

СОДЕРЖАНИЕ

1.	Введение.....	
2.	Задачи курсового проекта	3
3.	Содержание курсового проекта	4
	а. Описание используемых инструментов	
	б. Высокоуровневое проектирование	
	с. Руководство пользователя	
4.	Заключение	
5.	Список использованной литературы.....	27

1 ВВЕДЕНИЕ

Данное приложение будет служить аналогом диспетчера задач – небольшая программа, весьма полезная для тех, кто привык исправлять небольшие погрешности в работе компьютера своими руками. Именно за это её не любят создатели вирусов.

В целом данная программа выполняет две функции – информирует пользователя о процессах и приложениях, запущенных в данный момент на компьютере, и позволяет управлять ими.

2 ЗАДАЧИ КУРСОВОГО ПРОЕКТА

Задачи:

- построение архитектуры приложения
- реализация основного функционала для работы с процессами системы
- построения удобного для пользователя графического интерфейса
- тестирование приложения

3 СОДЕРЖАНИЕ КУРСОВОГО ПРОЕКТА

а. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ИНСТРУМЕНТОВ

Проект был реализован на языке C# с использованием платформы .NET Framework 4.5, технологией WinForms.

C# — элегантный, типобезопасный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде .NET Framework. С помощью языка C# можно создавать обычные приложения Windows, XML-веб-службы, распределенные компоненты, приложения "клиент-сервер", приложения баз данных и т. д. Visual C# предоставляет развитый редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и множество других средств, упрощающих разработку приложений на базе языка C# и .NET Framework.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка, англ. assembly. Затем код либо выполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке Framework Class Library (FCL). В FCL входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

В данном проекте мною была выбрана технология WinForms - интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причем управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке MFC, изначально написанной на языке C++. С другой стороны, WF не предлагает парадигму, сравнимую с MVC. Для исправления этой ситуации и реализации данного функционала в WF существуют сторонние библиотеки. Одной из наиболее используемых подобных библиотек является User Interface Process Application Block, выпущенная специальной группой Microsoft, занимающейся примерами и рекомендациями, для бесплатного скачивания. Эта библиотека также содержит исходный код и обучающие примеры для ускорения обучения.

Внутри .NET Framework, Windows Forms реализуется в рамках пространства имён System.Windows.Forms.

в. ВЫСОКОУРОВНЕВОЕ ПРОЕКТИРОВАНИЕ

Полное решение проекта представлено в виде 5 сборок.

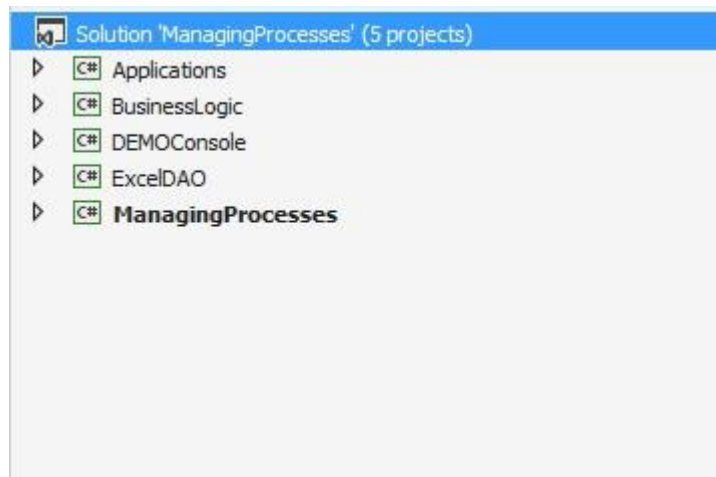


Рисунок 1. Решение проектов

Сборка «Applications» представляет собой классы для работы с конкретными приложениями.

Сборка «Business Logic» представляет собой всю собранную логику для управления процессами и потоками компьютера. Реализаций одного из классов:

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

namespace BusinessLogic
{
    public class ManageProcess
    {
        public ManageProcess()
        {
        }

        public List<Process> GetProcesses()
        {
            return Process.GetProcesses().ToList();
        }

        public List<string> GetProcessNames()
```

```

{
    return Process.GetProcesses().ToList().Select((x) =>
x.ProcessName).ToList();
}

public bool RemoveById(int id)
{
    var process = Process.GetProcessById(id);
    try
    {
        process.Kill();
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}

public bool RemoveByName(string name)
{
    try
    {
        var processes = Process.GetProcessesByName(name);
        processes.ToList().ForEach(x => x.Kill());
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}

public bool AddApplication(string path)
{
    try
    {
        Process.Start(path);
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}

```

```
}  
}
```

Следующая сборка «Demo Console» была проведена для тестирования данных. Благодаря этой сборке были обнаружены неустойчивости работы приложения, которые в последствие были исправлены, например, такие, как:

