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

Анализ конфигурации и подключенных устройств, производительности подсистем компьютера, ограниченное управление устройствами

Студент:

гр. 953501

И.Д.Кондрашов

Руководитель:

Доцент, кандидат физико-математических наук С.И.Сиротко

Минск 2022

**Оглавление**

[Введение 2](#_Toc103336741)

[Определение основного функционала приложения 4](#_Toc103336742)

[Обзор используемых технологий 6](#_Toc103336743)

[Проектирование приложения 6](#_Toc103336744)

[Программная реализация 7](#_Toc103336745)

[Заключение 10](#_Toc103336746)

[Список использованных источников 13](#_Toc103336747)

[Приложение А - Текст программы 14](#_Toc103336749)

# 

# Введение

Основная идея данного курсового проекта – показать возможность взаимодействия с операционной системой (в данном случае OC 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 С#:

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 — С#, 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).

# Определение основного функционала приложения

В данном приложении была реализована возможность отслеживать текущее состояние компьютера по таким критерием, как нагрузка на процессор, загрузка оперативной памяти, список текущих процессов и возможность их останавливать, определение конфигурации некоторых подсистем.

# 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.

# 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.

# Программная реализация

Для реализации функционала взаимодействия с операционной системой была использована библиотека System.Diagnostics.

При запуске приложения операционная система создает для него отдельный процесс, которому выделяется определённое адресное пространство в памяти и который изолирован от других процессов. Процесс может иметь несколько потоков. Как минимум, процесс содержит один - главный поток. В приложении на C# точкой входа в программу является метод Main. Вызов этого метода автоматически создает главный поток. А из главного потока могут запускаться вторичные потоки.

В .NET процесс представлен классом Process из пространства имен System.Diagnostics. Этот класс позволяет управлять уже запущенными процессами, а также запускать новые. В данном классе определено ряд свойств и методов, позволяющих получать информацию о процессах и управлять ими:

Свойство Handle: возвращает дескриптор процесса

Свойство Id: получает уникальный идентификатор процесса в рамках текущего сеанса ОС

Свойство MachineName: возвращает имя компьютера, на котором запущен процесс

Свойство MainModule: представляет основной модуль - исполняемый файл программы, представлен объектом типа ProcessModule

Свойство Modules: получает доступ к коллекции ProcessModuleCollection, которая в виде объектов ProcessModule хранит набор модулей (например, файлов dll и exe), загруженных в рамках данного процесса

Свойство ProcessName: возвращает имя процесса, которое нередко совпадает с именем приложения

Свойство StartTime: возвращает время, когда процесс был запущен

Свойство PageMemorySize64: возвращает объем памяти, который выделен для данного процесса

Свойство VirtualMemorySize64: возвращает объем виртуальной памяти, который выделен для данного процесса

Метод CloseMainWindow(): закрывает окно процесса, который имеет графический интерфейс

Метод GetProcesses(): возвращающий массив всех запущенных процессов

Метод GetProcessesByName(): возвращает процессы по его имени. Так как можно запустить несколько копий одного приложения, то возвращает массив

Метод GetProcessById(): возвращает процесс по Id. Так как можно запустить несколько копий одного приложения, то возвращает массив

Метод Kill(): останавливает процесс

Метод Start(): запускает новый процесс

В данной программной реализации были использованы методы

GetProcesses, Kill и свойство PageMemorySize64 для сортировки процессов.

Text

Description automatically generated

Здесь можно увидеть метод, отвечающий за обработку нажатия кнопки отображения процессов. При этом вызывается вышеописанный метод GetProcesses и происходит заполнение словаря элементами, а также отображение имён процессов на форме.

При загрузке формы программным образом был изменен интервал таймера и инициализированы два счетчика для отображения нагрузки на память и процессор.

Text

Description automatically generated

При прохождении времени интервала вызывается событие Tick:

Text

Description automatically generated

Оно обновляет значения текста на форме в соответствии с текущими показателями счетчиков.

Это краткое описание программой реализации этого средства.

# Заключение

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

Привожу скриншоты работы приложения:

Graphical user interface, text, website

Description automatically generated

Начальное окно приложения с основными элементами приложения.

Кнопка “Показать процессы” выводит список процессов и подпроцессов объединенных по имени:

Text

Description automatically generated with medium confidence

При нажатии на любой элемент отображаются полные свойства процесса и появляются кнопки для управления им:

Graphical user interface, website

Description automatically generated

Кнопки “Показать нагруженность памяти” и “Показать нагруженность процессора” запускают вышеописанный таймер и начинается отображение информации о нагрузке на процессор и/или память.

