**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Избранные главы информатики»

на тему: «Новостной портал»

Исполнитель: студент гр. ИП-31

В.А. Дранев

Руководитель: старший преподаватель

Т.Л. Романькова

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2020

Содержание

[Введение 5](#_Toc58365008)

[1 Теоретические сведения 6](#_Toc58365009)

[1.1 Обзор существующих программных продуктов 6](#_Toc58365010)

[1.2 Постановка задачи 10](#_Toc58365011)

[1.3 Язык и среда программирования 11](#_Toc58365012)

[1.4 Основы графических интерфейсов 12](#_Toc58365013)

[2 Алгоритмический анализ 17](#_Toc58365014)

[2.1 Анализ задания на проектирование 17](#_Toc58365015)

[3 Разработка программного кода 26](#_Toc58365016)

[3.1 Интерфейс пользователя 26](#_Toc58365017)

[3.2 Используемые функции 32](#_Toc58365018)

[4 Тестирование 35](#_Toc58365019)

[Заключение 39](#_Toc58365020)

[Список использованных источников 40](#_Toc58365021)

[Приложение А 41](#_Toc58365022)

# **ВВЕДЕНИЕ**

В любом электронном устройстве с операционной системой есть специальная программа, называемая диспетчером задач. Ей активно пользуются опытные пользователи компьютера, а начинающим нужно потратить несколько минут на изучение ее предназначения и возможностей.

Диспетчер задач - это специальная программа, показывающая различную информацию о работе системы и позволяющая частично управлять ею. В нее входит запущенные программы, процессы и службы, загрузка процессора и оперативной памяти, состояние и загрузку сетевых подключений, а также, подключенных пользователей.

[Диспетчер задач](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%82%D1%87%D0%B5%D1%80_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87), системный монитор и менеджер запуска, входящий в состав [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows). Он предоставляет информацию о производительности компьютера и использовании [центрального](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) процессора, фиксирует нагрузку и сведения о памяти, сетевой активности и статистике, системных службах. Диспетчер задач также может использоваться для установки приоритетов процессов, свойства процессора, запуска и остановки служб и принудительного завершения процессов.

Возможности стандартного диспетчера задач в Windows 7/10 ограничены, а в некоторых случаях, и вовсе не позволяют решить положенную задачу. Вот краткий перечень того, чем стандартный диспетчер задач:

* в нём нельзя посмотреть иерархию запущенных процессов, их связь

между собой;

* нет информации о процессах;
* нет никакой информации об открытых дескрипторах и загруженных

библиотеках DLL;

* нельзя завершить (прикончить) некоторые зависшие процессы,

особенно системные;

* блокируется вирусами.

В некоторых операционных системах диспетчер задач предоставляет специфическую информацию о системе, процессах и потоках, и наоборот, некоторые диспетчеры задач дредоставляют если не малый набор данных, то не то данные не нужные конкретному пользователю.

Цель проекта - создать приложение, позволяющее просматривать информацию о потоках и процессах в системе и управлять ими. И постараться решить проблемы стандартного диспетчера задач Windows.

Приложение должно собирать информацию в системе о процессах и потоках, выводить данную информацию на окно, для просмотра пользователем. Приложение должно позволять управлять процессами и потоками, а именно, повышение их приоритета и закрытие потоков и процессов.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

## Обзор существующих программных продуктов

В настоящее время существуют разные решения подобной задачи. Все они различаются графическим оформлением, функционалом, возможностями в управлении операционной системой, процессами и потоками. Но суть всех этих приложений всегда остается одной – помощь пользователю в управлении системой.

Общими чертами всех диспетчеров задач, начиная от стандартных и заканчивая диспетчеров, которые были написаны самими пользователями, являются:

1. управление процессами:
2. открытие;
3. завершение;
4. просмотр информации;
5. повышение приоритета;
6. понижение приоритета;
7. просмотр информации о системе:
8. нагрузка на центральный процессор;
9. информация о центральном процессоре;
10. информация о нагруженности памяти;
11. информация о нагруженности диска;
12. информация о нагрузке на сеть;
13. управление автозагрузкой приложений.

Согласно [4] одним из готовых решений является Process Explorer от компании Sysinternals. изображенная на рисунке 1.

Компания Microsoft уже давно сотрудничает с Sysinternals и даже перекупила себе некоторых программистов. Утилиты этой компании Microsoft часто приводит на своём официальном сайте как вспомогательные для решения проблем с Windows. И именно Process Explorer рекомендуется в Microsoft как альтернатива диспетчеру задач.

Это бесплатная, лёгкая и не требующая установки программа. Достаточно загрузить утилиту по прямой ссылке, распаковать и запустить исполняемый файл «procexp.exe».

Process Explorer может делать:

* показывает всю необходимую информацию о ВСЕХ запущенных

процессах и используемых ресурсах по каждому из них;

* показывает подробную информацию по использованию ресурсов

системы в целом (оперативная память, процессор, диски, а также по GPU (видеокарта));

* располагает информацией о том, какие DLL используются каждым

процессом и с какими ключами они запускались;

* имеет хорошую систему поиска по всей доступной информации;
* есть поиск в интернете по выбранным подозрительным процессам на

предмет вирусности;

* умеет [закрывать](https://it-like.ru/kak-zakryit-programmu/) любые процессы.

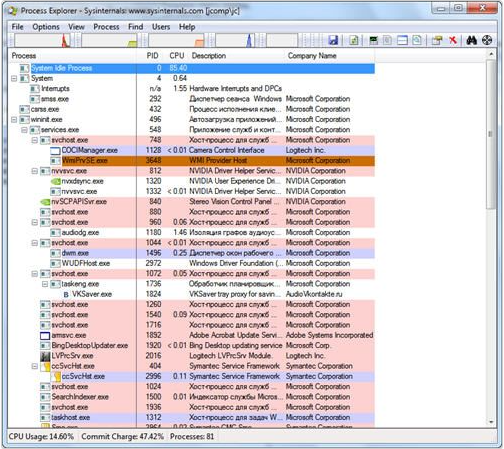


Рисунок 1 – Окно приложения Process Explorer

Еще одно приложение это System Explorer, изображенная на рисунке 2.

Программа справляется со всеми функциями встроенного диспетчера, плюс имеет ряд преимуществ.  самой программе всегда можно проверить отдельный процесс на принадлежность к вирусам, и вообще разработчики уделили особое внимание безопасности. System Explorer отображает информативное дерево, не перегруженное лишней информацией. Рядом с каждым процессом есть кнопка «Подробности».

Помимо стандартных функций, таких как контролирование запущенных процессов и ресурсов компьютера, System Explorer умеет:

* [управлять автозагрузкой](https://it-like.ru/kak-otklyuchit-avtozapusk-programm-windows-7/);
* удалять программы;
* отслеживать процессы с ключами запуска;
* обнаруживать драйвера, системные службы и модули программ;
* можно просмотреть информацию о запускаемом файле ещё до его

запуска, просто перетянув файл в специальное окно программы;

* встраивается в систему и заменяет стандартный диспетчер задач;
* интегрирован с социальными сетями;
* имеет встроенный фильтр для более наглядного отображения

информации;

* возможность перейти к файлу процесса прямо из программы;

Очень интересная возможность System Explorer делать снимки системы (файлы, реестр и файлы+реестр) и потом их сравнивать. Это очень удобно для отслеживания изменений в системе после действий какой-нибудь программы.