```
1400 Handle  
596 Id  
EvtEng ProcessName  
255 ProcessorAffinity  
System.Diagnostics.ProcessStartInfo StartInfo  
24.04.2015 22:52:30 StartTime  
24.04.2015 22:52:30 threed!! 0  
24.04.2015 22:52:30 threed!! 1  
24.04.2015 22:52:30 threed!! 2  
24.04.2015 22:52:30 threed!! 3  
24.04.2015 22:52:30 threed!! 4  
24.04.2015 22:52:30 threed!! 5  
24.04.2015 22:52:30 threed!! 6  
24.04.2015 22:52:30 threed!! 7  
24.04.2015 22:52:30 threed!! 8  
24.04.2015 22:52:30 threed!! 9  
24.04.2015 22:52:30 threed!! 10  
24.04.2015 22:52:30 threed!! 11  
24.04.2015 22:52:30 threed!! 12  
24.04.2015 22:52:30 threed!! 13  
24.04.2015 22:52:30 threed!! 14  
24.04.2015 22:52:32 threed!! 15  
24.04.2015 22:52:36 threed!! 16  
24.04.2015 22:52:36 threed!! 17  
26.04.2015 13:43:30 threed!! 18  
26.04.2015 13:48:00 threed!! 19  
26.04.2015 13:49:10 threed!! 20  
00:00:00.5148033 UserProcessorTime  
105136128 VirtualMemorySize64  
7401472 WorkingSet64  
True PriorityBoostEnabled  
Normal PriorityClass  
exception!!!!!!!!!!!!!!!!!!!!System.Collections.ListDictionaryInternal0отказано в доступе  
1404 Handle  
2956 Id  
MailRuUpdater ProcessName  
255 ProcessorAffinity  
System.Diagnostics.ProcessStartInfo StartInfo  
24.04.2015 22:52:38 StartTime  
24.04.2015 22:52:38 threed!! 0  
24.04.2015 22:52:44 threed!! 1  
24.04.2015 22:52:57 threed!! 2  
24.04.2015 22:52:57 threed!! 3  
24.04.2015 22:52:57 threed!! 4  
24.04.2015 22:52:58 threed!! 5  
26.04.2015 13:34:31 threed!! 6  
00:00:11.8092757 UserProcessorTime  
119504896 VirtualMemorySize64  
10928128 WorkingSet64  
True PriorityBoostEnabled  
Normal PriorityClass  
exception!!!!!!!!!!!!!!!!!!!!System.Collections.ListDictionaryInternal0отказано в доступе
```

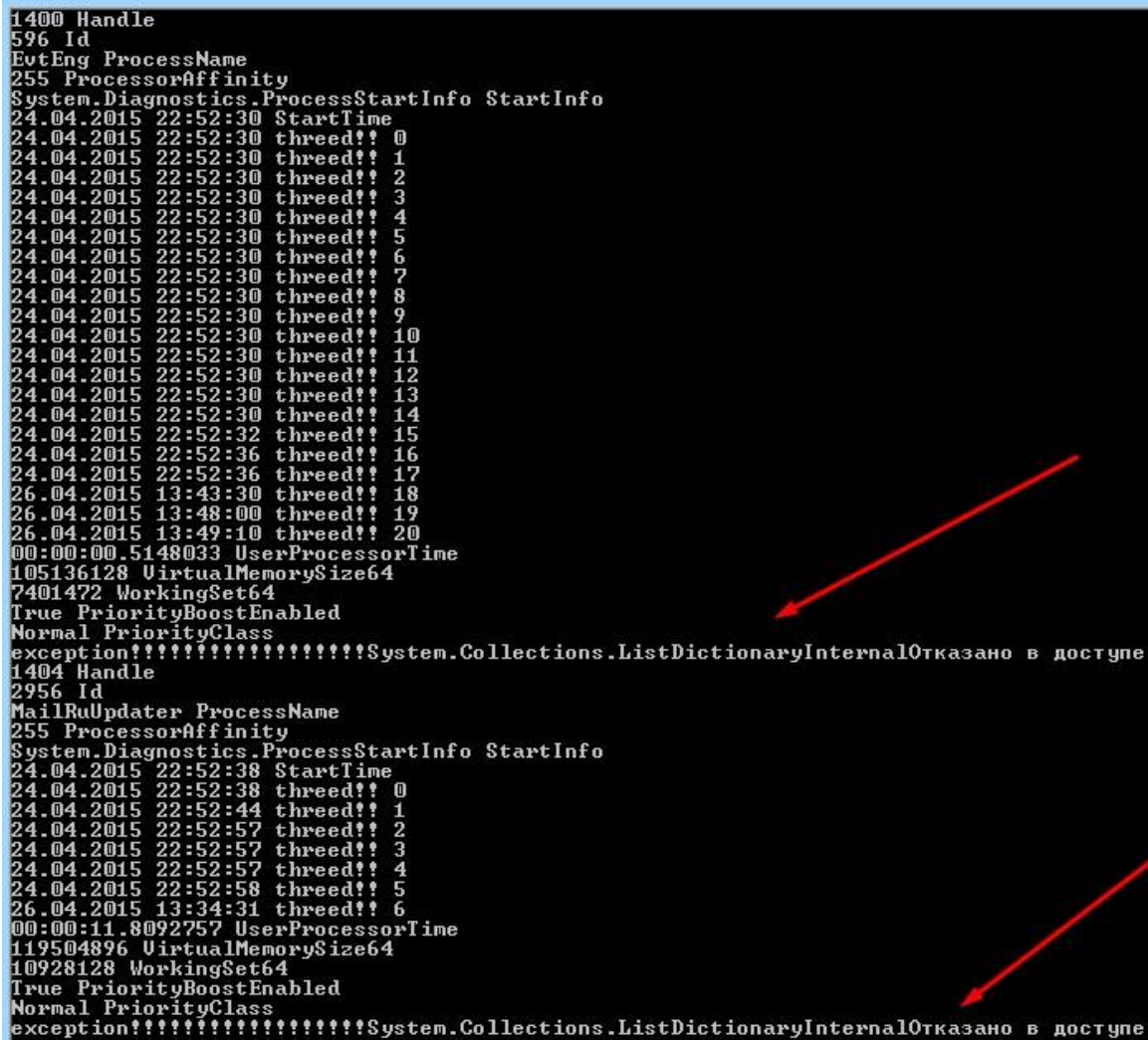


Рисунок 2. Демонстрация работы тестового консольного приложения

Следующая сборка «Excel DAO» служит для того, чтобы вести статистику приложения. При необходимости отлавливать неустойчивости приложения.

Класс, представляющий доступ для записи в Excel-файл, который использует стороннюю библиотеку, работающую через COM-объект.

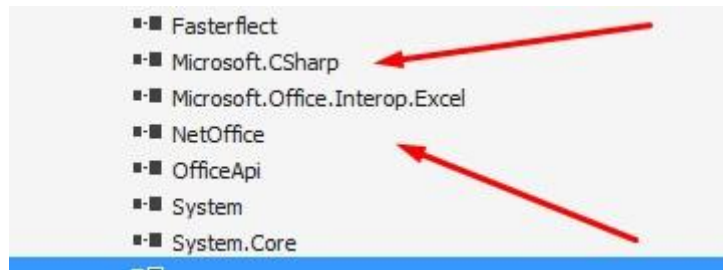


Рисунок 3. Ссылки

Исходный код файла представляет собой:

```
using System;
using System.Collections.Generic;

namespace ExcelDAO
{
    public class ExcelFile : IDisposable
    {
        private readonly Microsoft.Office.Interop.Excel.Application _application; private
        readonly Microsoft.Office.Interop.Excel.Workbook _workbook; private
        Microsoft.Office.Interop.Excel.Worksheet _worksheet;
        private readonly string _filename;
        private int _currentSheetNumber;

        public bool Visible
        {
            get { return _application.Visible; }
            set { _application.Visible = value; }
        }

        public int UsedRows
        {
            get { return _worksheet.UsedRange.Rows.Count; }
        }

        public int UsedCols
        {
            get { return _worksheet.UsedRange.Columns.Count; }
        }

        public void NextSheet()
        {
            _currentSheetNumber++;
            _worksheet = _workbook.Worksheets[_currentSheetNumber];
        }

        public ExcelFile(string filePath)
```

```

{
    _application = new Microsoft.Office.Interop.Excel.Application();
    _workbook = _application.Workbooks.Open(filePath);
    _worksheet =
(Microsoft.Office.Interop.Excel.Worksheet)_workbook.Worksheets.Item[1];
    _filename = filePath;
    _currentSheetNumber = 1;
}
public void SetValue(string value, int i, int j)
{
    _worksheet.Cells[i, j] = value;
}

public string GetValue(int i, int j)
{
    var range = (Microsoft.Office.Interop.Excel.Range)_worksheet.Cells[i, j];
    return range.Value2 != null ? range.Value2.ToString() : string.Empty;
}

public void Dispose()
{
    try
    {
        _workbook.Close(true, _filename, System.Reflection.Missing.Value);
        _application.Quit();

