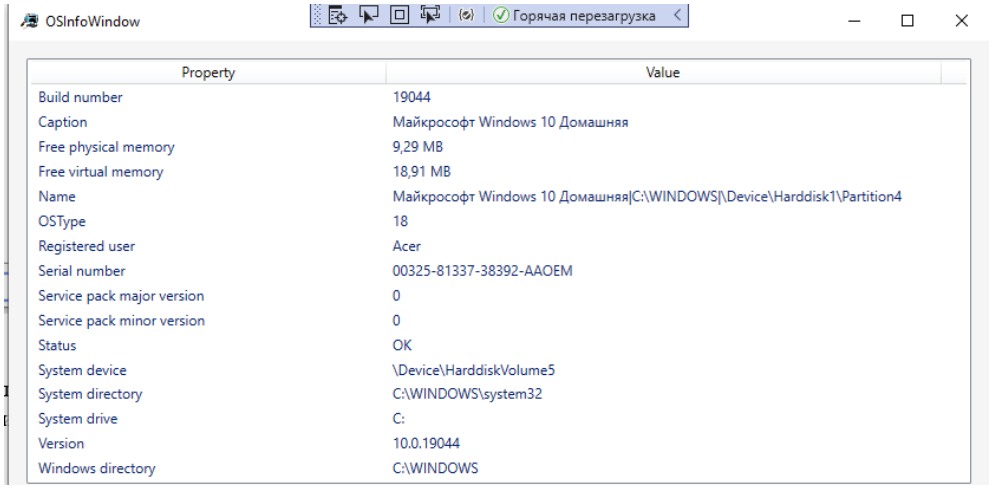


Рисунок 7. Информация о ОС пользователя

Кнопка “Service Info” выводит информацию о службах (рисунок 8):

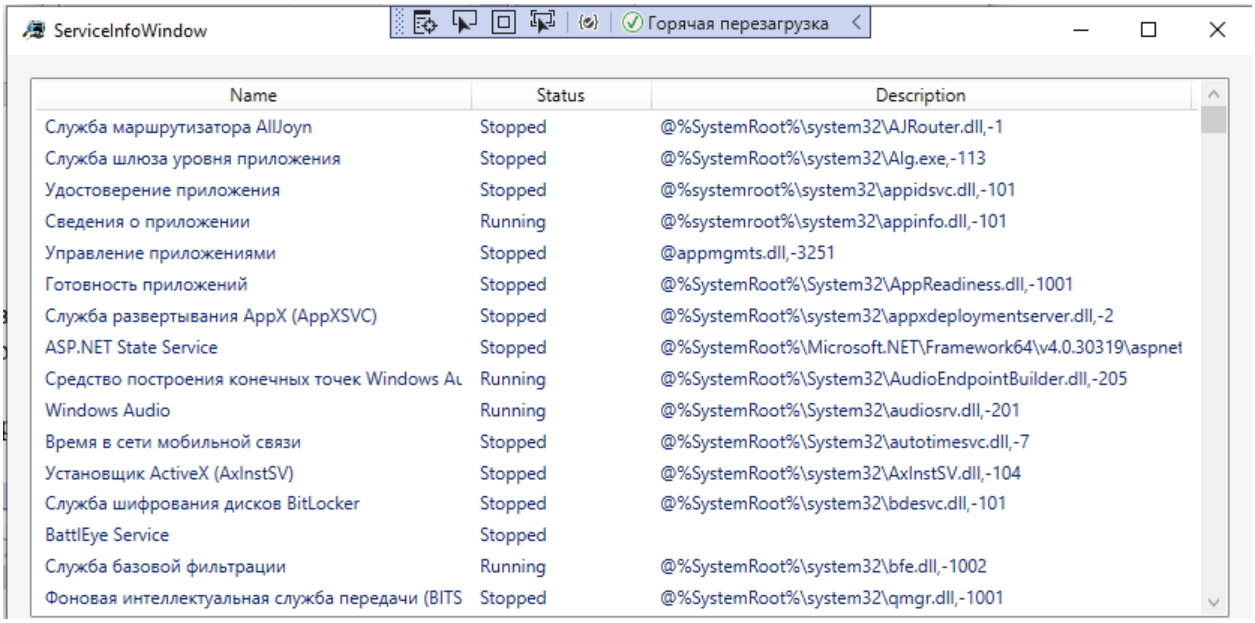


Рисунок 8. Информация о службах на пользовательском ПК

Кнопка “Disk Info” выводит информацию о локальных дисках (рисунок 9):

