

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»  
Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина: Операционные системы и среды

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему  
«Список окон и управление ими (в т.ч. с фильтрацией/отбором)»

Выполнил студент гр. 853505  
Яскевич Н.Н.

Проверил  
Протьюко М.И.

Минск 2021

## **Оглавление**

Введение	3
1. Анализ предметной области	5
2. Описание работы программы	8
3. Сравнение с существующими аналогами	11
4. Заключение	13
Список использованной литературы	14
Приложение 1. Текст программы	15

## Введение

**Менеджер окон (Task Manager)** — компьютерная программа, управляющая

размещением и определяющая внешний вид окон в оконной системе графического пользовательского интерфейса. Оконные менеджеры работают «поверх» существующей оконной системы, обеспечивающей требуемую функциональность, такую как поддержку графического оборудования, манипуляторов, клавиатуры. Часто оконные менеджеры базируются на некоторой библиотеке пользовательского интерфейса.

Менеджеры окон делятся на 2 типа:

Фреймовый(плиточный, тайловый) оконный менеджер

Композитный оконный менеджер

**Диспетчер задач** в операционных системах семейства Microsoft Windows — утилита для вывода на экран списка запущенных процессов и потребляемых ими ресурсов (в частности статус, процессорное время и потребляемая оперативная память). Также есть возможность некоторой манипуляции процессами.

Windows Task Manager в Windows NT можно вызвать, одновременно нажав клавиши Ctrl+Shift+Esc. В Windows NT и в Windows XP существует более известная комбинация клавиш — Ctrl+Alt+Del. Диспетчер задач можно также запустить в командной строке, введя имя его исполняемого файла (*taskmgr.exe*) или выбрав соответствующий пункт в контекстном меню панели задач.

Диспетчер задач — встроенная в операционную систему утилита. Она содержит вкладки:

- **Приложения.** Позволяет переключиться в нужное приложение, либо завершить его.

- Процессы. Разнообразные данные обо всех запущенных в системе процессах, можно завершать, менять приоритет, задавать соответствие процессорам (в многопроцессорных системах)
- Службы (начиная с Vista). Сведения обо всех службах Windows.
- Быстродействие. Графики загрузки процессора (процессоров), использования оперативной памяти.
- Сеть (отсутствует в случае отсутствия активных сетевых подключений). Графики загрузки сетевых подключений.
- Пользователи (только в режиме администратора). Манипулирование активными пользователями.

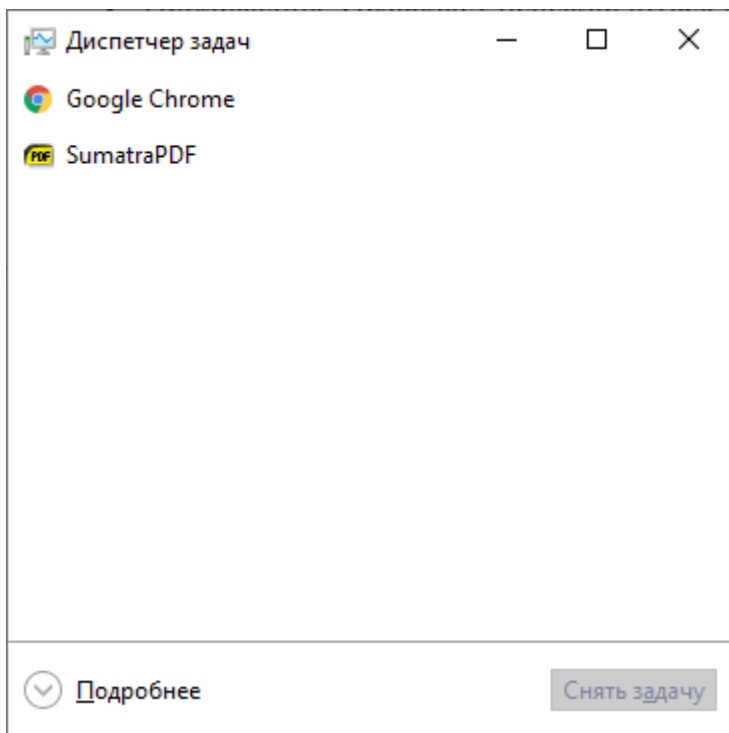


Рис. 1. Встроенный в Windows диспетчер задач

Диспетчер задач						
Файл Параметры Вид						
Процессы Производительность Журнал приложений Автозагрузка Пользователи Подробности Службы						
Имя	Состояние	9% ЦП	67% Память	19% Диск	0% Сеть	
>  Служба узла: Служба пользов...		0,3%	7,1 МБ	0,1 МБ/с	0 Мбит/с	^
>  Google Chrome (11)		1,8%	354,0 МБ	0,1 МБ/с	0 Мбит/с	
>  Oracle RDBMS Kernel Executable		0%	674,6 МБ	0,1 МБ/с	0 Мбит/с	
System		0,3%	0,1 МБ	0,1 МБ/с	0 Мбит/с	
Google Chrome		0%	14,4 МБ	0,1 МБ/с	0,1 Мбит/с	
>  Antimalware Service Executable		0%	185,9 МБ	0,1 МБ/с	0 Мбит/с	
>  Узел службы: локальная систе...		0%	107,9 МБ	0,1 МБ/с	0 Мбит/с	
>  Intel(R) System Usage Report		0%	21,9 МБ	0,1 МБ/с	0 Мбит/с	
Intel(R) System Usage Report		0%	4,9 МБ	0,1 МБ/с	0 Мбит/с	
ACMON (32 бита)		0%	0,1 МБ	0 МБ/с	0 Мбит/с	
>  Adobe Genuine Software Integri...		0%	0,1 МБ	0 МБ/с	0 Мбит/с	
>  Adobe Genuine Software Servic...		0%	0,1 МБ	0 МБ/с	0 Мбит/с	
Application Frame Host		0%	6,7 МБ	0 МБ/с	0 Мбит/с	
>  ASLDR Service (32 бита)		0%	0,1 МБ	0 МБ/с	0 Мбит/с	v
<  >						
<a href="#">Меньше</a>						<a href="#">Снять задачу</a>