        System.Runtime.InteropServices.Marshal.ReleaseComObject(_application);
    }
    catch (Exception)
    {
    }
}

List<string[]> logs = new List<string[]>();
public void AddLogs(List<string[]> ar)
{
    int count = 1;
    ar.ForEach(x =>
    {
        for (int i = 0; i < x.Length; i++)
        {
            this.SetValue(x[i], count, i+1);
        }
        count++;
    });
}

```

}

И, наконец, финальная сборка, это «Managing processes». Сборка включает в себя весь графический интерфейс приложения, а также отдельных окон, которые созданы для удобства работы. Первое – это главное окно приложения.

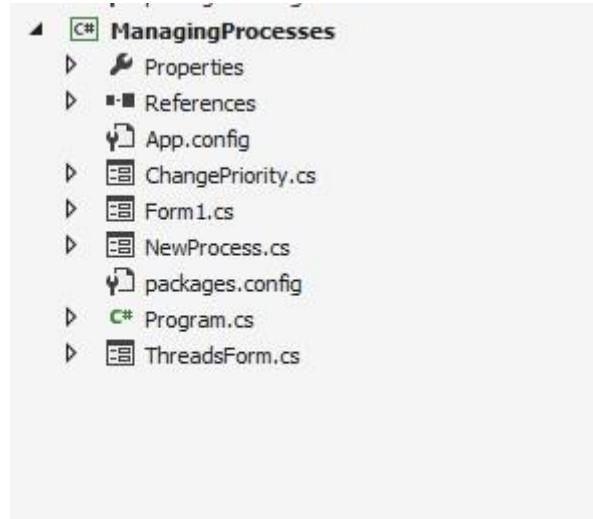


Рисунок 4. Проект «Managing Processes»

Класс Form1

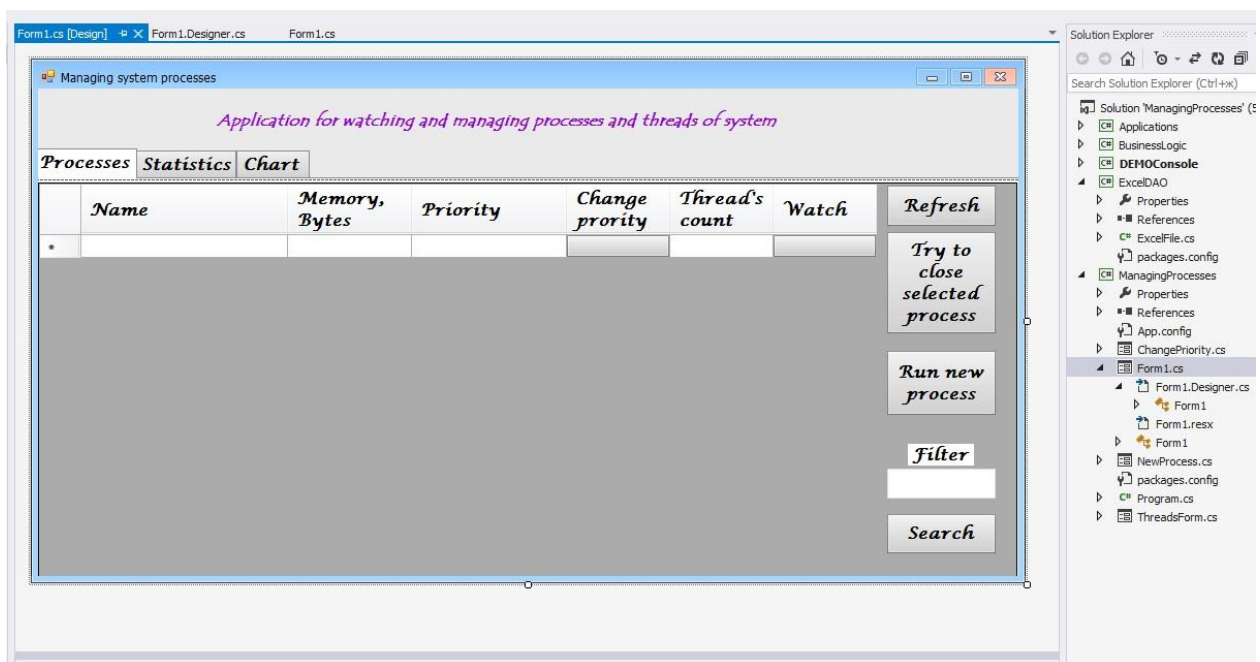


Рисунок 5. Главное окно программы

Исходный код класса включает в себя обработку всех событий, происходящих в данном окне. А также, используя сборку «ExcelDAO» контролирует запись статистику через определенный промежуток времени в файл log.xlsx:

```

using System;
using System.Collections.Generic; using System.ComponentModel; using
System.Data;
using System.Diagnostics; using System.Drawing; using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using BusinessLogic;
using ExcelDAO;

namespace ManagingProcesses
{
    public partial class Form1 : Form
    {
        private const string LogFile = @"Log\" + "logger.xlsx";
        private static int i = 1;
        private int _counter = 1;
        ManageProcess manage = new ManageProcess();
        PerformanceCounter cpucounter = new PerformanceCounter("Processor", "%
Processor Time", "_Total");
        PerformanceCounter memcounter = new PerformanceCounter("Memory", "%
Committed Bytes In Use");
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            FillFormWithProcesses(); Timer timer = new Timer();
            timer.Tick += new EventHandler(timer_Tick);
            timer.Interval = 1000;
            timer.Start();

            Timer timer2 = new Timer();
            timer2.Tick += new EventHandler(timer_TickForLog);
            timer2.Interval = 2000;
            timer2.Start();

            chart2.Series.First().XValueMember = "X"; chart2.Series.First().YValueMembers =
            "Y"; chart2.Series[0].Name = "% загрузки памяти, динамика";
            chart2.Series[0].ShadowColor = BackColor;
            chart2.BackColor = Color.Aquamarine;

```

```
chart2.BackSecondaryColor = Color.Green; chart2.ForeColor = Color.Aquamarine;  
chart2.Series[0].Color = Color.Crimson;
```

```
chart1.Series.First().XValueMember = "X"; chart1.Series.First().YValueMembers =  
"Y"; chart1.Series[0].Name = "% загруженности ЦП, динамика";  
chart1.Series[0].ShadowColor = BackColor;  
chart1.BackColor = Color.Aquamarine; chart1.BackSecondaryColor = Color.Green;  
chart1.ForeColor = Color.Aquamarine; chart1.Series[0].Color = Color.BlueViolet;  
}
```

```
void timer_Tick(object sender, EventArgs e)
```

```
{  
var cp = cpucounter.NextValue();  
chart1.Series[0].Points.AddXY(i, cp);
```