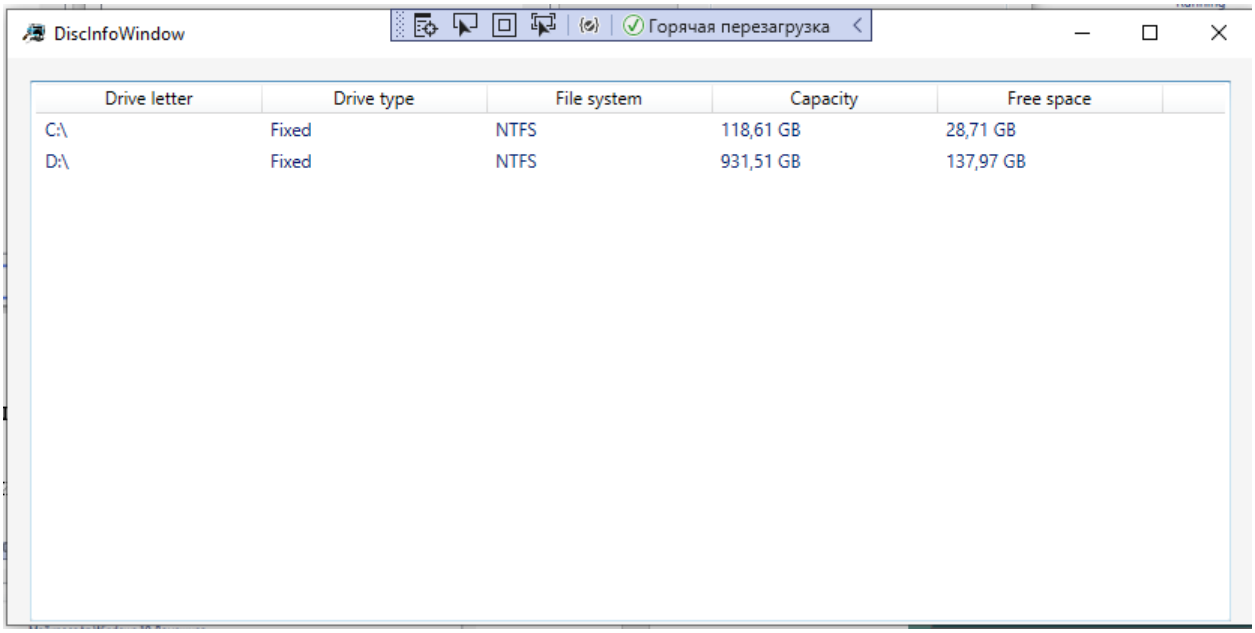


Рисунок 9. Информация о дисках имеющихся на пользовательском ПК

Кнопка “VideoCard Info” выводит информацию о видеоадаптерах (рисунок 10):