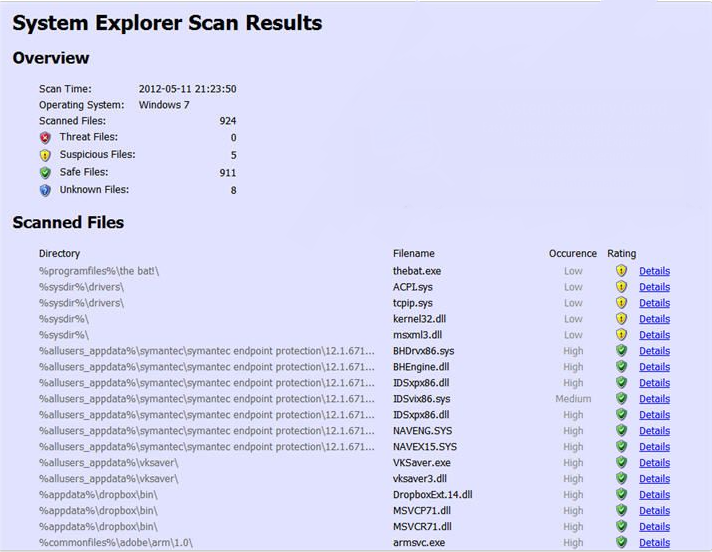


Рисунок 2 – Окно приложения System Explorer

Еще одним из готовых решений является Free Extended Task Manager, изображенная на рисунке 3.

Эта программа наглядно отображает информацию о запущенных программах и процессах, а также о драгоценных ресурсах компьютера, которые непонятно куда деваются. Очень похож на стандартный диспетчер программ.

Какие есть достоинства у Free Extended Task Manager:

* отображение дисковой активности по каждому приложению;
* отображение сетевой активности отдельно по каждому IP-адресу;
* отображение информации по приложениям, которые наиболее всего

используют процессорное время и оперативную память в графической форме;

* показывает все заблокированные (используемые) файлы по каждому

процессу;

* находит все процессы, заблокировавшие определённый файл;
* есть вкладка «Общее» («Summary»), для быстрого доступа к

информации об общем состоянии системы;

* позволяет ставить на паузу («замораживать») отдельно взятые

процессы;

* другие визуальные и функциональные улучшения по сравнению со

стандартным диспетчером задач.

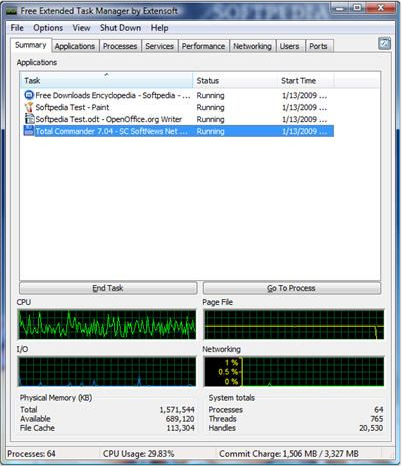


Рисунок 3 – Окно приложения Free Extended Task Manager

## Постановка задачи

[Операционная система](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) – комплекс [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), обеспечивающий управление аппаратными средствами [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), организующий работу с [файлами](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB) и выполнение [прикладных программ](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), осуществляющий ввод и вывод данных.

На сегодняшний день, операционная система — это первый и основной набор программ, загружающийся в компьютер. Помимо вышеуказанных функций ОС может осуществлять и другие, например, предоставление общего пользовательского интерфейса.

Поэтому важно иметь возможность просматривать информацию осистеме и иметь корректировать ее работу или управлять ею.

Целью курсовой работы является спроектировать приложение, которое собирало бы информацию о системе, процесах и потоках и предоставляла ее в структурированном виде для пользователя. Зная это информацию, пользователь должен иметь возможность управлять системой и процессами.

Виды собираемой информации:

1. количество оперативной памяти, которое использует процесс;
2. список используемых модулей;
3. текущий приоритет процесса;
4. время запуска процесса;
5. время выполнения процесса;
6. название окна исполняемого файла процесса;
7. количество свободной оперативной памяти в системе;
8. количество потоков в каждом процессе.

Приложение должно иметь возможность просматривать запущенные процессы и потоки, позволять завершать процессы, запускать их и повышать или понижать приоритет.

Выводить информацию о загруженности памяти, подробную информацию о процессах, а именно время запуска, расположение в системе исполняемого файла.

Приложение должно функционировать на различных операционных системах таких как:

* Windows;
* Linux;
* Mac OS.

Для большего распространения среди пользователей.

Интерфейс должен быть интуитивно понятен для любых пользователей, для исключения дальнейших проблем при использовании приложения. Также допускается использование системных возможностей, такие как проводник для открытия процессов и просмотра исполняемых файлов процессов.

## Язык и среда программирования

Согласно [3] язык C# и связанную с ним среду [.NET Framework](https://professorweb.ru/my/csharp/charp_theory/level1/1_3.php) можно без преувеличения назвать самой значительной из предлагаемых в настоящее время технологий для разработчиков.

Среда .NET является такой средой, которая была создана для того, чтобы в ней можно было разрабатывать практически любое приложение для запуска в Windows, а C# является языком программирования, который был специально создан для использования в .NET Framework.

Например, с применением C# и .NET Framework можно создавать динамические веб-страницы, приложения Windows Presentation Foundation, веб-службы XML, компоненты для распределенных приложений, компоненты для доступа к базам данных, классические настольные приложения Windows и даже клиентские приложения нового интеллектуального типа, обладающие возможностями для работы в оперативном и автономном режимах.

C# является объектно-ориентированным языком. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности.

Главная особенность таких компонентов в том, что они представляют собой модель программирования со свойствами, методами и событиями. У них есть атрибуты, предоставляющие декларативные сведения о компоненте. Они включают в себя собственную документацию.

C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# подходит для создания и применения программных компонентов.

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

1. Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.
2. Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Последняя версия платформы на данный момент .NET Core поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.
3. Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов.

## Основы графических интерфейсов

Согласно [5] **Windows Forms** — [интерфейс программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) (API), отвечающий за [графический интерфейс пользователя](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) и являющийся частью [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework).

Данный интерфейс упрощает доступ к элементам интерфейса [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) за счет создания обёртки для существующего [Win32 API](https://ru.wikipedia.org/wiki/Windows_API) в [управляемом коде](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Причём управляемый код — классы, реализующие [API](https://ru.wikipedia.org/wiki/API) для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке [MFC](https://ru.wikipedia.org/wiki/Microsoft_Foundation_Classes), изначально написанной на языке [C++](https://ru.wikipedia.org/wiki/C%2B%2B). С другой стороны, WF не предлагает парадигму, сравнимую с [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки.

Windows Forms включает широкий набор элементов управления, которые можно добавлять на формы: текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы.

Список всех элементов управления, которые можно использовать в форме, представлены в разделе [Элементы управления для использования в формах Windows Forms](https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/controls/controls-to-use-on-windows-forms). Если существующий элемент управления не удовлетворяет потребностям, в Windows Forms можно создать пользовательские элементы управления с помощью класса [UserControl](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.usercontrol).

В состав Windows Forms входят многофункциональные элементы пользовательского интерфейса, позволяющие воссоздавать возможности таких сложных приложений, как Microsoft Office.

Используя элементы управления [ToolStrip](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.toolstrip) и [MenuStrip](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.menustrip), можно создавать панели инструментов и меню, содержащие текст и рисунки, подменю и другие элементы управления, такие как текстовые поля и поля со списками.

Во многих приложениях нужно отображать данные из базы данных, XML-файла, веб-службы XML или другого источника данных. Windows Forms предоставляет гибкий элемент управления с именем [DataGridView](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datagridview) для отображения таких табличных данных в традиционном формате строк и столбцов так, что каждый фрагмент данных занимает свою собственную ячейку.