Рис. 1.2 Встроенный в Windows диспетчер задач

**Композитный менеджер окон** — менеджер окон, использующий возможности окружения (например, опциональной функции Composite X11-сервера или средств Windows Aero) по задействованию аппаратного ускорения для отображения прозрачности, отрисовки теней, отображения текстур, трёхмерных эффектов, анимации, экранных луп.

В отличие от ранних оконных менеджеров, которые делали каждую индивидуальную программу ответственной за предоставление своего окна непосредственно в кадровом буфере, композитный менеджер обеспечивает приложениям вне экрана буфер памяти окна и композитов окна в изображение, представляющее экран и пишет результат в кадровом буфере.

Композитный менеджер может выполнять дополнительную обработку буфера окна, применяя 2D- и 3D-анимационные эффекты, такие как альфа-смешивание, выцветание, масштабирование, поворот, копирование, изгиб и искривление, размытость. Также возможен перевод окна в один из нескольких дисплеев и виртуальных рабочих столов. Данная технология позволяет в режиме реального времени просчитывать такие эффекты как падающие тени, живые предварительные просмотры окон и другие сложные эффекты.

# 1. Анализ предметной области

## 1.1. Краткие теоретические сведения. Список используемых терминов

**Окно** — графически выделенная часть экрана, принадлежащая какому-либо объекту, с которым работает пользователь

**Активное окно** - это текущее фокусируемое окно в текущем оконном менеджере. Различные оконные менеджеры по-разному указывают на активное в данный момент окно и позволяют пользователю переключаться между окнами по-разному. Например, в Microsoft Windows, если открыты и Блокнот , и Microsoft Paint , щелчок в окне Блокнот приведет к тому, что это окно станет активным. В Windows активное окно обозначается полосой заголовка другого цвета.

## 1.2. Технологии и средства разработки

В качестве языка реализации программного средства выбран C#. Разработка велась с использованием библиотек “**Microsoft.VisualBasic**” ”**System.Windows.Forms**” – библиотеки, позволяющие облегчить работу с окнами и операциями создания оконного приложения. При написании и тестировании исходного кода использовалась интегрированная среда разработки Visual Studio 2019.

**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



### **1.3. Задачи проектирования приложения. Описание модулей программы**

Основные задачи проектирования приложения:

Список окон и управление ими (в т.ч. с фильтрацией/отбором)

Готовый программный продукт должен удовлетворять следующим требованиям:

реализовывать все необходимые алгоритмы поведения приложения и его полноценного взаимодействия с пользователем.

В реализованном программном продукте имеются следующие модули:

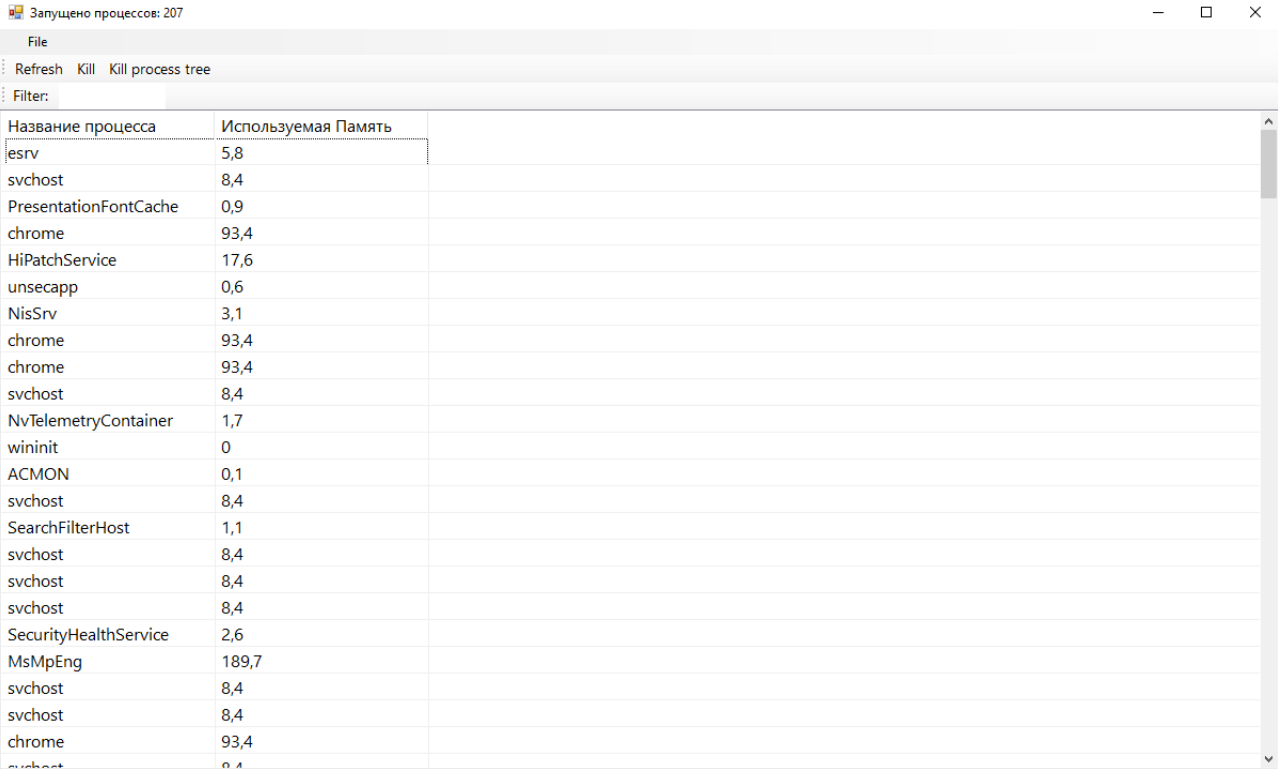
`ListViewItemComparer.cs` – класс, позволяющий нам сортировать элементы в списке `ListViewItems` без переопределения списка.