```
var m = memcounter.NextValue();  
chart2.Series[0].Points.AddXY(i, m);
```

```
i++; Invalidate();  
}
```

```
List<string[]> logs = new List<string[]>();
```

```
void timer_TickForLog(object sender, EventArgs e)
```

```
{  
var m = manage.GetProcesses().Select(x => x.VirtualMemorySize64 /  
100000).Aggregate((x,y) => x+y);  
var numberOfProcesses = manage.GetProcesses().Count.ToString();  
var time = DateTime.Now.ToLongTimeString();  
logs.Add(new string[] { "Следующее действие. Подробная информация:" });  
logs.Add(new string[] { "Время : ", time });  
logs.Add(new string[] { "Количество процессов : ", numberOfProcesses + "  
processes" });  
logs.Add(new string[] { "Загруженность памяти : ", m + " kb" });  
}
```

```
private void RefreshButton_Click(object sender, EventArgs e)
```

```
{  
FillFormWithProcesses();  
}
```

```
private void dataGridProcesses_CellContentClick(object sender,  
DataGridViewCellEventArgs e)
```

```
{  
if (e.ColumnIndex == 3 && e.RowIndex >= 0)  
{  
var name = dataGridProcesses["ProcessName", e.RowIndex].Value.ToString();  
try
```

```

{
    new
    ChangePriority(Process.GetProcessesByName(name).First()).Show();
}
catch (Exception ex)
{
    MessageBox.Show("Request rejected because of " + ex.Message);
}
}
if (e.ColumnIndex == 5 && e.RowIndex >= 0)
{
    var name = dataGridProcesses["ProcessName", e.RowIndex].Value.ToString();
    try
    {
        new
        ThreadsForm(Process.GetProcessesByName(name).First()).Show();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Request rejected because of " + ex.Message);
    }
}

}

#region Logic methods
private void FillFormWithProcesses(List<Process> list = null)
{
    dataGridProcesses.Rows.Clear();
    if (list == null)
        list = manage.GetProcesses();
    for (int i = 0; i < list.Count; i++)
    {
        dataGridProcesses.Rows.Add();
        try
        {
            dataGridProcesses["ProcessName", i].Value = list[i].ProcessName;
        }
        catch (Exception)
        {
            dataGridProcesses["ProcessName", i].Value = "Rejected";
            dataGridProcesses["ProcessName", i].Style.BackColor = Color.Crimson;
        }
        try
        {
            dataGridProcesses["Priority", i].Value =
            list[i].PriorityClass.ToString();

```

```

    }
    catch (Exception)
    {
        dataGridProcesses["Priority", i].Value = "Rejected";
        dataGridProcesses["Priority", i].Style.BackColor = Color.Crimson;
    }
    try
    {
        dataGridProcesses["NumberOfThreads", i].Value =
            list[i].Threads.Count;
    }
    catch (Exception)
    {
        dataGridProcesses["NumberOfThreads", i].Value = "Rejected";
        dataGridProcesses["NumberOfThreads", i].Style.BackColor = Color.Crimson;
    }
    try
    {
        dataGridProcesses["Memory", i].Value = list[i].PagedMemorySize64;
    }
    catch (Exception)
    {
        dataGridProcesses["Memory", i].Value = "Rejected";
        dataGridProcesses["Memory", i].Style.BackColor = Color.Crimson;
    }

    dataGridProcesses["WatchThreads", i].Value = "Threads";
    dataGridProcesses["ChangePriority", i].Value = "change";
}
}
#endregion

private void button1_Click(object sender, EventArgs e)
{
    var text = textBox1.Text;
    textBox1.Text = "";
    if (text.Length == 0) FillFormWithProcesses();
    var filteredList = manage.GetProcesses().Where((x) =>
        x.ProcessName.Contains(text)).ToList(); FillFormWithProcesses(filteredList);
}

private void button2_Click(object sender, EventArgs e)
{
    var name =
        dataGridProcesses["ProcessName", dataGridProcesses.SelectedCells[0].RowIndex]
        .Value.ToString();

```

```

try
{
var process = manage.GetProcesses().FirstOrDefault(x =>
x.ProcessName == name);
if (process != null)
{
process.Kill();
}
}
catch (Exception ex)
{
MessageBox.Show("Process was not closed because of " +
ex.Message);
}

private void TabControlProcess_Click(object sender, EventArgs e)
{

}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
}

private void button4_Click(object sender, EventArgs e)
{

using (
var excel =
new ExcelFile(LogFile)
)
{
excel.AddLogs(logs);
}
Process.Start(LogFile);
}

private void button3_Click(object sender, EventArgs e)
{
new NewProcess().Show();
}
}
}

```


Следующий класс – класс изменения приоритета процесса. Код окна представлен ниже:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics; using System.Drawing; using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ManagingProcesses
{
    public partial class ChangePriority : Form
    {
        private Process _process;
        public ChangePriority(Process process)
        {
            this._process = process;

            InitializeComponent(); CheckForNeedButton(_process.PriorityClass);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (radioButton1.Checked)
            {
                _process.PriorityClass = ProcessPriorityClass.Normal;
            }
            else if (radioButton2.Checked)
            {
                _process.PriorityClass = ProcessPriorityClass.AboveNormal;
            }
            else if (radioButton3.Checked)
            {
                _process.PriorityClass = ProcessPriorityClass.BelowNormal;
            }
            else if (radioButton4.Checked)
            {
                _process.PriorityClass = ProcessPriorityClass.High;
            }
            else if (radioButton5.Checked)
            {
                _process.PriorityClass = ProcessPriorityClass.Idle;
            }
        }
    }
}
```

```

else
{
    _process.PriorityClass = ProcessPriorityClass.RealTime;
}
this.Close();
}

private bool CheckForNeedButton(ProcessPriorityClass priority)
{
    if (priority == ProcessPriorityClass.Normal)
    {
        radioButton1.Checked = true;
        return true;
    }
    else if (priority == ProcessPriorityClass.AboveNormal)
    {
        radioButton2.Checked = true;
        return true;
    }
    else if (priority == ProcessPriorityClass.BelowNormal)
    {
        radioButton3.Checked = true;
        return true;
    }
    else if (priority == ProcessPriorityClass.High)
    {
        radioButton4.Checked = true;
        return true;
    }
    else if (priority == ProcessPriorityClass.Idle)
    {
        radioButton5.Checked = true;
        return true;
    }
    else if (priority == ProcessPriorityClass.RealTime)
    {
        radioButton6.Checked = true;
        return true;
    }
    return false;
}
}
}

```

Вот таким образом мы можем изменить приоритет процесса. Как видно исходные данные передаются через конструктор по ссылке и таким образом мы изменяем настоящий объект, а не его копию.