С помощью [DataGridView](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datagridview) можно, помимо прочего, настроить внешний вид отдельных ячеек, зафиксировать строки и столбцы на своем месте, а также обеспечить отображение сложных элементов управления внутри ячеек.

При использовании интеллектуальных клиентов Windows Forms можно легко подключаться к источникам данных по сети. Компонент [BindingSource](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingsource) представляет соединение с источником данных и предоставляет методы для привязки данных к элементам управления, перехода к предыдущим и следующим записям, изменения записей и сохранения изменений в исходном источнике. Элемент управления [BindingNavigator](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingnavigator) предоставляет простой интерфейс на основе компонента [BindingSource](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingsource) для перехода между записями.

Вы можете легко создавать элементы управления с привязкой к данным с помощью окна "Источники данных". В нем приводятся имеющиеся в проекте источники данных, такие как базы данных, веб-службы и объекты.

Создавать элементы управления с привязкой к данным можно путем перетаскивания объектов из этого окна в формы проекта. Также можно связывать существующие элементы управления с данными, перетаскивая объекты из окна "Источники данных" в существующие элементы управления. (Согласно [5])

Согласно [6] **Технология WPF** (Windows Presentation Foundation) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML. XAML (Extensible Application Markup Language — расширяемый язык разметки приложений) представляет собой язык разметки, используемый для создания экземпляров объектов .NET. Хотя язык XAML — это технология, которая может быть применима ко многим различным предметным областям, его главное назначение — конструирование пользовательских интерфейсов WPF.

Согласно [2] WPF позволяет разрабатывать приложения, используя как разметку, так и код программной части, что привычно для разработчиков на ASP.NET. Разметка XAML обычно используется для определения внешнего вида приложения, а управляемые языки программирования (код программной части) — для реализации его поведения. Такое разделение внешнего вида и поведения имеет ряд преимуществ.

1. Затраты на разработку и обслуживание снижаются, так как разметка, определяющая внешний вид, не связана тесно с кодом, обуславливающим поведение.
2. Повышается эффективность разработки, так как дизайнеры, занимающиеся внешним видом приложения, могут работать параллельно с разработчиками, реализующими поведение приложения.

Платформа WPF предоставляет широкий, гибкий и масштабируемый набор графических функций, который обладает перечисленными ниже преимуществами.

1. независимость графики от разрешения и устройства. Основной единицей измерения в графической системе WPF является аппаратно-независимый пиксель, размер которого составляет 1/96 дюйма вне зависимости от разрешения экрана. Это создает основу для, независимой от разрешения и аппаратной платформы, отрисовки. Каждый аппаратно-независимый пиксель автоматически масштабируется в соответствии с заданным в системе количеством точек на дюйм (DPI);
2. повышение точности. Система координат WPF основана на числах двойной точности с плавающей запятой, а не числах одинарной точности. Значения преобразования и прозрачности также выражаются числами двойной точности. Платформа WPF также поддерживает широкую цветовую палитру (scRGB) и имеет встроенную поддержку управления входными данными из разных цветовых схем;
3. аппаратное ускорение. Система графики WPF использует возможности графического оборудования, чтобы снизить нагрузку на центральный процессор.

Согласно [1] платформа **ASP.NET Core** представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, ASP.NET Core является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход ASP.NET Core фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась ASP.NET vNext. В июне 2016 года вышел первый релиз платформы. А в декабре 2019 года вышла версия ASP.NET Core 3.1, которая собственно и будет охвачена в текущем руководстве.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на [GitHub](https://github.com/aspnet/).

ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux.

И таким образом, с помощью ASP.NET Core мы можем создавать кросс-платформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер Kestrel.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку System.Web.dll.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом С#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры.

В качестве инструментария разработки мы можем использовать последние выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, мы можем создавать приложения в среде Visual Studio Code, которая является кросс-платформенной и может работать как на Windows, так и на Mac OS X и Linux.

Для обработки запросов теперь используется новый конвейер HTTP, который основан на компонентах Katana и спецификации OWIN. А его модульность позволяет легко добавить свои собственные компоненты.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

* новый легковесный и модульный конвейер HTTP-запросов;
* возможность развертывать приложение как на IIS, так и в рамках своего

собственного процесса;

* использование платформы .NET Core и ее функциональности;
* распространение пакетов платформы через NuGet;
* интегрированная поддержка для создания и использования пакетов

NuGet;

* единый стек веб-разработки, сочетающий Web UI и Web API;
* конфигурация для упрощенного использования в облаке;
* встроенная поддержка для внедрения зависимостей;
* расширяемость;
* кроссплатформенность: возможность разработки и развертывания

приложений ASP.NET на Windows, Mac и Linux;

* развитие как open source, открытость к изменениям.

Эти и другие особенности и возможности стали основой для новой модели программирования.

Хоть платформа ASP.NET Core и обеспечивает кроссплатформенность, данная технология не подходит для реализации проекта в силу требований задания. Так как программа должна быть установленна на компьютере и собирать сведения о нем. Поэтому создавать браузерное решение поставленной задачи не целесообразно.

В результате, язык C# и принципы объектно - ориентированного программирования подходят для реализации программы, а технология WPF не только проста в использовании, но и обеспечит кроссплатформенность, что позволит запускать программу на практически любом компьютере.

1. **АЛГОРИТМИЧЕСКИЙ АНАЛИЗ**

## Анализ задания на проектирование

На рисунке 4 графически показаны возможности пользователя в созданном приложении по управлению процессами, среди них:

1. понижение приоритета;
2. просмотр используемых модулей;
3. просмотр адреса исполняемого файла;
4. повышение приоритета;
5. просмотр списков процессов;
6. добавление процесса;
7. завершение процесса.

Данные функции доступны для любых пользователей в системе с любым уровнем системног доступа.

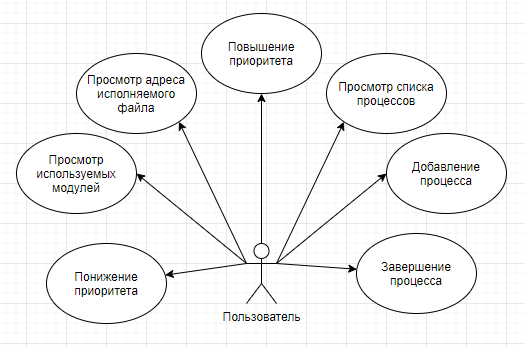


Рисунок 4 – Возможности управления процессами для пользователя

Собираемая приложением информация о системе, согласно рисунку 5, полезна для пользователя и может показывать, как состояние отдельного процесса в системе, так и состояние всей системы в целом.

Информация о процессе и информация о системе, сязанная с этим процессом, может быть не разграничена по разным областям окна и находиться рядом. Это может быть удобно для просмотра основних параметров системы и процесса.

Также, приоритеты процессов могут быть расположены ввиде списка для удобства их выбора. Добавление можно реализовать в виде выбора исполняемого файла через проводник, для удобства пользователя.