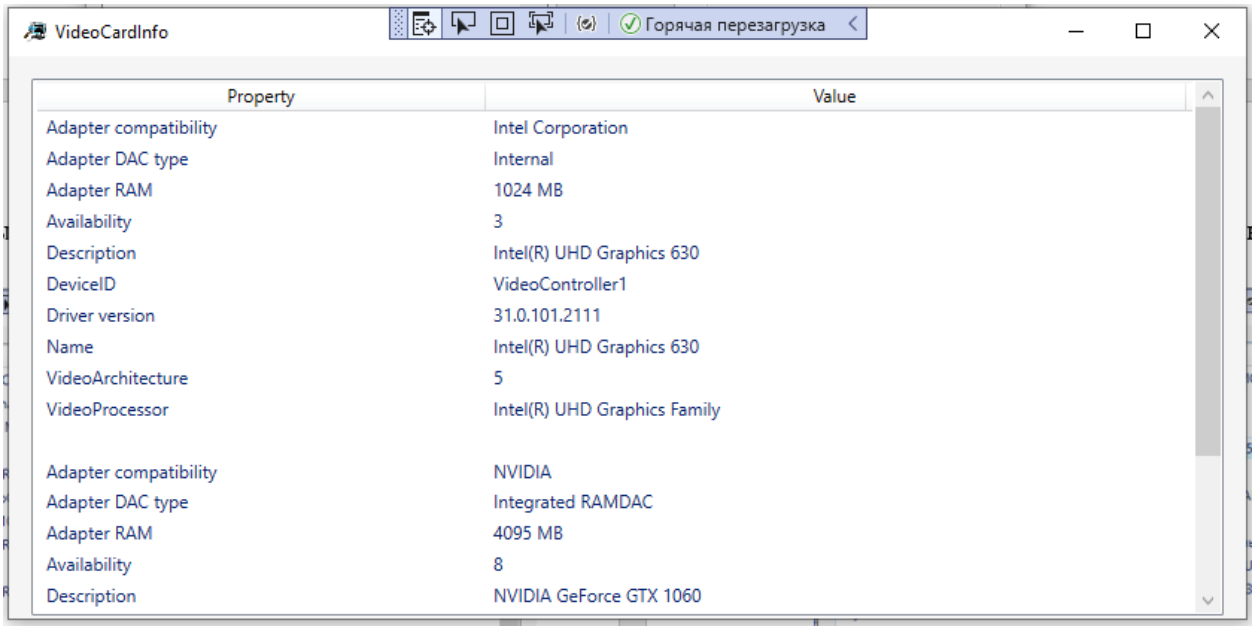


Рисунок 10. Информация о видеоадаптерах на пользовательском ПК

Кнопка “Processor Info” выводит информацию о процессоре (рисунок 11):

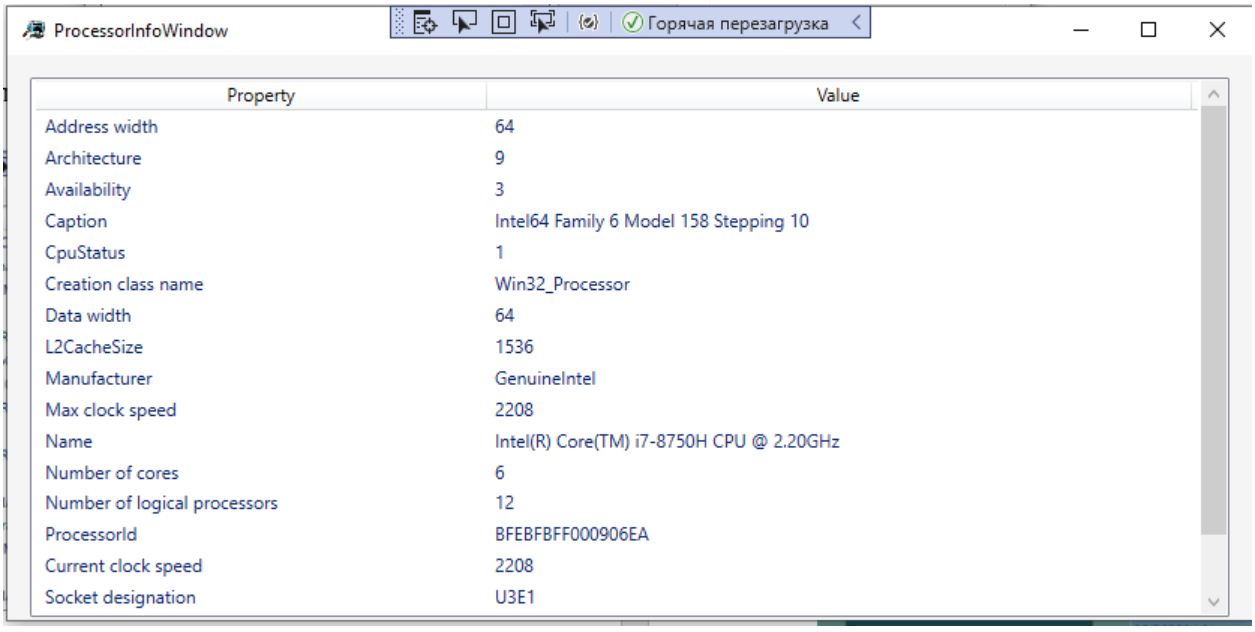


Рисунок 11. Информация о процессоре на пользовательском ПК

Кнопка “HDD Info” выводит информацию о накопителях (рисунок 12):