Класс для представления потоков отдельного процесса:

```
using System;
using System.Collections.Generic; using System.ComponentModel; using
System.Data;
using System.Diagnostics; using System.Drawing; using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ManagingProcesses
{
    public partial class ThreadsForm : Form
    {
        private Process _process;
        public ThreadsForm(Process process)
        {
            this._process = process; InitializeComponent(); dataGridView1.Rows.Clear();
            for (int i = 0; i < process.Threads.Count; i++)
            {
                dataGridView1.Rows.Add();
                dataGridView1["ThreadId", i].Value = process.Threads[i].Id;
                dataGridView1["StartTime", i].Value =
                    process.Threads[i].StartTime.ToLongDateString() +
                    process.Threads[i].StartTime.ToLongTimeString();
                dataGridView1["ThreadPriority", i].Value =
                    process.Threads[i].CurrentPriority;
                dataGridView1["ThreadWaitReason", i].Value =
                    process.Threads[i].WaitReason;
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Как видно, аналогично классу изменения приоритета, процесс передается по ссылке в конструктор.

И, наконец, класс, представляющий логику для запуска нового процесса:

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

namespace ManagingProcesses
{
    public partial class NewProcess : Form
    {
        private string _file;
        public NewProcess()
        {
            InitializeComponent();
            button2.Hide();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int size = -1;
            DialogResult result = openFileDialog1.ShowDialog(); // Show the dialog. if (result
== DialogResult.OK) // Test result.
            {
                _file = openFileDialog1.FileName;
                try
                {
                    button2.Show();
                    button2.Text = button2.Text + " " + _file.Split('/').Last();
                }
                catch (IOException)
                {
                }
            }

            private void button2_Click(object sender, EventArgs e)
            {
                try
                {
                    Process.Start(_file);
                }
                catch (Exception)
```

```
{  
    MessageBox.Show("Файл данного типа нельзя запустить");  
}  
this.Close();  
}  
}  
}
```

с. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В работе главное окно выглядит следующим образом:

Managing system processes

Application for watching and managing processes and threads of system

ProcessesStatisticsChart

	Name	Memory, Bytes	Priority	Change prority	Thread's count	Watch	Refresh
▶	chrome	89550848	Normal	change	8	Threads	Try to close selected process
	SynTPEnh	20578304	AboveNormal	change	9	Threads	
	UNS	12771328	Normal	change	6	Threads	
	wlanext	18210816	Normal	change	14	Threads	Run new process
	armsvc	1376256	Normal	change	4	Threads	
	wuaucflt	2957312	Normal	change	3	Threads	Filter
	chrome	63131648	Normal	change	11	Threads	
	VsHub	27418624	Normal	change	14	Threads	Search
	amigo	50614272	Normal	change	26	Threads	
	SkypeC2CPNRSvc	2154496	Normal	change	13	Threads	
	conhost	2097152	Normal	change	1	Threads	

Рисунок 6. Процессы

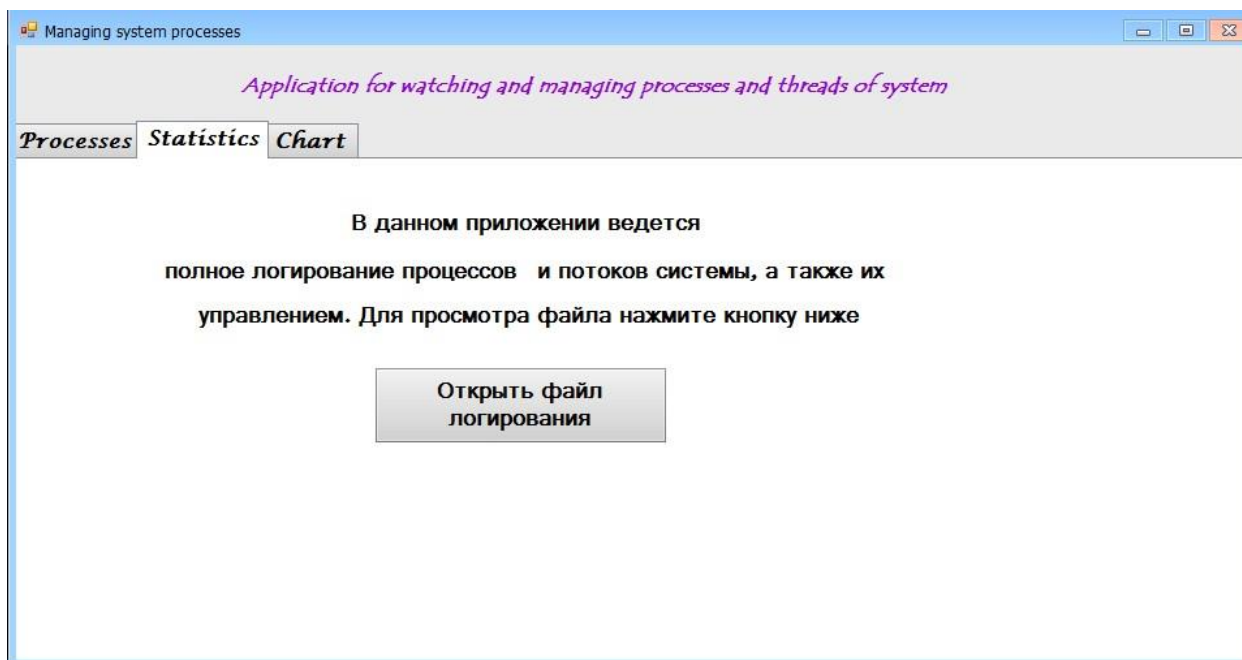


Рисунок 7. Логирование работы программы



Рисунок 8. Графики загрузки ЦП и памяти

Также при просмотре статистики из Excel-файла мы можем получить следующую информацию, содержащую точное время взятия данных, количество процессов на данный момент, загрузка памяти, а также загрузка центрального процессора.