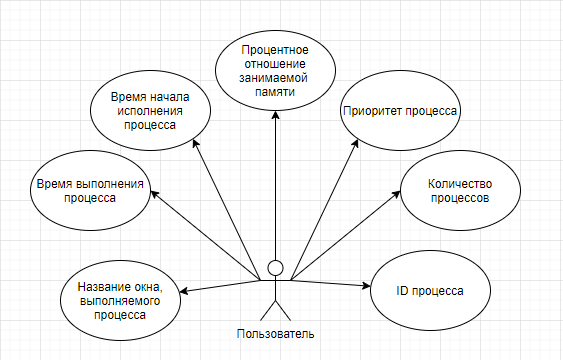


Рисунок 5 – Доступная пользователю информация о системе

***2.1.1*** Для просмотра информации о процессе, согласно рисунку 6, пользователь должен выбрать любой процесс из списка доступных, в левой части окна. После выбора процесса информация о нем доступна для просмотра в правой части окна. Для просмотра доступна следующая информация:

1. название окна;
2. время исполнения запущенного процесса;
3. время запуска процесса;
4. приоритет процесса;
5. модули, которые использует процесс.

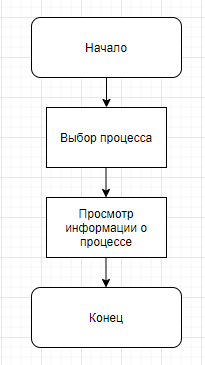


Рисунок 6 – Алгоритм просмотра информации о процессе

***2.1.2*** Повышение приоритета процесса, согласно рисунку 7, осуществляется с помощью выбора процесса из списка доступных в левой чсти окна. После выбора процесса в правой части окна устанавливается его текущий приоритет.

Далее, из списка доступных вариантов приоритета процессов, выбирается один из следующих приоритетов:

1. низкий (Low);
2. ниже среднего (Below Normal);
3. средний (Normal);
4. выше среднего (Above Normal);
5. высокий (Hight);
6. приоритет реального времени (Real Time).

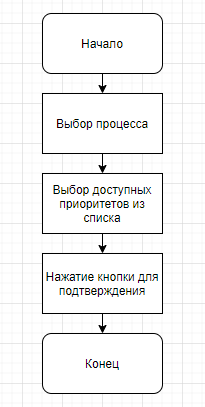


Рисунок 7 – Алгоритм повышения приоритета процесса

***2.1.3*** Понижение приоритета, алгоритм которого изображен на рисунке 8, позволяет пользователю ослабить нагрузку на систему, путем уменьшения процессорного времени которое обслуживает даный процесс.

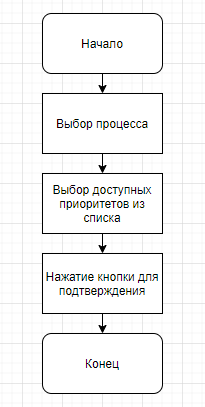


Рисунок 8 – Алгоритм понижения приоритета процесса

***2.1.4*** Для просмотра модулей, которые использует процесс, согласно рисунку 9, пользователь должен выбрать процесс из списка доступных в левой касти окна, после чего программа выведет весь список используемых модулей в равой части окна в специальном списке, для удобного просмотра пользователем.

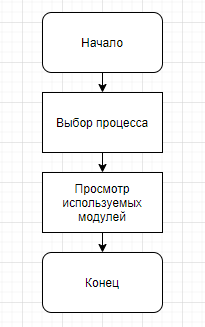


Рисунок 9 – Алгоритм просмотра используемых модулей

***2.1.5*** Для просмотра размещения исполняемого файла, в списке доступных процессов в левой части окна, согласно рисунку 10, пользователь должен выбрать процесс. Далее пользователь должен нажать правую кнопку мыши для вызова контекстного меню, в котором присутствует пункт «Show folder».

После выбора данного пункта меню должен открыться проводник с расположением исполняемого файла, сам исполняемый файл должен быть выделен, для удобства.

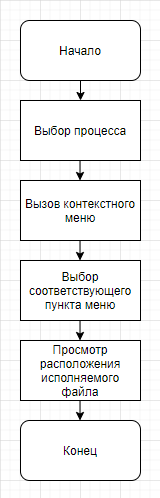


Рисунок 10 – Алгоритм просмотра расположения исполняемого файла

***2.1.6*** Добавление – важный пункт и функционал приложения. Возможность запускать процессы из приложения позволяет запустить и просмотреть информацию об этом процессе и увидеть, как влияет тот или иной процесс на систему и как процессы взаимодействуют с собой.

Для добавления/вызова нового процесса, согласно рисунку 11, пользователь должен нажать на кнопку добавления нового процесса с названием «Add». После нажатия пользователем кнопки вызывается проводник для выбора расположения исполняемого файла, который будет запущен.



Рисунок 11 – Алгоритм запуска нового процесса

После добавления, процесс появляется в списке доступных процессов, в левой части окна. После этого можно просматривать всю возможную информацию об этом процессе.

***2.1.7*** Удаление – не маловажная возможность приложения, алгоритм которого изображен на рисунке 12. С помощью этой функции можно завершать процессы, которые пользователь не может завершить самостоятельно, использовав интерфейс самого процесса, если такой имеется.

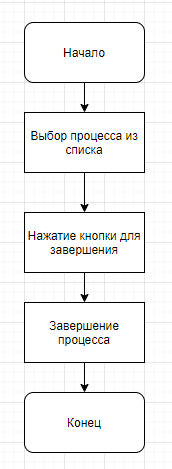


Рисунок 12 – Алгоритм удаления процесса

1. **РАЗРАБОТКА ПРОГРАММНОГО КОДА**

Программа включает в себя одно окно, содержащее информационные панели и кнопки, с помощью которых осуществляется управление процессами и потоками, согласно рисунку 13.



Рисунок 13 - Структурная схема программы

## Интерфейс пользователя

Пользовательский интерфейс - это совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги.

Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи. Каждый диалог состоит из отдельных процессов ввода / вывода, которые физически обеспечивают связь пользователя и компьютера.

Окно MainWindow

Основное окно приложения, изображенное нарисунке 14. В этом окне располагаются информационные панели, список процессов, список используемых модулей, кнопки открытия и закрытия процесса, пункт повышения и понижения приоритета процесса.

Окно содержат обработку исключительных ситуаций, таких как: повышение приоритета не выбранного процесса, просмотр модулей не выбранного процесса, просмотр информации о не выбранном процессе.

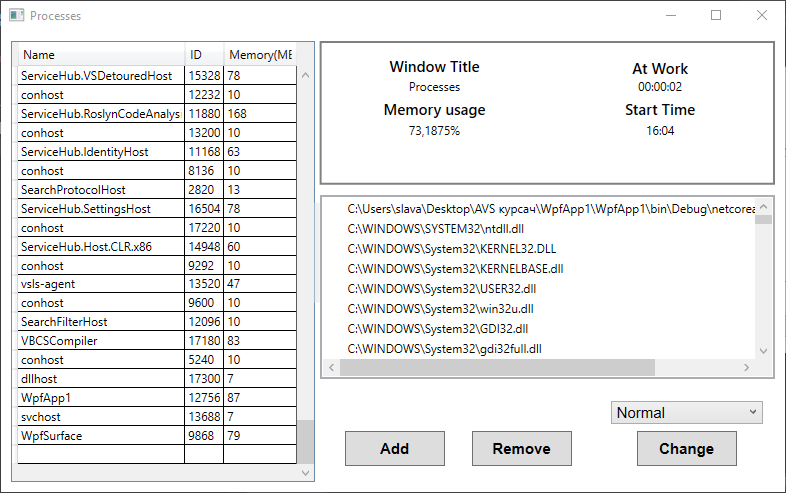


Рисунок 14 – Основное окно программы

Добавление/Запуск процесса.

Запуск нового процесса выполняется с помощью нажатия кнопки «Add». После нажатия появляется проводник операционной системы, в котором выбирается путь до исполняемого файла. После выбора исполняемого файла запускаемого процесса, следует нажать кнопку «Open» для запуска процесса, согласно рисунку 15.