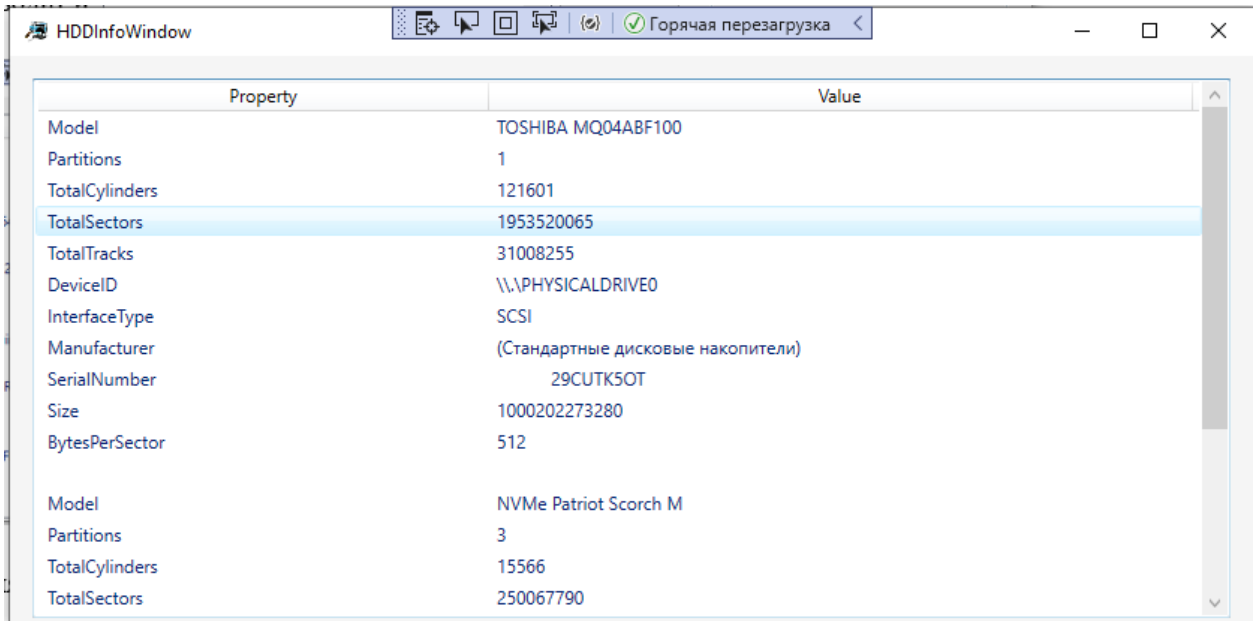


Рисунок 12. Информация о  накопителях в пользовательском ПК

# 7 Список использованных источников

Официальный сайт с документацией по C# и .NET - https://docs.microsoft.com/en-us/dotnet/csharp/

Книга “Язык программирования C# 7 и платформы .NET и .NET Core | Джепикс Филипп, Троелсен Эндрю”