`Form1.cs` – класс определяющий поведение приложения, при взаимодействии с пользователем

`Form1.Designer.cs` – класс, определяющий визуальные элементы окна, и связывающий методы приложения, с элементами интерфейса, их вызывающими.

## 2. Описание работы программы

При запуске программы, мы видим в таблице все запущенные на компьютере процессы. В верхнем правом угле находятся элементы управления списком процессов



File	
Refresh	Kill Kill process tree
Filter:	
Название процесса	Используемая Память
esrv	5,8
svchost	8,4
PresentationFontCache	0,9
chrome	93,4
HiPatchService	17,6
unsecapp	0,6
NisSrv	3,1
chrome	93,4
chrome	93,4
svchost	8,4
NvTelemetryContainer	1,7
wininit	0
ACMON	0,1
svchost	8,4
SearchFilterHost	1,1
svchost	8,4
svchost	8,4
svchost	8,4
SecurityHealthService	2,6
MsMpEng	189,7
svchost	8,4
svchost	8,4
chrome	93,4
svchost	8,4

Рис. 2. Список процессов



В контекстном меню, мы можем закрыть нужный нам процесс. Если у приложения несколько процессов, например у Google Chrome - то можно закрыть все процессы:

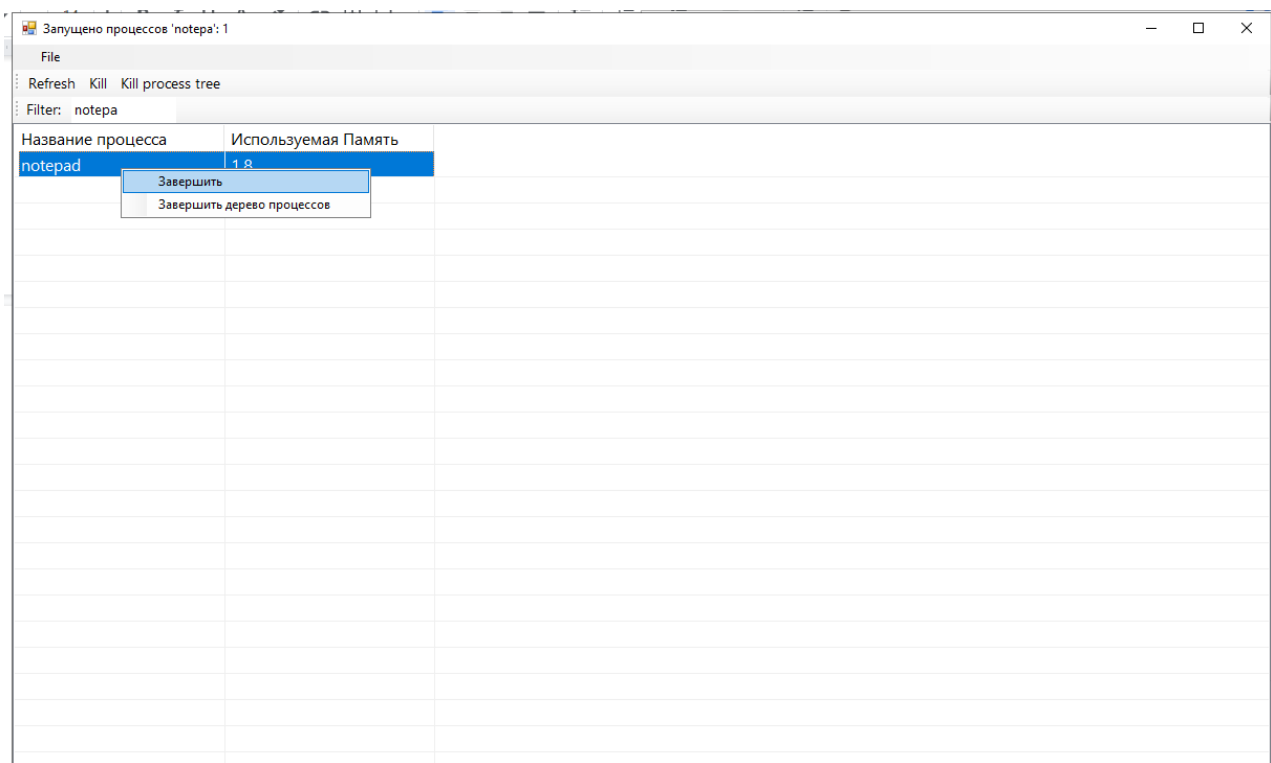


Рис. 5. Закрываем один процесс

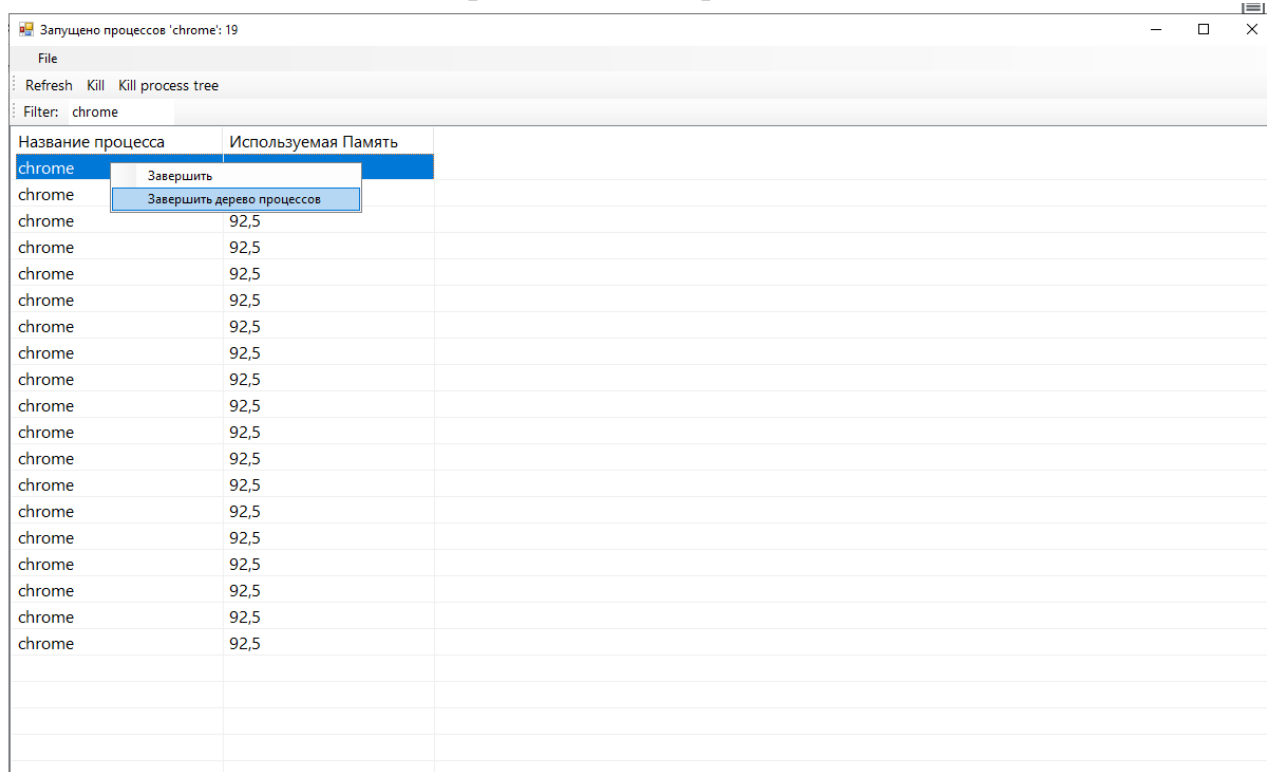
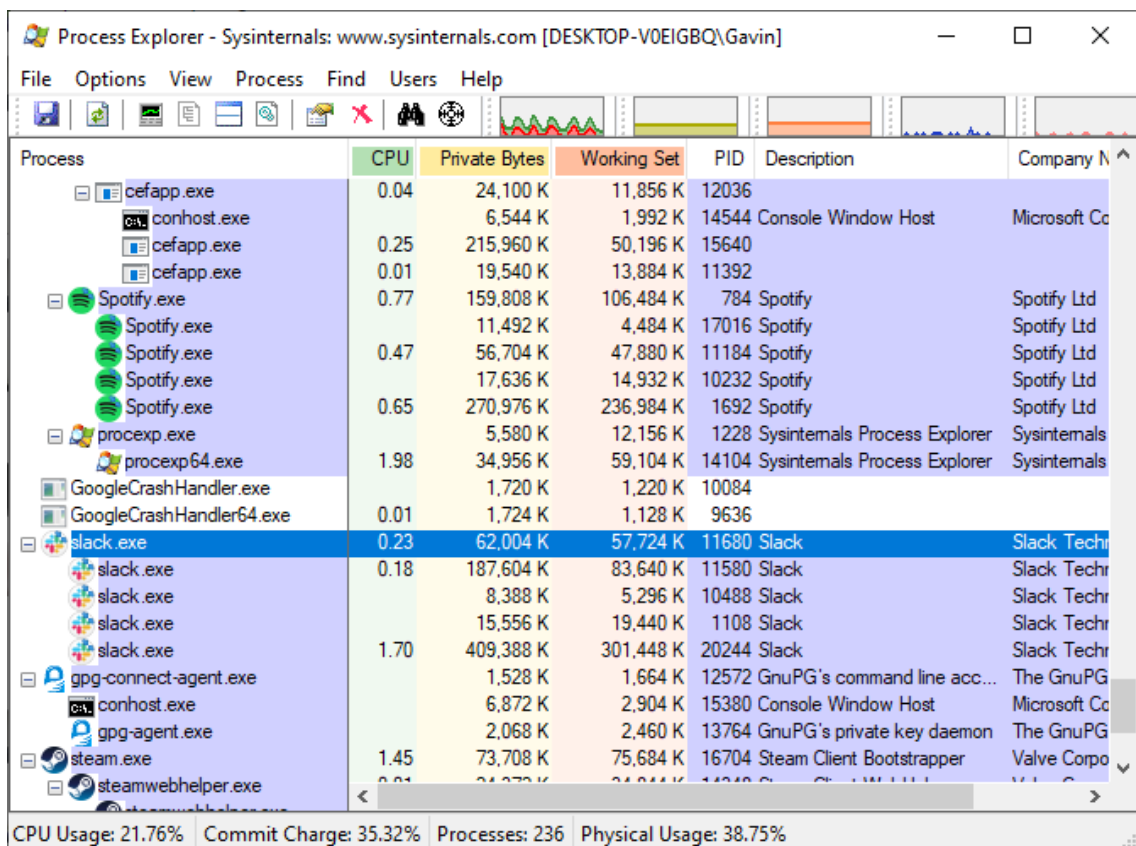