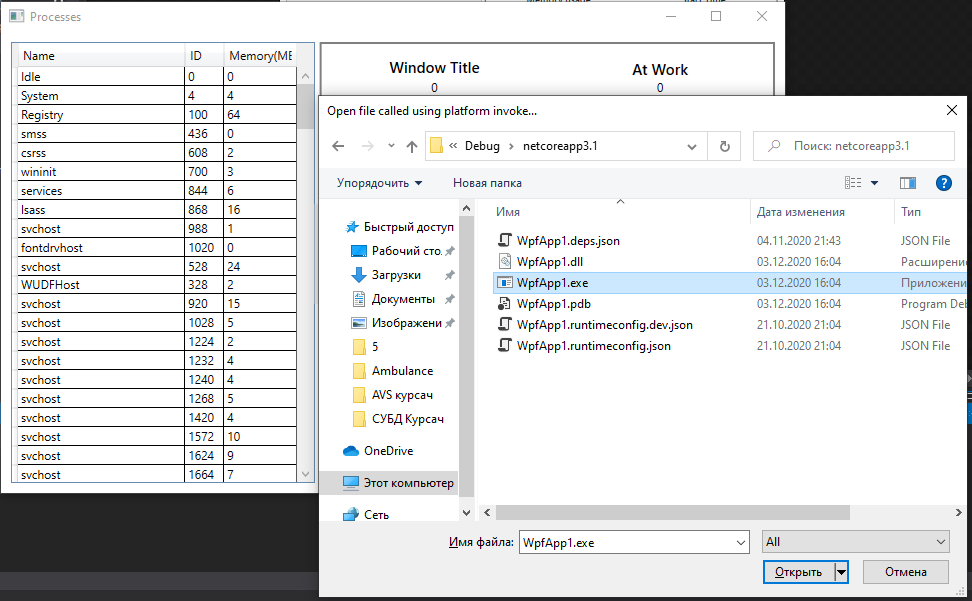


Рисунок 15 – Основное окно

Изменение приоритета процесса.

Для изменения приоритета процесса, пользователь должен выбрать процесс из списка доступных, в левой части экрана, изображенного на рисунке 16. После выбора, вся информация о процессе выводится в информационную панель в верхней части экрана, а установленный приоритет выделяется зи списка доступных.

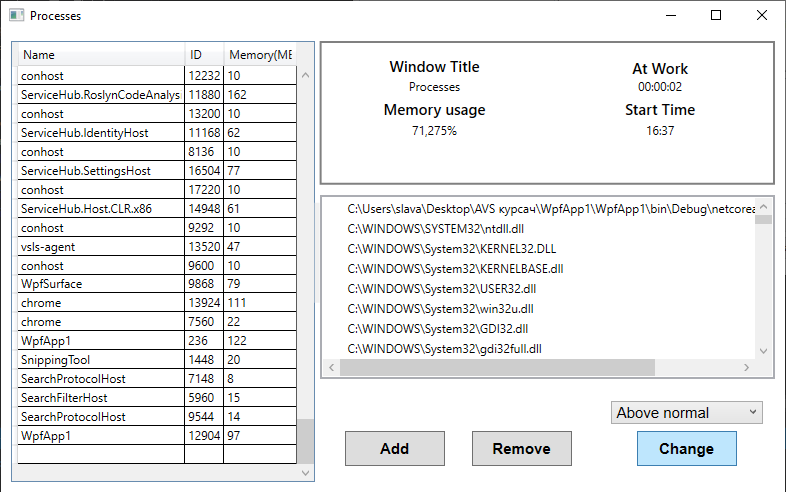


Рисунок 16 – Выбор процесса для изменения приоритета

По желанию пользователя, приоритет может быть изменен, для этого нужно выбрать один из доступних приоритетов и списка и нажать кнопку «Change», согласно рисунку 17. После этого смена приоритета процесса закончена.

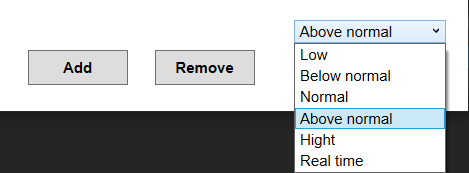


Рисунок 17 – Список доступных приоритетов процесса

Просмотр расположения исполняемого файла процесса

Для просмотра исполняемого файла процесса, пользователь должен выбрать проццесс из списка доступных в левой части окна, изображенный на рисунке 18. После выбора нужно нажать правую кнопку миши, там самым вызвав контекстное меню с пунктом «Show folder».

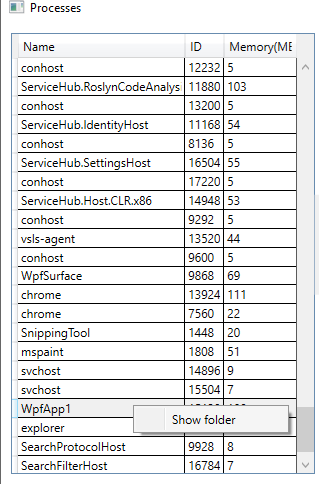


Рисунок 18 – Вызов контекстного меню

После выбора данного пункта меню, открывается проводник системы в котором уже указан путь до файла и сам файл выделен, для удобства пользователя, согласно рисунку 19.

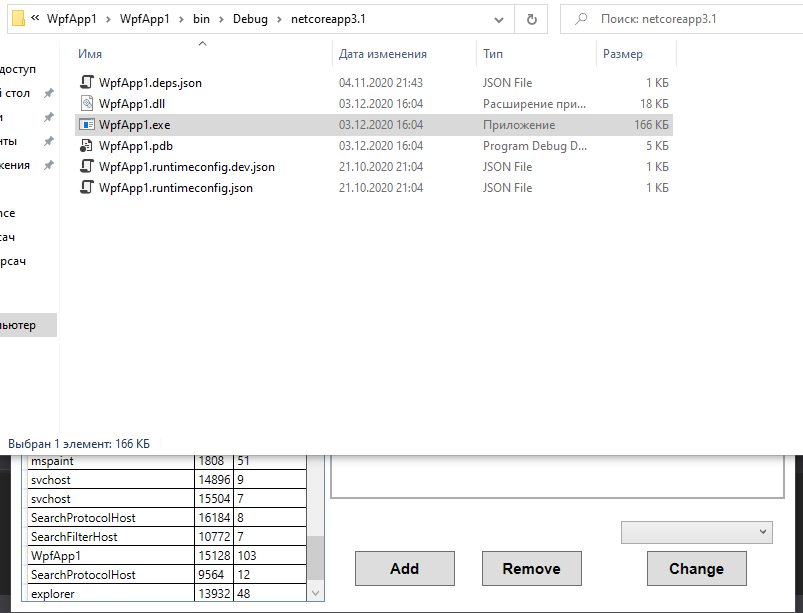


Рисунок 19 – Расположение исполняемого файла

Просмотр модулей, которые использует процесс

Для просмотра модулей, которые использует процесс, пользователь должен выбрать процесс из списка доступных в левой части окна, согласно рисунку 20. После выбора, список модулей можно просмотрет в правой части окна в специальном ListBox’е.

Данный ListBox предназначен для удобного просмотра списка модулей и из расположений на компьютере, которые использует данный процесс.