## Приложение А - Текст программы

```
using Microsoft.Win32;
using ProcessesInformation;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using System.Timers;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace TaskManager
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private SystemProcess systemProcess;
        private ThreadsInfo threadsInfo;
        private CurrentProcess currentProcess;
        private VideoCardInformation videoCard;
        private OSInformation os;
        private ProcessorInformation processor;
        private DiskInformation disc;
        private HDDInformation hdd;
        private ProcessorUsage usage;
        private System.Timers.Timer timer;
        private ServiceInformation service;
        private bool isReady = false;
        private List<SystemProcess> processes;
        private List<ThreadsInfo> threads;
        private List<CurrentProcess> currentProcessInfo;
        private List<VideoCardInformation> videoCardInfo;
        private List<HDDInformation> hddInfo;


        public MainWindow()
        {
            InitializeComponent();
            systemProcess = new SystemProcess();
            threadsInfo = new ThreadsInfo();
            currentProcess = new CurrentProcess();
            videoCard = new VideoCardInformation();
            os = new OSInformation();
            processor = new ProcessorInformation();
            disc = new DiskInformation();
            hdd = new HDDInformation();
            usage = new ProcessorUsage();
            service = new ServiceInformation();
            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 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 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!");
            }
        }


    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProcessesInformation
{
    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
= Math.Round(double.Parse(process.MaxWorkingSet.ToString()) / 1024 / 1024, 2).ToString()
+ "MB" });
            info.Add(new CurrentProcess() { Property = "Minimum working set size", Value
= Math.Round(double.Parse(process.MinWorkingSet.ToString()) / 1024 / 1024, 2).ToString()
+ "MB" });
            info.Add(new CurrentProcess() { Property = "Peak virtual memory size", Value
= Math.Round(double.Parse(process.PeakVirtualMemorySize.ToString()) / 1024 / 1024,
2).ToString() + "MB" });
            info.Add(new CurrentProcess() { Property = "Peak working set size", Value =
Math.Round(double.Parse(process.PeakWorkingSet.ToString()) / 1024 / 1024, 2).ToString() +
"MB" });
            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 =
Math.Round(double.Parse(process.WorkingSet.ToString()) / 1024 / 1024, 2).ToString() +
"MB" });
            info.Add(new CurrentProcess() { Property = "Main module", Value =
process.MainModule.ToString() });
            info.Add(new CurrentProcess() { Property = "Paged memory size", Value =
Math.Round(double.Parse(process.PagedMemorySize.ToString()) / 1024 / 1024, 2).ToString()
+ "MB" });
```

```
            info.Add(new CurrentProcess() { Property = "Paged system memory size", Value
= Math.Round(double.Parse(process.PagedSystemMemorySize.ToString()) / 1024 / 1024,
2).ToString() + "MB" });
            info.Add(new CurrentProcess() { Property = "User processor time", Value =
process.UserProcessorTime.ToString() });


            return info;
        }
        #endregion
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace ProcessesInformation
{
    public class DiskInformation
    {
        #region Fields
        private List<DiskInformation> info;
        #endregion

        #region Properties
        public string Capacity { get; private set; }
        public string DriveLetter { get; private set; }
        public string DriveType { get; private set; }
        public string FileSystem { get; private set; }
        public string FreeSpace { get; private set; }
        #endregion

        #region Methods
        public List<DiskInformation> GetDrivesInfo()
        {
            info = new List<DiskInformation>();
            var allDrives = DriveInfo.GetDrives();

            foreach(var d in allDrives)
            {
                if (!d.IsReady) continue;

                info.Add(new DiskInformation()
                {
                    DriveType = d.DriveType.ToString(),
                    DriveLetter = d.RootDirectory.ToString(),
                    Capacity = Math.Round(double.Parse(d.TotalSize.ToString()) / 1024 /
1024 / 1024, 2).ToString() + " GB",
                    FileSystem = d.DriveFormat.ToString(),
                    FreeSpace = Math.Round(double.Parse(d.TotalFreeSpace.ToString()) /
1024 / 1024 / 1024, 2).ToString() + " GB"
                });
            }
            return info;
        }
        #endregion
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Management;
using System.Text;
using System.Threading.Tasks;

namespace ProcessesInformation
{
    public class HDDInformation
    {
        private List<HDDInformation> info;

        public string NameProperty { get; private set; }
        public string Value { get; private set; }

        public List<HDDInformation> GetHDDInfo()
        {
            info = new List<HDDInformation>();

            foreach(var model in GetModelsInfo())
            {