Graphical user interface

Description automatically generated

По кнопке “Конфигурация” отображается конфигурации приведенных параметров(полученных средствами WinApi):

1. Конфигурация системы
2. Конфигурация логического диска
3. Конфигурация BIOS
4. Конфигурация физического накопителя
5. Конфигурация батареи компьютера
6. Конфигурация кэш-памяти
7. Конфигурация подключенных устройств

Graphical user interface, text, website

Description automatically generated

# 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

Метод инициализации формы:

public MainWindow()

{

InitializeComponent();

cpuCounter = new PerformanceCounter("Processor", "% Processor Time", "\_Total");

memoryCounter = new PerformanceCounter("Memory", "% Committed Bytes In Use");

memoryTimer.Interval = TimeSpan.FromSeconds(1);

memoryTimer.Tick += MemoryTimer\_Tick;

cpuTimer.Interval = TimeSpan.FromSeconds(1);

cpuTimer.Tick += CpuTimer\_Tick;

}

События, вызываемые таймером:

private void CpuTimer\_Tick(object sender, EventArgs e)

{

showCPUUsageButton\_Click(null, null);

}

private void MemoryTimer\_Tick(object sender, EventArgs e)

{

showMemoryUsageButton\_Click(null, null);

}

Метод, вызываемый кнопкой, отображающей процессы системы:\

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

{

var procList = new List<string>();

procList.Clear();

processes.Clear();

processesListBox.Items.Clear();

foreach (var winProc in Process.GetProcesses())

{

if (!processes.ContainsKey(winProc.ProcessName))

{

processes.Add(winProc.ProcessName, new List<Process>());

processes[winProc.ProcessName].Add(winProc);

}

else

{

processes[winProc.ProcessName].Add(winProc);

}

}

foreach (var processname in processes.Keys)

{

procList.Add(processname);

}

foreach (var item in procList)

{

processesListBox.Items.Add(item);

}

processesListBox.Visibility = Visibility.Visible;

}

Метод, отображающий нагруженность памяти:

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

{

memoryUsageTextBox.Visibility = Visibility.Visible;

memoryUsageLabel.Visibility = Visibility.Visible;

memoryUsageTextBox.Text = memoryCounter.NextValue().ToString() + " %";

memoryTimer.Start();

}

Метод, отображающий нагруженность процессора:

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

{

cpuUsageLabel.Visibility = Visibility.Visible;

cpuUsageTextBox.Visibility = Visibility.Visible;

cpuUsageTextBox.Text = cpuCounter.NextValue().ToString() + " %";

cpuTimer.Start();

}

Метод, отображающий подпроцессы выбранного процесса:

private void processesListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

MessageBox.Show(processesListBox.SelectedItem.ToString(), "Имя процесса", MessageBoxButton.OK, MessageBoxImage.Information);

subProcessesListBox.Items.Clear();

var myproc = processes[processesListBox.SelectedItem.ToString()];

foreach (var proc in myproc)

{

// 1MB = 1024KB = 1024 \* 1024

subProcessesListBox.Items.Add(string.Format($"Process: {proc.Id}, Memory: {proc.PagedMemorySize64 / 1048576} MB"));

}

subProcessesListBox.Visibility = Visibility.Visible;

killProcessesButton.Visibility = Visibility.Visible;

killSingleProcessButton.Visibility = Visibility.Visible;

}

Метод, завершающий все подпроцессы:

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

{

try

{

var myproc = processes[processesListBox.SelectedItem.ToString()];

foreach (var proc in myproc)

{

proc.Kill();

}

showProcessesButton\_Click(null, null);

MessageBox.Show("Процессы завершены", "Завершение", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Win32Exception ex)

{

MessageBox.Show($"{ex.Message}\nИзвините, это могут быть процессы Windows недоступные к завершению...", "Невозможность завершить процессы", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (Exception ex)

{

MessageBox.Show($"{ex.Message}\nИзвините, кажется, что-то пошло не по плану...", "Исключение", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

try

{

var myproc = processes[processesListBox.SelectedItem.ToString()];

if (subProcessesListBox.SelectedIndex != -1)

{

myproc[subProcessesListBox.SelectedIndex].Kill();

showProcessesButton\_Click(null, null);

}

else

{

MessageBox.Show("Выберите процесс!", "Процесс не выбран", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Win32Exception ex)

{

MessageBox.Show($"{ex.Message}\nИзвините, это может быть процесс Windows недоступный к завершению...", "Невозможность завершить процесс", MessageBoxButton.OK, MessageBoxImage.Error);