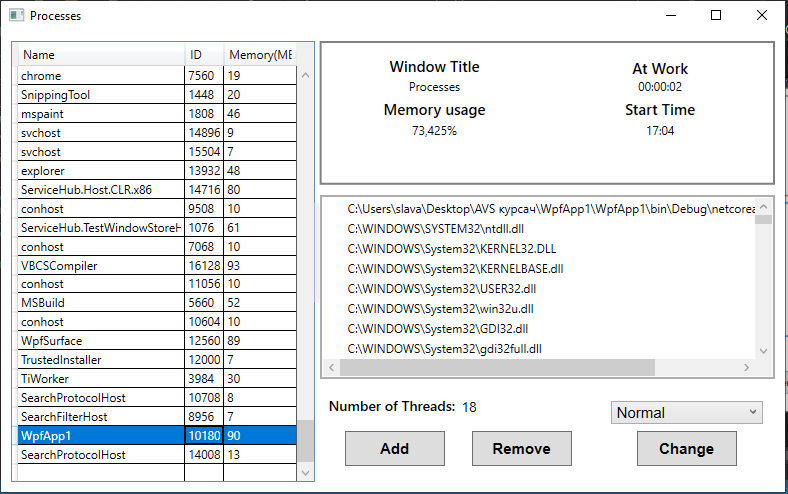


Рисунок 20 – Список используемых модулей

Также в окне просмотра списка используемых содулей, можно просмотреть полный путь к используемому модулю на этом компьютере.

## Используемые функции

В данном разделе описываются функции, использованные или созданные в программе.

Функция IEnumerable<ProcessInfo> GetProcessesInfo().

Данная функция вызывается без параметров и используется для получения списка запущенных процессов на компьютере.

Функция void ProcessUpdate(object sender, EventArgs e).

Данная функция вызывается таймером, поэтому содержит 2 параметра.

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция void SetInformationForUser(Process process).

Данная функия используется для установки значений в поле информации о процессе. Устанавливаются такие значения как: название окна исполняемого процесса, время выполнения процесса, время запуска процесса, количество потоков процесса.

Параметры:

* process – процесс информация которого будет выводиться.

Функция void SetAdditionalInformation(Process process).

Данная функция используется для установки названий используемых модулей в поле дополнительной информации.

Параметры:

* process – процесс информация которого будет выводиться.

Функция int GetPriorityNumber(ProcessPriorityClass priorityClass).

Данная функция используется получения индекса приоритета, для повышения или понижения. Возвращает индекс приоритета.

Параметры:

* priorityClass – объект приоритета.

Функция ProcessPriorityClass GetSelectedPriorityClass(int index).

Данная функция используется для получения объекта приоритета по его индексу. Возвращаемое значение - объект приоритета.

Параметры:

* index – индекс приоритета.

Функция void ListBox\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e).

Данная функция вызывается в ответ на отпускание левой кнопки мыши. Функиця вызывает другие методы, которые устанавливают дополнительную и основную информацию о процессе. Также показывает текущий приоритет процесса.

Параметры:

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция Process GetSelectedProcess(object sender).

Данная функция возвращает выбранный процесс для использования другими функциями.

Параметры:

* sender – объект DataGrid.

Функция void ButtonRemoveProc\_Click(object sender, RoutedEventArgs e)

Данная функция вызывается в ответ на нажатие кнопки удаления процесса. Метод удаляет выбранный процесс из списка.

Параметры:

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция void Button\_Click(object sender, RoutedEventArgs e)

Данная функция вызывается в ответ на нажатие кнопки открытия процесса. Функция запускает исполняемый файл по его пути.

Параметры:

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция void Button\_Click\_1(object sender, RoutedEventArgs e)

Данная функция вызывается в ответ на нажатие кнопки изменения приоритета.

Функция исзменяет приоритет процесса на приоритет который вырал пользователь из списка доступных.

Параметры:

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция void MenuItem\_Click(object sender, RoutedEventArgs e)

Данная функция вызывается в ответ на нажатие кнопки пункта меню, о переходе в расположение исполняемого файла.

Параметры:

* sender – объект который вызвал данный метод;
* e – параметры передаваемые в функцию.

Функция static extern bool GetOpenFileName([In, Out] OpenFileName ofn).

Данная функция находится в библиотеке «Comdlg32.dll». Данная функция используется для вызова системного проводника. Данная функция принимает структуру типа OpenFileName. В этой структуре кроме всех свойсв, присутствует свойство хранящее путь до выбранного исполняемого файла. Этот путь используется для запуска процесса.

Параметры:

* ofn – структура типа OpenFileName.

1. **Тестирование**
   1. **Тестирование запуска процесса**

Для запуска процесса, согласно рисунку 21, нажимается соответствующая кнопка и выбирается путь до исполняемого файла процесса, согласно рисунку 22, после чего, программа сама запускает данный файл по его адресу на компьютере. Запущенный файл отображается в списках процессов, согласно рисунку 23 и рисунку 24.

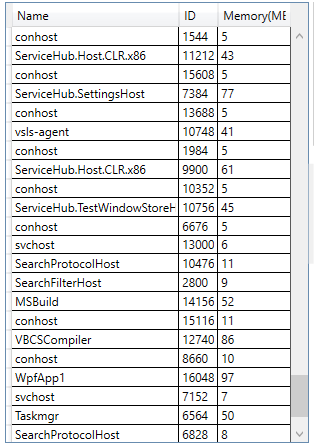


Рисунок 21 – Список процессов до запуска нового процесса



Рисунок 22 – Ярлык на исполняемый файл процесса

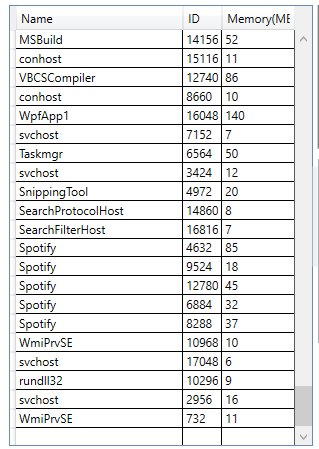


Рисунок 23 – Список процессов после запуска нового процесса



Рисунок 24 – Запущенный процесс в стандартном диспетчере задач

* 1. **Завершение процесса**

Для завершения процесса нужно выбрать желаемый процесс, согласно рисунку 25, и нажать на соответствующую кнопку завершения, согласно рисунку 26.

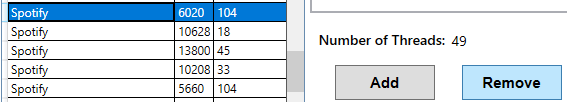


Рисунок 25 – Выбранный процесс и кнопка завершения

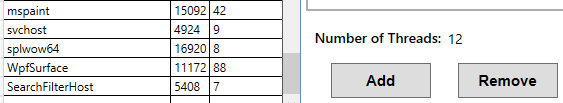


Рисунок 26 – Список процессов после завершения

* 1. **Изменение приоритета**

Для изменения приоритета выбирается процесс, из списка доступных приоритетов, согласно рисунку 27, выбирается соответствующий приоритет и нажимается кнопка подтверждения. Результат соответствует рисунку 28.



Рисунок 27 – Выбранный процесс с Normal приоритетом

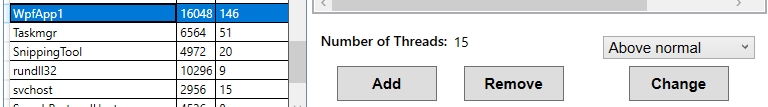


Рисунок 28 – Выбранный процесс с измененным приоритетом на Above normal

* 1. **Просмотр информации о процессе**

Для просмотра информации о процессе, нужно выбрать процесс из списка процессов, после чего, вся доступная информация будет выведена в соответствующие поля, согласно рисунку 29.