Рис.6. Закрываем дерево процессов

### 3. Сравнение с существующими аналогами

Существует достаточно большое количество программ, которые имеют схожий с разработанным в рамках курсового проекта программным продуктом функционал. Одними из самых популярных являются Process Explorer (см. рис. 7) и System Explorer (см. рис. 8).

Process Explorer - это заряженная версия диспетчера задач Windows. Process Explorer был разработан SysInternals до тех пор, пока Microsoft не приобрела компанию. Альтернатива диспетчеру задач продолжает жить, и компания переименована в Windows Sysinternals. После запуска вы увидите обзор всех запущенных процессов в вашей системе, организованных в виде иерархии. Также есть нижняя панель (по умолчанию отключена), которая показывает, какие библиотеки DLL или дескрипторы используются процессами. Сочетание обеих функций значительно упрощает поиск и устранение неисправностей в системе.



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
cefapp.exe	0.04	24,100 K	11,856 K	12036		
conhost.exe		6,544 K	1,992 K	14544	Console Window Host	Microsoft Co
cefapp.exe	0.25	215,960 K	50,196 K	15640		
cefapp.exe	0.01	19,540 K	13,884 K	11392		
Spotify.exe	0.77	159,808 K	106,484 K	784	Spotify	Spotify Ltd
Spotify.exe		11,492 K	4,484 K	17016	Spotify	Spotify Ltd
Spotify.exe	0.47	56,704 K	47,880 K	11184	Spotify	Spotify Ltd
Spotify.exe		17,636 K	14,932 K	10232	Spotify	Spotify Ltd
Spotify.exe	0.65	270,976 K	236,984 K	1692	Spotify	Spotify Ltd
procexp.exe		5,580 K	12,156 K	1228	Sysinternals Process Explorer	Sysinternals
procexp64.exe	1.98	34,956 K	59,104 K	14104	Sysinternals Process Explorer	Sysinternals
GoogleCrashHandler.exe		1,720 K	1,220 K	10084		
GoogleCrashHandler64.exe	0.01	1,724 K	1,128 K	9636		
slack.exe	0.23	62,004 K	57,724 K	11680	Slack	Slack Techn
slack.exe	0.18	187,604 K	83,640 K	11580	Slack	Slack Techn
slack.exe		8,388 K	5,296 K	10488	Slack	Slack Techn
slack.exe		15,556 K	19,440 K	1108	Slack	Slack Techn
slack.exe	1.70	409,388 K	301,448 K	20244	Slack	Slack Techn
gpg-connect-agent.exe		1,528 K	1,664 K	12572	GnuPG's command line acc...	The GnuPG
conhost.exe		6,872 K	2,904 K	15380	Console Window Host	Microsoft Co
gpg-agent.exe		2,068 K	2,460 K	13764	GnuPG's private key daemon	The GnuPG
steam.exe	1.45	73,708 K	75,684 K	16704	Steam Client Bootstrapper	Valve Corpo
steamwebhelper.exe						

CPU Usage: 21.76% Commit Charge: 35.32% Processes: 236 Physical Usage: 38.75%

Рис.7. Сессия Proc Explorer

Несмотря на свое общее название, System Explorer далеко не обычная замена диспетчеру задач. Он не только помогает в управлении процессами, но также имеет несколько функций, которые могут повысить безопасность системы и защитить от сбоев. Есть даже портативная версия. Моя любимая функция - это история использования ЦП для каждого процесса, которую можно просмотреть за предыдущую минуту, прошедший час и прошедший день. Вы также можете просматривать общую производительность системы в режиме реального времени, которая показывает некоторые сложные детали, такие как количество ошибок страниц или процент системных прерываний.