}

catch (Exception ex)

{

MessageBox.Show($"{ex.Message}\nИзвините, кажется, что-то пошло не по плану...", "Исключение", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

string computer = @"root\cimv2";

string partOSQuery = "Win32\_OperatingSystem";

ConnectionOptions options = new ConnectionOptions();

options.Impersonation = ImpersonationLevel.Impersonate;

ManagementScope osScope = new ManagementScope(computer, options);

ObjectQuery osQuery = new ObjectQuery($"SELECT \* FROM {partOSQuery}");

ManagementObjectSearcher osSearcher = new ManagementObjectSearcher(osScope, osQuery);

ManagementObjectSearcher logicalDisk = new ManagementObjectSearcher("SELECT \* FROM Win32\_LogicalDisk");

ManagementObjectSearcher bios = new ManagementObjectSearcher("SELECT \* FROM Win32\_BIOS");

ManagementObjectSearcher disk = new ManagementObjectSearcher("SELECT \* FROM Win32\_DiskDrive");

ManagementObjectSearcher battery = new ManagementObjectSearcher("SELECT \* FROM Win32\_Battery");

ManagementObjectSearcher cacheMemory = new ManagementObjectSearcher("SELECT \* FROM Win32\_CacheMemory");

ManagementObjectSearcher usbHubs = new ManagementObjectSearcher("SELECT \* FROM Win32\_USBHub");

foreach (var osObj in osSearcher.Get())

{

configListBox.Items.Add("Конфигурация системы:");

configListBox.Items.Add(string.Format($"Имя компьютера : {osObj["csname"]}"));

configListBox.Items.Add(string.Format($"Директория Windows : {osObj["WindowsDirectory"]}"));

configListBox.Items.Add(string.Format($"Операционная система : {osObj["Caption"]}"));

configListBox.Items.Add(string.Format($"Версия : {osObj["Version"]}"));

configListBox.Items.Add(string.Format($"Производитель : {osObj["Manufacturer"]}\n"));

}

foreach (var o in logicalDisk.Get())

{

configListBox.Items.Add("Конфигурация логического диска:");

configListBox.Items.Add(string.Format($"Диск : {o["DeviceID"]}"));

configListBox.Items.Add(string.Format($"Размер диска : {Convert.ToInt64(o["Size"].ToString()) / 1048576} MB\n"));

}

foreach (var b in bios.Get())

{

configListBox.Items.Add("Конфигурация BIOS:");

configListBox.Items.Add(string.Format($"Версия BIOS : {b["SoftwareElementID"]}"));

configListBox.Items.Add(string.Format($"Статус BIOS : {b["Status"]}\n"));

}

foreach (var dis in disk.Get())

{

configListBox.Items.Add("Конфигурация физического накопителя:");

configListBox.Items.Add(string.Format($"Описание : {dis["Description"]}"));

configListBox.Items.Add(string.Format($"ID устройства : {dis["DeviceID"]}"));

configListBox.Items.Add(string.Format($"Размер : {Convert.ToInt64(dis["Size"]) / 1048576} MB\n"));

}

foreach (var bat in battery.Get())

{

configListBox.Items.Add("Конфигурация батареи компьютера:");

configListBox.Items.Add(string.Format($"Описание : {bat["Description"]}"));

configListBox.Items.Add(string.Format($"ID устройства : {bat["DeviceID"]}\n"));

}

foreach (var cache in cacheMemory.Get())

{

configListBox.Items.Add($"Конфигурация кэш-памяти:");

configListBox.Items.Add(string.Format($"Уровень : {cache["Level"]}"));

configListBox.Items.Add(string.Format($"ID устройства : {cache["DeviceID"]}"));

configListBox.Items.Add(string.Format($"Статус : {cache["StatusInfo"]}"));

configListBox.Items.Add(string.Format($"Размер : {cache["BlockSize"]} Bytes"));

configListBox.Items.Add(string.Format($"Цель : {cache["Purpose"]}"));

configListBox.Items.Add(string.Format($"Тип кэша : {cache["CacheType"]}\n"));

}

foreach (var usb in usbHubs.Get())

{

configListBox.Items.Add($"Конфигурация подключенных устройств:");

configListBox.Items.Add(string.Format($"Название подключенного устройства : {usb["Name"]}"));

configListBox.Items.Add(string.Format($"ID подключенного устройства : {usb["DeviceID"]}"));

configListBox.Items.Add(string.Format($"Имя системы : {usb["SystemName"]}"));

}

}