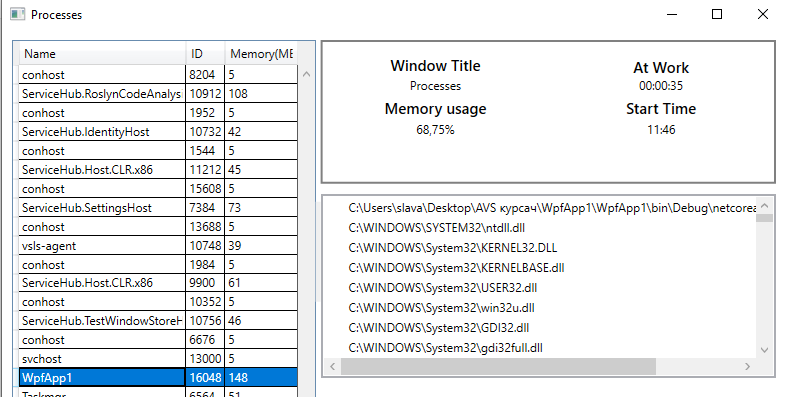


Рисунок 29 – Вывод информации о процессе

Список доступной информации:

1. список используемых содулей;
2. время начала процесса;
3. время исполнения процесса;
4. название окна исполняемого процеса;
5. id процесса;
6. количество используемой памяти.

**ЗАКЛЮЧЕНИЕ**

В ходе данного курсового проекта была смоделирована программа, которая собирает информацию о системе, процессах и потоках. В возможности данного проекта входит просмотр подробной информации о процессе, закрытие процессов и открытие новых.

Открытие реализовано с помощью системного проводника для удобства пользователя, который работает с конкретной системой.

Также, программа позволяет повышать и понижать приоритет процесса с помощью списка возможных приоритетов и кнопки для подтверждения изменений.

Программа имеет функцию для просмотра местоположения исполняемого файла процесса на диске, с помощью вызова контекстного меню и выбора соответствующего пункта, после чего, открывается стандартный системный проводник, который указывает на исполняемый файл, который находится в данной директории.

Были реализованы поставленные задачи и проведены тесты, для исключения непредвиденных ситуаций в ходе использования программы пользователями.

В планах развития проекта – создание распределенной базы данных, для повышения надежности, адаптация данного сайта под разрешение мобильных телефонов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Фриман, Адам Pro ASP.NET Core MVC 2 / Адам Фриман. - Лондон: Apress. 2017. – 1013 с
2. Натан, Адам WPF 4 Подробное руководство / Адам Натан. - Санкт-Петербург: Символ-Плюс. 2019. – 878 с
3. Рихтер, Джефри CLR via C#: Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Джефри Рихтер. - Санкт-Петербург: Питер, 2013. – 893 с
4. Альтернативный диспетчер задач для Windows 7/10 [Электронный ресурс].: Свободная энциклопедия. – Электронные данные. – Режим доступа: <https://it-like.ru/alternativa-dispetcheru-zadach/> – Дата доступа : 17.10.2020.
5. Обзор Windows Forms [Электронный ресурс].: Официальная документация. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/windows-forms-overview>. – Дата доступа: 05.05.2020.
6. Общие сведения о WPF [Электронный ресурс].: Официальная документация. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/introduction-to-wpf>. – Дата доступа: 05.05.2020.

**ПРИЛОЖЕНИЕ A**

**MainWindow.xaml**

<Window x:Class="WpfApp1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1"

mc:Ignorable="d"

Title="Processes" Height="500" Width="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="1.5\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

<RowDefinition Height="2\*"/>

</Grid.RowDefinitions>

<DataGrid x:Name="dataGrid" Margin="10,10,0,10" Grid.RowSpan="2" MouseLeftButtonUp="ListBox\_MouseLeftButtonUp" ItemsSource="{Binding listOfProcInfo}" AutoGenerateColumns="False">

<DataGrid.Columns>

<DataGridTextColumn Header="Name" Binding="{Binding Name}" Width="3\*"/>

<DataGridTextColumn Header="ID" Binding="{Binding Id}" Width="0.7\*"/>

<DataGridTextColumn Header="Memory(MB)" Binding="{Binding MemorySize}" Width="1.3\*"/>

</DataGrid.Columns>

<DataGrid.ContextMenu>

<ContextMenu>

<MenuItem Header="Show folder" Click="MenuItem\_Click"/>

</ContextMenu>

</DataGrid.ContextMenu>

</DataGrid>

<Button Content="Add" FontSize="15" FontWeight="SemiBold" FontFamily="Arial" Grid.Column="1" Margin="30,0,0,26" Grid.Row="1" Click="Button\_Click" HorizontalAlignment="Left" Width="100" Height="35" VerticalAlignment="Bottom"/>

<Button x:Name="buttonRemoveProc" Content="Remove" FontSize="15" FontWeight="SemiBold" FontFamily="Arial" Grid.Column="1" Margin="157,0,0,26" Grid.Row="1" Click="ButtonRemoveProc\_Click" Height="35" VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="100"/>

<Border BorderThickness="2" BorderBrush="Gray" Grid.Column="1" Margin="5,10,10,0">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Content="Window Title" FontSize="15" FontWeight="Medium" FontFamily="Times" HorizontalAlignment="Center" Margin="0,10,0,0" VerticalAlignment="Top"/>

<Label Content="Memory usage" FontSize="15" FontWeight="Medium" FontFamily="Times" HorizontalAlignment="Center" Margin="0,0,0,58" Height="28" VerticalAlignment="Bottom"/>

<Label Content="At Work" FontSize="15" FontWeight="Medium" FontFamily="Times" HorizontalAlignment="Center" Margin="0,12,0,0" VerticalAlignment="Top" Grid.Column="1"/>

<Label Content="Start Time" FontSize="15" FontWeight="Medium" FontFamily="Times" Grid.Column="1" HorizontalAlignment="Center" Margin="0,0,0,58" VerticalAlignment="Bottom"/>

<Label x:Name="lableWindowTitle" Content="0" HorizontalAlignment="Center" Margin="0,30,0,0" VerticalAlignment="Top"/>

<Label x:Name="LableAtWork" Content="0" HorizontalAlignment="Center" Margin="0,30,0,0" VerticalAlignment="Top" Grid.Column="1"/>

<Label x:Name="lableStartTime" Content="0" Grid.Column="1" HorizontalAlignment="Center" Margin="0,0,0,40" VerticalAlignment="Bottom"/>

<Label x:Name="lablePersent" Content="0" HorizontalAlignment="Center" Margin="0,0,0,40" VerticalAlignment="Bottom"/>

</Grid>

</Border>

<GridSplitter HorizontalAlignment="Left" Height="100" Margin="0,18,0,0" Grid.Row="1" VerticalAlignment="Top" Width="5" Grid.Column="1" MaxWidth="20"/>

<ListBox x:Name="listBoxAdditionalInfo" Grid.Column="1" Margin="5,10,10,113" Grid.Row="1" BorderThickness="2"/>

<ComboBox x:Name="comboPriority" Grid.Column="1" FontFamily="Arial" FontSize="15" Margin="296,0,0,68" Grid.Row="1" VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="152">

<ComboBoxItem Content="Low"/>

<ComboBoxItem Content="Below normal"/>

<ComboBoxItem Content="Normal"/>

<ComboBoxItem Content="Above normal"/>

<ComboBoxItem Content="Hight"/>

<ComboBoxItem Content="Real time"/>

</ComboBox>

<Button Content="Change" FontSize="15" FontWeight="SemiBold" FontFamily="Arial" Grid.Column="1" Margin="322,0,0,26" Grid.Row="1" VerticalAlignment="Bottom" Height="35" Click="Button\_Click\_1" HorizontalAlignment="Left" Width="100"/>