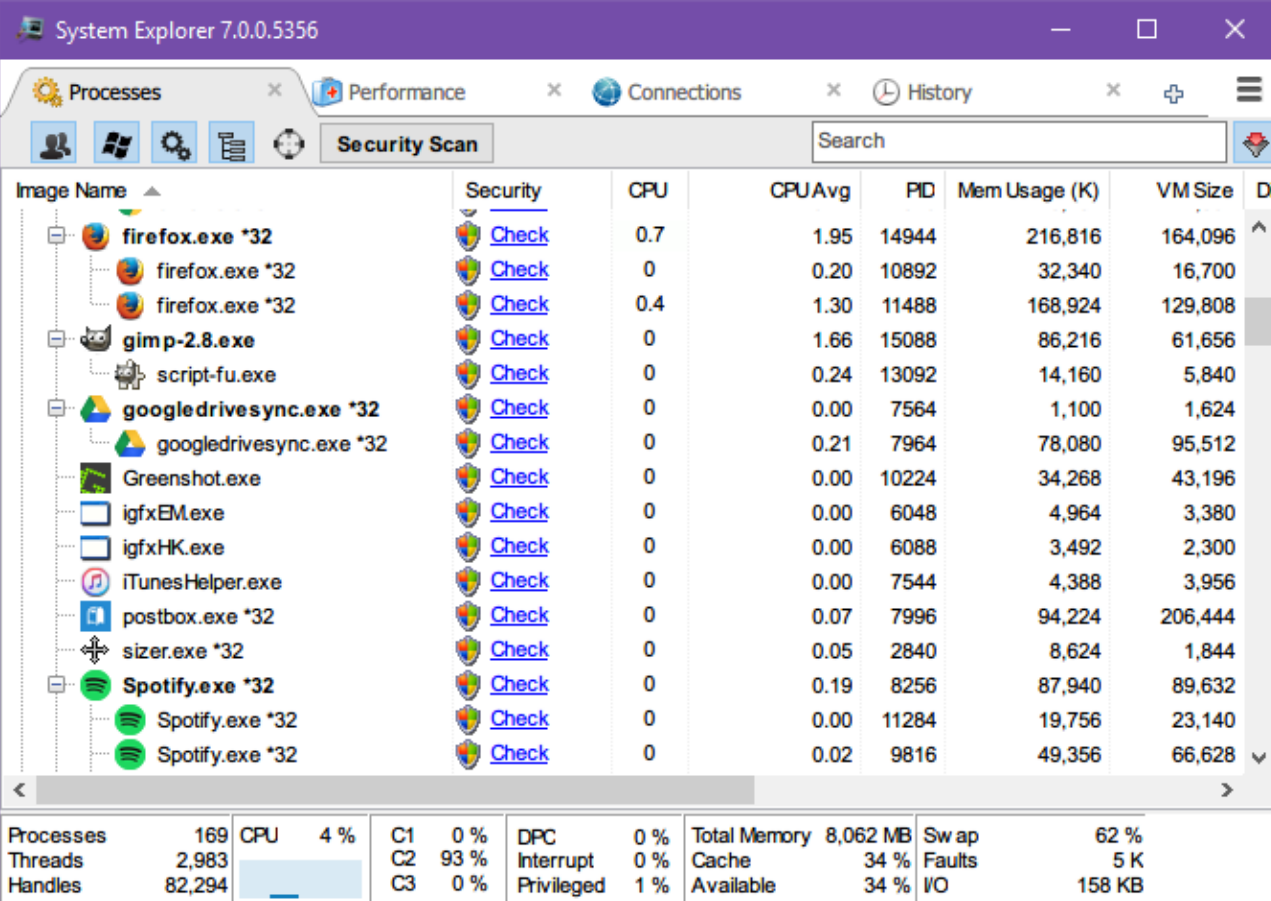


Image Name	Security	CPU	CPU Avg	PID	Mem Usage (K)	VM Size
firefox.exe *32	<a href="#">Check</a>	0.7	1.95	14944	216,816	164,096
firefox.exe *32	<a href="#">Check</a>	0	0.20	10892	32,340	16,700
firefox.exe *32	<a href="#">Check</a>	0.4	1.30	11488	168,924	129,808
gimp-2.8.exe	<a href="#">Check</a>	0	1.66	15088	86,216	61,656
script-fu.exe	<a href="#">Check</a>	0	0.24	13092	14,160	5,840
googledrivesync.exe *32	<a href="#">Check</a>	0	0.00	7564	1,100	1,624
googledrivesync.exe *32	<a href="#">Check</a>	0	0.21	7964	78,080	95,512
Greenshot.exe	<a href="#">Check</a>	0	0.00	10224	34,268	43,196
igfxEM.exe	<a href="#">Check</a>	0	0.00	6048	4,964	3,380
igfxHK.exe	<a href="#">Check</a>	0	0.00	6088	3,492	2,300
iTunesHelper.exe	<a href="#">Check</a>	0	0.00	7544	4,388	3,956
postbox.exe *32	<a href="#">Check</a>	0	0.07	7996	94,224	206,444
sizer.exe *32	<a href="#">Check</a>	0	0.05	2840	8,624	1,844
Spotify.exe *32	<a href="#">Check</a>	0	0.19	8256	87,940	89,632
Spotify.exe *32	<a href="#">Check</a>	0	0.00	11284	19,756	23,140
Spotify.exe *32	<a href="#">Check</a>	0	0.02	9816	49,356	66,628