                var query = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive
WHERE Model = '" + model + "'");

                foreach (ManagementObject obj in query.Get())
                {
                    AddInList("Model", obj);
                    AddInList("Partitions", obj);
                    AddInList("TotalCylinders", obj);
                    AddInList("TotalSectors", obj);
                    AddInList("TotalTracks", obj);
                    AddInList("TracksPerSylinder", obj);
                    AddInList("DeviceID", obj);
                    AddInList("InterfaceType", obj);
                    AddInList("Manufacturer", obj);
                    AddInList("SerialNumber", obj);
                    AddInList("Size", obj);
                    AddInList("BytesPerSector", obj);

                    info.Add(new HDDInformation() { NameProperty = "", Value = "" });
                }
            }

            return info;
        }

        private List<string> GetModelsInfo()
        {
            var query = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive");
            var models = new List<string>();

            foreach (ManagementObject obj in query.Get())
            {
                models.Add(obj["Model"].ToString());
            }

            return models;
        }

        private void AddInList(string value,  ManagementObject obj)
        {
            try
            {
                info.Add(new HDDInformation() { NameProperty = value, Value =
obj[value].ToString() });
```

```
            }
            catch { }
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Management;
using System.Text;
using System.Threading.Tasks;

namespace ProcessesInformation
{
    public class OSInformation
    {
        private List<OSInformation> info;

        public string NameProperty { get; private set; }
        public string Value { get; private set; }

        public List<OSInformation> GetOSInfo()
        {
            info = new List<OSInformation>();
            var query = new ManagementObjectSearcher("SELECT * FROM
Win32_OperatingSystem");

            foreach (ManagementObject obj in query.Get())
            {
                info.Add(new OSInformation() { NameProperty = "Build number", Value =
obj["BuildNumber"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Caption", Value =
obj["Caption"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Free physical memory",
Value = Math.Round(double.Parse(obj["FreePhysicalMemory"].ToString())/1024/1024,
2).ToString() + " MB"});
                info.Add(new OSInformation() { NameProperty = "Free virtual memory",
Value = Math.Round(double.Parse(obj["FreeVirtualMemory"].ToString())/1024/1024,
2).ToString() + " MB"});
                info.Add(new OSInformation() { NameProperty = "Name", Value =
obj["Name"].ToString() });
                info.Add(new OSInformation() { NameProperty = "OSType", Value =
obj["OSType"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Registered user", Value =
obj["RegisteredUser"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Serial number", Value =
obj["SerialNumber"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Service pack major
version", Value = obj["ServicePackMajorVersion"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Service pack minor
version", Value = obj["ServicePackMinorVersion"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Status", Value =
obj["Status"].ToString() });
                info.Add(new OSInformation() { NameProperty = "System device", Value =
obj["SystemDevice"].ToString() });
                info.Add(new OSInformation() { NameProperty = "System directory", Value =
obj["SystemDirectory"].ToString() });
                info.Add(new OSInformation() { NameProperty = "System drive", Value =
obj["SystemDrive"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Version", Value =
obj["Version"].ToString() });
                info.Add(new OSInformation() { NameProperty = "Windows directory", Value
= obj["WindowsDirectory"].ToString() });
            }
```

```
            return info;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace ProcessesInformation
{
    public class SystemProcess
    {
        #region Fields
        private List<SystemProcess> processes;
        #endregion

        #region Properties
        public string ProcessName { get; private set; }
        public int ProcessID { get; private set; }
        public int Threads { get; private set; }
        #endregion

        #region Methods
        public List<SystemProcess> GetProcesses()
        {
            processes = new List<SystemProcess>();
            foreach (var process in Process.GetProcesses())
            {
                processes.Add(new SystemProcess() { ProcessName = process.ProcessName,
ProcessID = process.Id, Threads = process.Threads.Count});
            }

            return processes;
        }

        public void KillProcessById(int id)
        {
            var process = Process.GetProcessById(id);

            try
            {
                process.Kill();
            }
            catch
            {
                throw new Exception("Kill exception!");
            }

        }
        #endregion
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Management;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ProcessesInformation
{