13	Следующее действие. Подробная информация:		
14	Время :	13:58:43	
15	Количество процессов :	103 processes	
16	Загруженность памяти :	194506 kb	
17	Следующее действие. Подробная информация:		
18	Время :	13:58:45	
19	Количество процессов :	103 processes	
20	Загруженность памяти :	194511 kb	
21	Следующее действие. Подробная информация:		
22	Время :	13:58:47	
23	Количество процессов :	103 processes	
24	Загруженность памяти :	194452 kb	
25	Следующее действие. Подробная информация:		
26	Время :	13:58:49	
27	Количество процессов :	103 processes	
28	Загруженность памяти :	194477 kb	
29	Следующее действие. Подробная информация:		
30	Время :	13:58:51	
31	Количество процессов :	103 processes	
32	Загруженность памяти :	194463 kb	
33	Следующее действие. Подробная информация:		
34	Время :	13:58:53	
35	Количество процессов :	103 processes	

Рисунок 9. Файл статистики работы программы

Теперь рассмотрим функции, которыми обладает программа:

1) Обновление списка процессов на текущий момент:

Application for watching and managing processes and threads of system							
Processes	Statistics	Chart					
	Name	Memory, Bytes	Priority	Change priority	Thread's count	Watch	Refresh
▶	chrome	87560192	Normal	change	10	Threads	Try to close selected process
	SynTPEnh	20578304	AboveNormal	change	9	Threads	
	UNS	12771328	Normal	change	6	Threads	
	wlanext	18210816	Normal	change	14	Threads	Run new process
	armsvc	1376256	Normal	change	4	Threads	
	wuaucft	2957312	Normal	change	3	Threads	
	chrome	63131648	Normal	change	11	Threads	Filter
	VsHub	27484160	Normal	change	14	Threads	
	amigo	50778112	Normal	change	29	Threads	
	SkypeC3CPNRSvc	2154406	Normal	change	13	Threads	Search

Рисунок 10. Обновление состояния

2) Закрытие выбранного процесса. «Try» значит то, что доступ к некоторым процессам может быть отклонен системой. В этом случае мы можем получить ситуацию, показанную на картинке ниже

Application for watching and managing processes and threads of system

Processes	Statistics	Chart				
Name	Memory, Bytes	Priority	Change prority	Thread's count	Watch	Refresh
chrome	87560192	Normal	change	10	Threads	Try to close selected process
SynTPEnh	20578304	AboveNormal	change	9	Threads	
UNS	12771328	Normal	change	6	Threads	
wlanext	18210816	Normal	change	14	Threads	Run new process
armsvc	1376256	Normal	change	4	Threads	
wuaucit	2957312	Normal	change	3	Threads	
chrome	63131648	Normal	change	11	Threads	Filter
VsHub	27484160	Normal	change	14	Threads	
amigo	50778112	Normal	change	29	Threads	
SkypeC2CPNRSvc	2154496	Normal	change	13	Threads	Search
svchost	6664192	Normal	change	5	Threads	

Рисунок 11. Заккрытие процесса

Name	Bytes	Priority	prority	count	watch	Refresh
WmiPrvSE	14397440	Normal	change	6	Threads	Try to close selected process
smss	753664	Normal	change	2	Threads	
igfxtray	4882432	Normal	change	3	Threads	
BTFS.AmpPalSer...	11640832	Normal	change	8	Threads	Run new process
mysqld	601825280	Normal	change	26	Threads	
svchost	13398016	Normal	change	9	Threads	
EvtEng	17981440	Normal	change	21	Threads	Filter
System	163840	Rejected	change	156	Threads	
MailRuUpdater	16674816	Normal	change	7	Threads	
Idle	0	Rejected	change	8	Threads	Search

Рисунок 12. Отклонение системой

3) Фильтр по имени процесса.

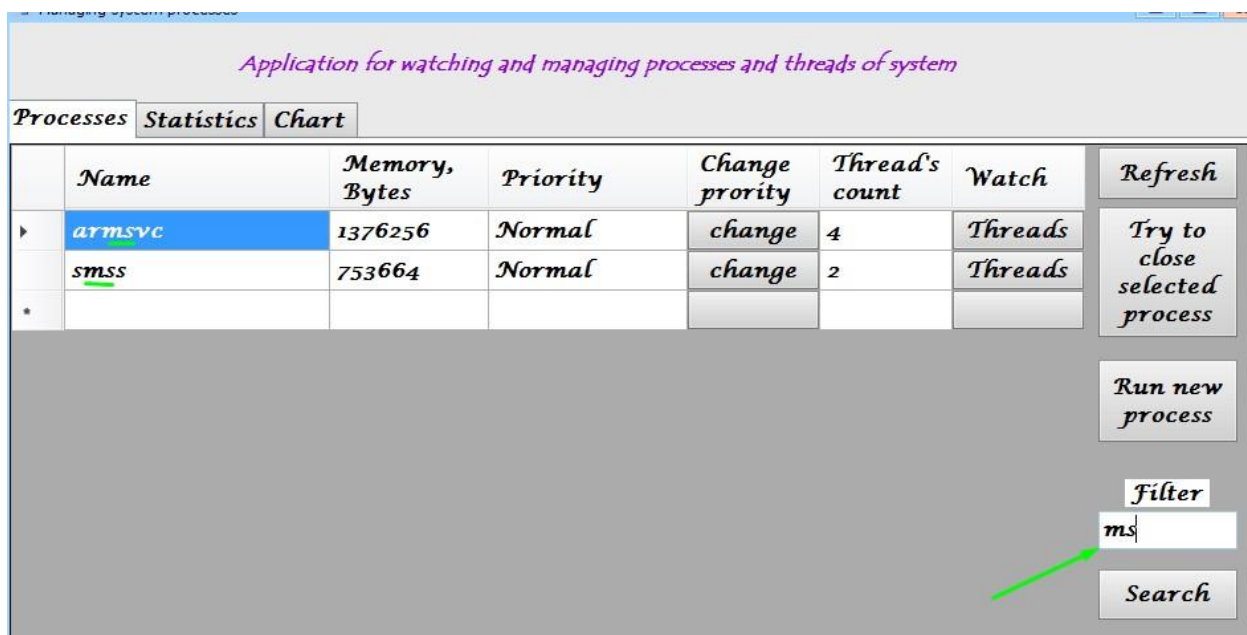


Рисунок 12. Поисковый фильтр

4) Также мы можем изменить приоритет каждого из процессов. Для этого достаточно кликнуть по кнопке «Change». Продемонстрируем пример:

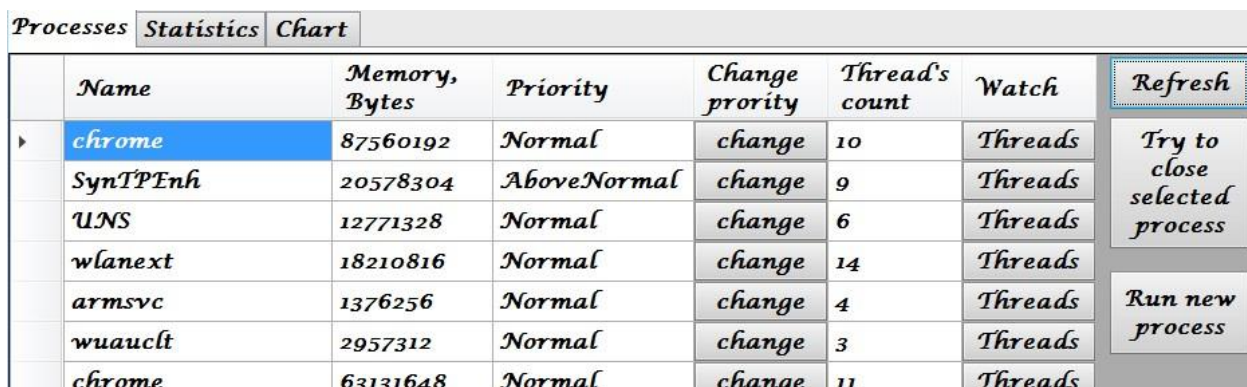


Рисунок 13. Изменение приоритета. Шаг 1

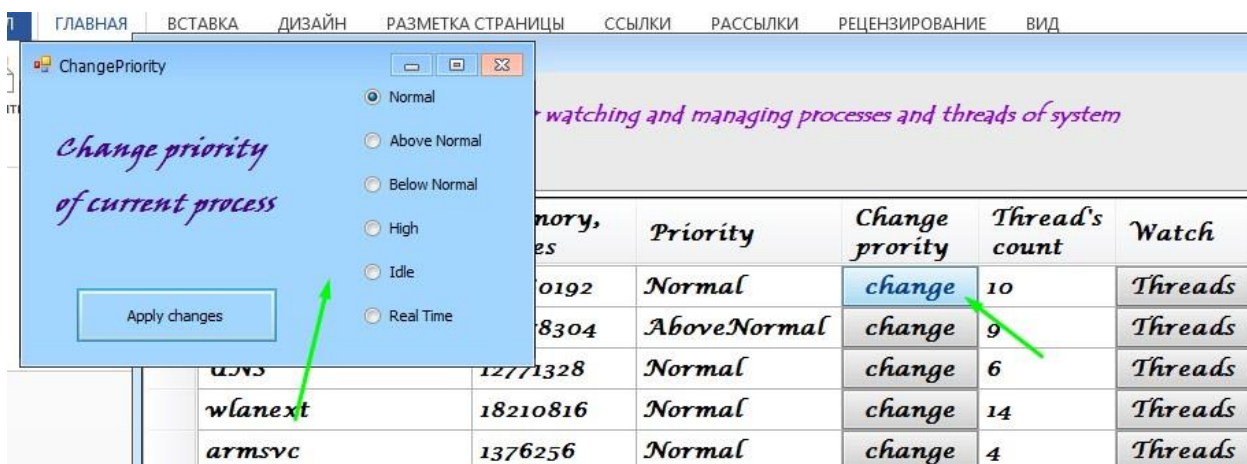


Рисунок 14. Изменение приоритета. Шаг 2

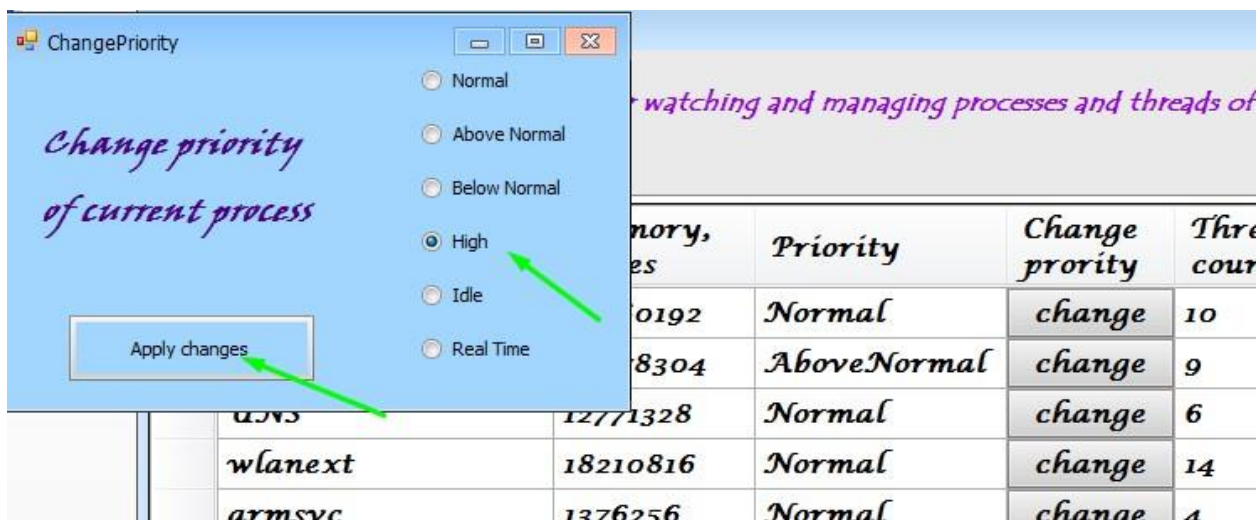


Рисунок 15. Изменение приоритета. Шаг 3

Name	Memory, Bytes	Priority	Change priority	Thread's count	Watch
chrome	87453696	High	change	8	Threads
SynTPEnh	20578304	Above Normal	change	9	Threads
UNS	12771328	Normal	change	6	Threads
wlanext	18210816	Normal	change	14	Threads
armsvc	1376256	Normal	change	4	Threads
wuaucft	2957312	Normal	change	3	Threads

Рисунок 16. Изменение приоритета. Шаг 4

5) Просмотр потоков, которые содержит определенный процесс:

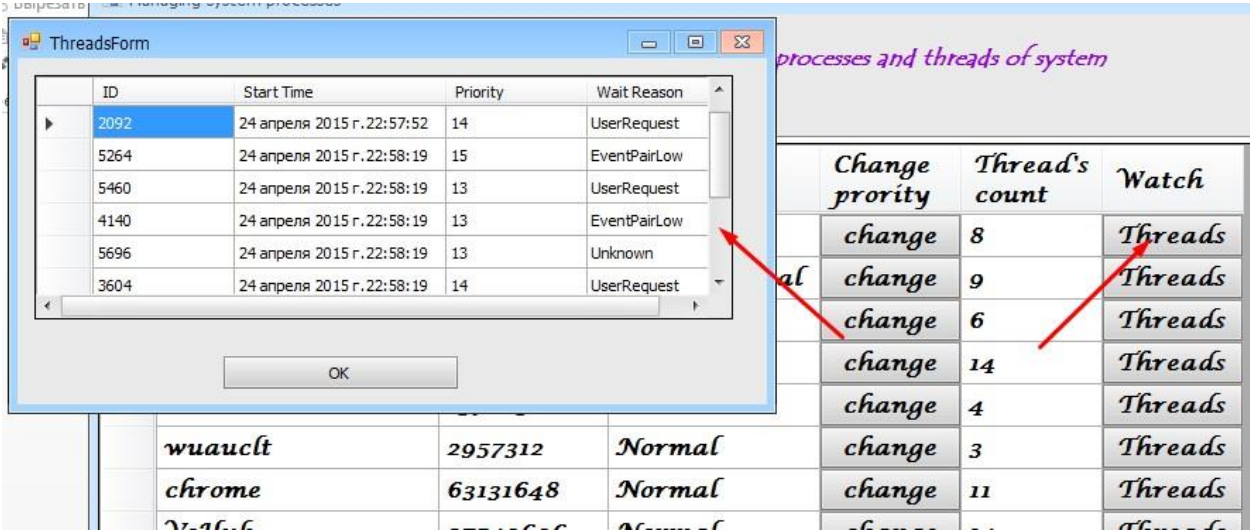


Рисунок 17. Просмотри потоков

Как видно мы можем получить подробную информацию об ID каждого потока, дате начала его работы, приоритете, а также ожидающего действия.