Processes	169	CPU	4 %	C1	0 %	DPC	0 %	Total Memory	8,062 MB	Swap	62 %
Threads	2,983			C2	93 %	Interrupt	0 %	Cache	34 %	Faults	5 K
Handles	82,294			C3	0 %	Privileged	1 %	Available	34 %	I/O	158 KB

Рис. 8. System Explorer

Для простоты использования и обширная функциональность, Process Explorer - отличный выбор в качестве альтернативы диспетчеру задач Windows. Он отлично интегрируется с Windows 10 и теперь является загрузочным продуктом Microsoft. Тем не менее, если вы хотите выйти за рамки того, что предлагает Windows, попробуйте один из других вариантов.

Поскольку каждая из замен диспетчера задач полностью бесплатна, вы можете увидеть, что соответствует вашим требованиям.

Из вышесказанного следует, что программа, разработанная в рамках данного курсового проекта реализует лишь базовый функционал task менеджеров(что может быть исправлено при дальнейшей разработке)

## **4. Заключение**

В результате курсового проекта был разработан программный продукт, представляющий собой простой таск менеджер. Данный продукт полностью функционален, совместим с операционными системами, начиная с Windows XP.

За время написания проекта был изучен большой объем информации по оконным менеджерам, в том числе используемым в ОС Windows, а также по созданию приложений с помощью Windows Forms на C#. Закреплены навыки разработки в интегрированной среде разработки Visual Studio.

Был проведён обзор и сравнительный анализ аналогов данного программного продукта, в том числе на других операционных системах. Также были проанализированы различные подходы к проектированию и разработке, после чего избраны наиболее подходящие для конкретного программного продукта.

Разработанное приложение представляет собой законченный программный продукт, готовый к использованию и удовлетворяющий поставленным в рамках курсового проекта требованиям и задачам проектирования. Тем не менее, функциональность приложения может быть расширена: перемещение окон, динамическое изменение размера и т.д. Несмотря на это, в целом данную программу можно использовать для базовых операций с процессами.



## Список использованной литературы

1. C# | Введение [Электронный ресурс]. Режим доступа: (<https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.dllimportattribute?view=net-5.0>)
2. Task managers: (<https://www.makeuseof.com/tag/5-powerful-alternatives-windows-task-manager/>)
3. Awesome task manager – Википедия [Электронный ресурс]. Режим доступа: ([https://en.wikipedia.org/wiki/Awesome\\_\(window\\_manager\)](https://en.wikipedia.org/wiki/Awesome_(window_manager)))

## Приложение 1. Текст программы

### ListViewItemComparer.cs

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowManager
{
    class ListViewItemComparer : IComparer
    {
        /*
        Специальный класс-компаратор
        */

        private int _columnIndex;
        public int ColumnIndex
        {
            get { return _columnIndex; }
            set { _columnIndex = value; }
        }

        private SortOrder _sortDirection;
        public SortOrder SortDirection
        {
            get { return _sortDirection; }
            set { _sortDirection = value; }
        }

        public ListViewItemComparer()
        {
            _sortDirection = SortOrder.None;
        }

        public int Compare(object x, object y)
        {
            ListViewItem listViewItemX = x as ListViewItem;
            ListViewItem listViewItemY = y as ListViewItem;

            int result;

            switch (_columnIndex)
            {
                case 0: // Сортировка по названию
                    result = string.Compare(listViewItemX.SubItems[_columnIndex].Text,
                        listViewItemY.SubItems[_columnIndex].Text, false);

                    break;
```

```

        case 1: // Сортировка по объему занимаемой памяти
            double valueX = double.Parse(listViewItemX.SubItems[_columnIndex].Text);
            double valueY = double.Parse(listViewItemY.SubItems[_columnIndex].Text);

            result = valueX.CompareTo(valueY);

            break;

        default:
            result = string.Compare(listViewItemX.SubItems[_columnIndex].Text,
                listViewItemY.SubItems[_columnIndex].Text, false);

            break;
    }

    if (_sortDirection == SortOrder.Descending)
    {
        return -result;
    }
    else
    {
        return result;
    }
}
}
}

```

## Form1.cs

```

using System;
using System.Windows.Forms;
using System.Management;
using Microsoft.VisualBasic;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;

namespace WindowManager
{
    public partial class Form1 : Form
    {
        List<Process> processes = null;
        private ListViewItemComparer comparer = null;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

```

```

processes = new List<Process>();
GetProcesses();
RefreshProcessesList();

comparer = new ListViewItemComparer();
comparer.ColumnIndex = 0;
}

private void GetProcesses()
{
    processes.Clear();
    processes = Process.GetProcesses().ToList<Process>();
}

private void RefreshProcessesList()
{
    listView1.Items.Clear();
    double memSize = 0;
    foreach (Process p in processes)
    {
        memSize = 0;

        PerformanceCounter pc = new PerformanceCounter();
        pc.CategoryName = "Process";
        pc.CounterName = "Working Set - Private";
        pc.InstanceName = p.ProcessName;

        memSize = (double)pc.NextValue() / (1000 * 1000);