    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;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ProcessesInformation
{
    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();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceProcess;
using Microsoft.Win32;

namespace ProcessesInformation
{
    public class ServiceInformation
    {
        private Lazy<ServiceController[]> services = new Lazy<ServiceController[]>(() =>
{ return ServiceController.GetServices(); });
        private List<ServiceInformation> info;

        public string Name { get; private set; }
        public string Status { get; private set; }
        public string Description { get; private set; }
        public string ServiceName { get; private set; }
```

```
        public List<ServiceInformation> GetServiceInfo()
        {
            info = new List<ServiceInformation>();

            foreach (var serv in services.Value)
            {
                try
                {
                    var regKey =
Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\services\\" +
serv.ServiceName);

                    if ((regKey != null) && (regKey.GetValue("Description") != null))
                        info.Add(new ServiceInformation() { Name = serv.DisplayName,
Status = serv.Status.ToString(), Description = regKey.GetValue("Description").ToString(),
ServiceName = serv.ServiceName });
                    else
                        info.Add(new ServiceInformation() { Name = serv.DisplayName,
Status = serv.Status.ToString(), Description = String.Empty, ServiceName =
serv.ServiceName });

                    regKey.Close();
                }
                catch (Exception ex)
                {
                    throw;
                }
            }
            return info;
        }

        public void ChangeStatus(int index)
        {
            var item = services.Value[index];

            if (item.Status.Equals(ServiceControllerStatus.Stopped))
                item.Start();

            else
                item.Stop();
        }
    }
}


using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ProcessesInformation
{
    public class ThreadsInfo
    {
        #region Fields
        private Process process;
        private List<ThreadsInfo> threads;
        #endregion

        #region Properties
        public int ThreadID { get; private set; }
        public string ThreadState { get; private set; }
```

```
        public string Priority { get; private set; }
        #endregion

        public List<ThreadsInfo> GetThreadsInformationById(int id)
        {
            threads = new List<ThreadsInfo>();
            process = Process.GetProcessById(id);
            var th = process.Threads;

            for (var i = 0; i < process.Threads.Count; i++)
            {
                threads.Add(new ThreadsInfo() { ThreadID = th[0].Id, ThreadState =
th[0].ThreadState.ToString(), Priority = th[0].PriorityLevel.ToString() });
            }

            return threads;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProcessesInformation
{
    public class UserName
    {
        public static string GetUserName()
        {
            return System.Security.Principal.WindowsIdentity.GetCurrent().Name;
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Management;

namespace ProcessesInformation
{
    public class VideoCardInformation
    {
        private List<List<VideoCardInformation>> info;

        public string NameProperty { get; private set; }
        public string Value { get; private set; }

        public List<List<VideoCardInformation>> GetVideoCardInfo()
        {
            info = new List<List<VideoCardInformation>>();
            var query = new ManagementObjectSearcher("SELECT * FROM
Win32_VideoController");

            foreach (ManagementObject obj in query.Get())
            {
```

```
                info.Add(new List<VideoCardInformation> {
                    new VideoCardInformation() { NameProperty = "Adapter compatibility",
Value = obj["AdapterCompatibility"].ToString() },
                    new VideoCardInformation() { NameProperty = "Adapter DAC type", Value
= obj["AdapterDACType"].ToString() },
                    new VideoCardInformation() { NameProperty = "Adapter RAM", Value =
Math.Round(double.Parse(obj["AdapterRAM"]. ToString()) / 1024 / 1024, 2).ToString() + "
MB"},
                    new VideoCardInformation() { NameProperty = "Availability", Value =
obj["Availability"].ToString() },
                    new VideoCardInformation() { NameProperty = "Description", Value =
obj["Description"].ToString() },
                    new VideoCardInformation() { NameProperty = "DeviceID", Value =
obj["DeviceID"].ToString() },
                    new VideoCardInformation() { NameProperty = "Driver version", Value =
obj["DriverVersion"].ToString() },
                    new VideoCardInformation() { NameProperty = "Name", Value =
obj["Name"].ToString() },
                    new VideoCardInformation() { NameProperty = "VideoArchitecture",
Value = obj["VideoArchitecture"].ToString() },
                    new VideoCardInformation() { NameProperty = "VideoProcessor", Value =
obj["VideoProcessor"].ToString() },
                    new VideoCardInformation() { NameProperty = "", Value = ""} });


            }

            return info;
        }

    }
}
```