<Label Content="Number of Threads: " FontSize="14" FontWeight="SemiBold" Grid.Column="1" HorizontalAlignment="Left" Margin="9,0,0,75" Grid.Row="1" Height="26" VerticalAlignment="Bottom"/>

<Label x:Name="NumberOfThreads" FontSize="14" Content="0" Grid.Column="1" HorizontalAlignment="Left" Margin="142,0,0,71" Grid.Row="1" VerticalAlignment="Bottom"/>

</Grid>

</Window>

**MainWindow.xaml.sh**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.InteropServices;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Threading;

using WpfApp1.Model;

namespace WpfApp1

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private readonly DispatcherTimer \_processTimer = new DispatcherTimer() { Interval = TimeSpan.FromSeconds(5),};

private static readonly int \_memorySize = 80;

[DllImport("Comdlg32.dll", CharSet = CharSet.Auto)]

public static extern bool GetOpenFileName([In, Out] OpenFileName ofn);

public MainWindow()

{

InitializeComponent();

ProcessUpdate(null, null);

\_processTimer.Tick += ProcessUpdate;

\_processTimer.Start();

}

private IEnumerable<ProcessInfo> GetProcessesInfo()

{

var processes = Process.GetProcesses();

var listOfProc = new List<ProcessInfo>();

foreach (var proc in processes)

{

var procInfo = new ProcessInfo

{

Name = proc.ProcessName,

Id = proc.Id,

MemorySize = proc.WorkingSet64 / 1000000,

};

listOfProc.Add(procInfo);

}

return listOfProc;

}

private void ProcessUpdate(object sender, EventArgs e)

{

double memorySize = 0;

double driveSize = 0;

var listOfProcInfo = GetProcessesInfo();

listOfProcInfo.ToList().ForEach(e => { memorySize += e.MemorySize; });

dataGrid.ItemsSource = listOfProcInfo;

var memoryPersent = memorySize / \_memorySize;

lablePersent.Content = memoryPersent.ToString() + "%";

}

private void SetInformationForUser(Process process)

{

lableWindowTitle.Content = process.MainWindowTitle;

LableAtWork.Content = process.UserProcessorTime.ToString(@"hh\:mm\:ss");

lableStartTime.Content = process.StartTime.ToString("HH:mm");

NumberOfThreads.Content = process.Threads.Count.ToString();

}

private void SetAdditionalInformation(Process process)

{

listBoxAdditionalInfo.Items.Clear();

foreach (ProcessModule modul in process.Modules)

{

var treeViewThread = new TreeViewItem

{

Header = modul.FileName,

};

listBoxAdditionalInfo.Items.Add(treeViewThread);

}

}

private int GetPriorityNumber(ProcessPriorityClass priorityClass)

{

return priorityClass switch

{

ProcessPriorityClass.Idle => 0,

ProcessPriorityClass.BelowNormal => 1,

ProcessPriorityClass.Normal => 2,

ProcessPriorityClass.AboveNormal => 3,

ProcessPriorityClass.High => 4,

ProcessPriorityClass.RealTime => 5,

\_ => 0,

};

}

private ProcessPriorityClass GetSelectedPriorityClass(int index)

{

return index switch

{

0 => ProcessPriorityClass.Idle,

1 => ProcessPriorityClass.BelowNormal,

2 => ProcessPriorityClass.Normal,

3 => ProcessPriorityClass.AboveNormal,

4 => ProcessPriorityClass.High,

5 => ProcessPriorityClass.RealTime,

\_ => ProcessPriorityClass.Normal,

};

}

private void ListBox\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

try

{

Process process = GetSelectedProcess(sender);

SetInformationForUser(process);

SetAdditionalInformation(process);

comboPriority.SelectedItem =comboPriority.Items[GetPriorityNumber(process.PriorityClass)];

}

catch

{

MessageBox.Show("процесс не выбран");

}

}

private Process GetSelectedProcess(object sender)

{

var selectedIndex = (sender as DataGrid).SelectedIndex;

var selectedItem = (ProcessInfo)dataGrid.Items.GetItemAt(selectedIndex);

var processId = selectedItem.Id;

Process process = Process.GetProcessById(processId);

return process;

}

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

{

try

{

var selectedIndex = dataGrid.SelectedIndex;

var selectedItem = (TreeViewItem)dataGrid.Items.GetItemAt(selectedIndex);

if (selectedItem != null)

{

var processId = (int)selectedItem.Tag;

var processName = Process.GetProcessById(processId).ProcessName;

foreach (var process in Process.GetProcessesByName(processName))

{

process.CloseMainWindow();

process.Close();

}

}

ProcessUpdate(null, null);

}

catch

{ }

}

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

{

OpenFileName ofn = new OpenFileName();

ofn.structSize = Marshal.SizeOf(ofn);

ofn.filter = "All\0\*.\*\0App\0\*.EXE\0";

ofn.file = new String(new char[256]);

ofn.maxFile = ofn.file.Length;

ofn.fileTitle = new String(new char[64]);

ofn.maxFileTitle = ofn.fileTitle.Length;

ofn.initialDir = "C:\\";

ofn.title = "Open file called using platform invoke...";

ofn.defExt = "txt";

GetOpenFileName(ofn);

if (ofn.file.Contains(".exe"))

{

Process.Start(ofn.file);

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

try

{

Process process = GetSelectedProcess(dataGrid);

var selectedIndex = comboPriority.SelectedIndex;

var priority = GetSelectedPriorityClass(selectedIndex);

process.PriorityClass = priority;

MessageBox.Show("Приоритет изменен");

}

catch

{

MessageBox.Show("Процесс не выбран");

}

}

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

{

var selectedIndex = dataGrid.SelectedIndex;

var selectedItem = (ProcessInfo)dataGrid.Items.GetItemAt(selectedIndex);

var selected = Process.GetProcesses().First(e => e.Id == selectedItem.Id);

Process PrFolder = new Process();

ProcessStartInfo psi = new ProcessStartInfo();

string file = selected.MainModule.FileName;

psi.CreateNoWindow = true;

psi.WindowStyle = ProcessWindowStyle.Normal;

psi.FileName = "explorer";

psi.Arguments = @"/n, /select, " + file;

PrFolder.StartInfo = psi;

PrFolder.Start();

}

}

}

**ProcessInfo.sh**

namespace WpfApp1.Model

{

class ProcessInfo

{

public string Name { get; set; }

public int Id { get; set; }

public double MemorySize { get; set; }

}

}

**OpenFileName.sh**

using System;

using System.Runtime.InteropServices;

namespace WpfApp1

{

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]

public class OpenFileName

{

public int structSize = 0;

public IntPtr dlgOwner = IntPtr.Zero;

public IntPtr instance = IntPtr.Zero;

public String filter = null;

public String customFilter = null;

public int maxCustFilter = 0;

public int filterIndex = 0;

public String file = null;

public int maxFile = 0;

public String fileTitle = null;

public int maxFileTitle = 0;

public String initialDir = null;

public String title = null;

public int flags = 0;

public short fileOffset = 0;

public short fileExtension = 0;

public String defExt = null;

public IntPtr custData = IntPtr.Zero;

public IntPtr hook = IntPtr.Zero;

public String templateName = null;

public IntPtr reservedPtr = IntPtr.Zero;

public int reservedInt = 0;

public int flagsEx = 0;

}

}