        // Массив строк, в котором будут храниться колонки
        string[] row = new string[] { p.ProcessName.ToString(), Math.Round(memSize,1).ToString()};
        listView1.Items.Add(new ListViewItem(row));

        pc.Close();
        pc.Dispose();
    }

    Text = "Запущено процессов: " + processes.Count.ToString();
}

private void RefreshProcessesList(List<Process> processes, string keyword)
{
    try
    {
        listView1.Items.Clear();
        double memSize = 0;

        foreach (Process p in processes)
        {
            if (p != null)
            {
                PerformanceCounter pc = new PerformanceCounter();
                pc.CategoryName = "Process";
                pc.CounterName = "Working Set - Private";
                pc.InstanceName = p.ProcessName;
            }
        }
    }
    catch { }
}

```

```

        memSize = (double)pc.NextValue() / (1000 * 1000);

        // Массив строк, в котором будут храниться колонки
        string[] row = new string[] { p.ProcessName.ToString(), Math.Round(memSize, 1).ToString() };
        listView1.Items.Add(new ListViewItem(row));

        pc.Close();
        pc.Dispose();
    }
}

Text = $"Запущено процессов '{keyword}': " + processes.Count.ToString();
}
catch (Exception) { }
}

private void KillProcess(Process process)
{
    process.Kill();
    process.WaitForExit();
}

private void KillProcessAndChildren(int pid)
{
    if (pid == 0) return;

    ManagementObjectSearcher searcher = new ManagementObjectSearcher(
        "Select * From Win32_Process Where ParentProcessID=" + pid);

    ManagementObjectCollection objectCollection = searcher.Get();

    foreach (ManagementObject obj in objectCollection)
    {
        KillProcessAndChildren(Convert.ToInt32(obj["ProcessID"]));
    }

    try
    {
        Process p = Process.GetProcessById(pid);
        p.Kill();
        p.WaitForExit();
    } catch (ArgumentException) { }
}

private int GetParentProcessId(Process p)
{
    int parentID = 0;

    try
    {
        ManagementObject management = new ManagementObject("win32_process.handle=" + p.Id +
            "");
        management.Get();
        parentID = Convert.ToInt32(management["ParentProcessId"]);
    } catch (Exception) { }
}

```

```

        return parentID;
    }

    private void refreshStripButton1_Click(object sender, EventArgs e)
    {
        GetProcesses();
        RefreshProcessesList();
    }

    private void killStripButton2_Click(object sender, EventArgs e)
    {
        try
        {
            if (listView1.SelectedItems[0] != null)
            {
                // Сравниваем имя каждого процесса в списке, с именем в правой колонке
                Process processToKill = processes.Where((p) => p.ProcessName ==
listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

                KillProcess(processToKill);
                GetProcesses();
                RefreshProcessesList();
            }
        }
        catch (Exception) { }
    }

    private void killTreeStripButton3_Click(object sender, EventArgs e)
    {
        try
        {
            if (listView1.SelectedItems[0] != null)
            {
                // Сравниваем имя каждого процесса в списке, с именем в правой колонке
                Process processToKill = processes.Where((p) => p.ProcessName ==
listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

                KillProcessAndChildren(GetParentProcessId(processToKill));
                GetProcesses();
                RefreshProcessesList();
            }
        }
        catch (Exception) { }
    }

    private void killProcessTreeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        try
        {
            if (listView1.SelectedItems[0] != null)
            {
                // Сравниваем имя каждого процесса в списке, с именем в правой колонке
                Process processToKill = processes.Where((p) => p.ProcessName ==
listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

                KillProcessAndChildren(GetParentProcessId(processToKill));

```

```

        GetProcesses();
        RefreshProcessesList();
    }
}
catch (Exception) { }
}

private void killToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (listView1.SelectedItems[0] != null)
        {
            // Сравниваем имя каждого процесса в списке, с именем в правой колонке
            Process processToKill = processes.Where((p) => p.ProcessName ==
listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

            KillProcess(processToKill);
            GetProcesses();
            RefreshProcessesList();
        }
    }
    catch (Exception) { }
}

//MessageBox & InputBox
private void runTaskToolStripMenuItem_Click(object sender, EventArgs e)
{
    string path = Interaction.InputBox("Введите имя программы", "Запуск новой задачи");

    try
    {
        Process.Start(path);
    }
    catch (Exception) { }
}

// Adding Filter
private void toolStripTextBox1_TextChanged(object sender, EventArgs e)
{
    GetProcesses();
    List<Process> filteredProcesses = processes.Where(p =>
p.ProcessName.ToLower().Contains(toolStripTextBox1.Text.ToLower())).ToList<Process>();

    RefreshProcessesList(filteredProcesses, toolStripTextBox1.Text);
}

private void listView1_ColumnClick(object sender, ColumnClickEventArgs e)
{
    // Получаем индекс колонки, по которой нажали
    comparer.ColumnIndex = e.Column;

    // Меняем направление сортировки
    comparer.SortDirection = (comparer.SortDirection == SortOrder.Ascending) ? SortOrder.Descending
: SortOrder.Ascending;
}

```

```
        listView1.ListViewItemSorter = comparer;
        listView1.Sort();
    }

    private void exitToolStripMenuItem1_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
}
```