6) И, наконец, создание нового процесса. Продемонстрируем работу и с этим функционалом:

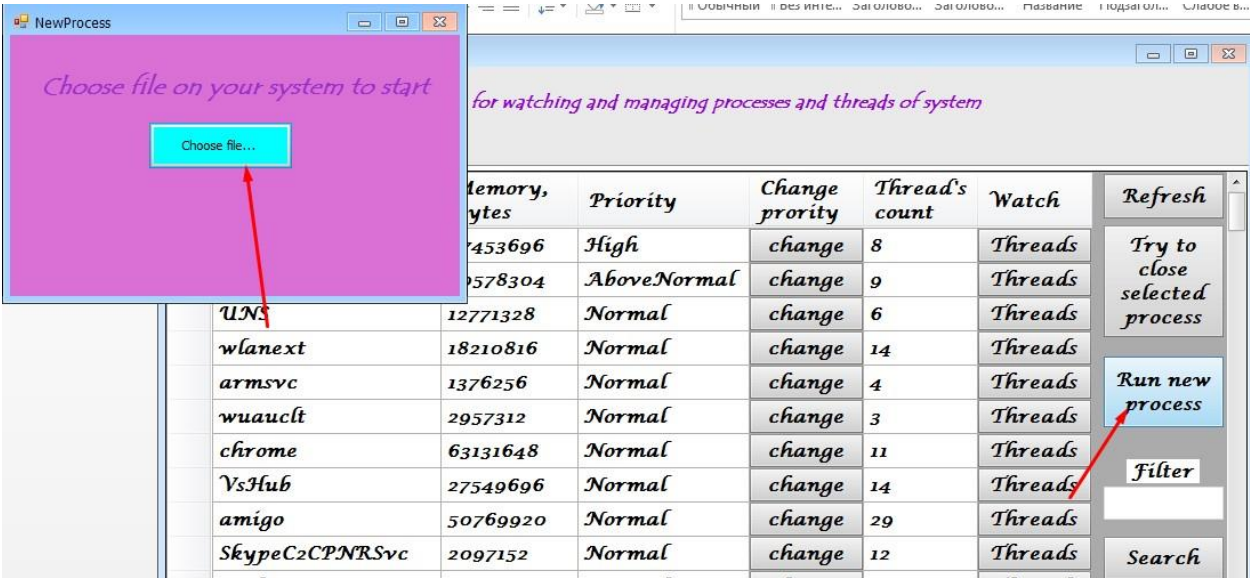


Рисунок 18. Запуск нового процесса. Шаг 1

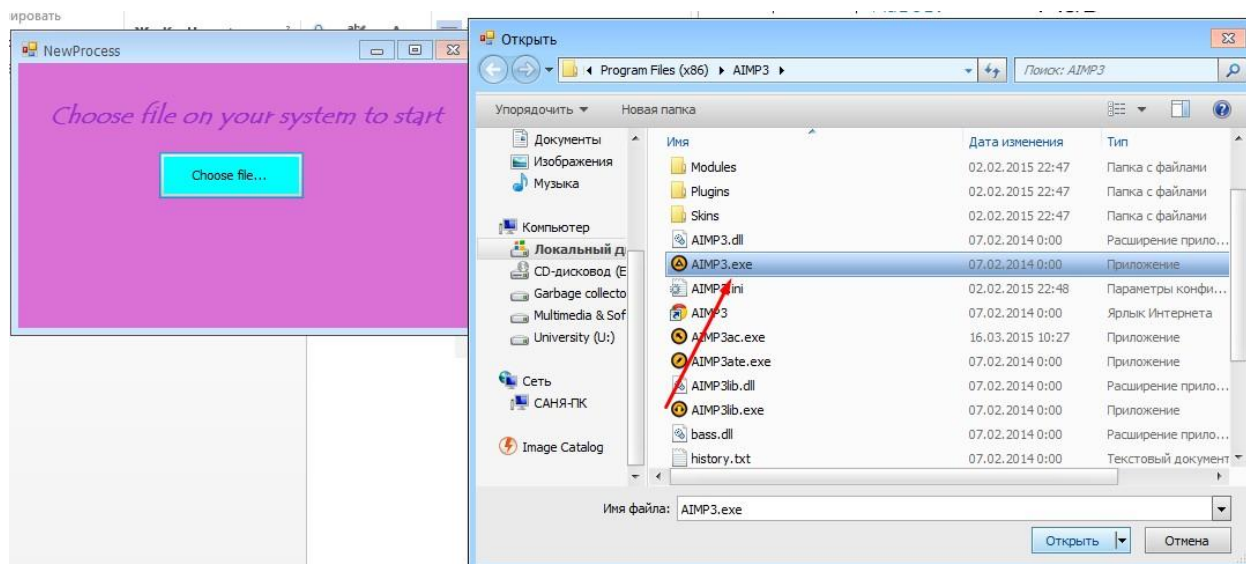


Рисунок 19. Запуск нового процесса. Шаг 2

При выборе приложения становится активной кнопка запуска процесса:

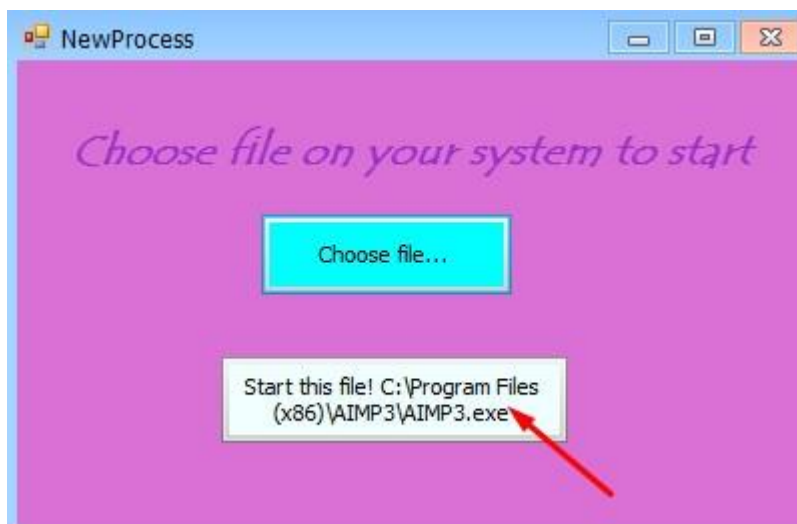


Рисунок 20. Запуск нового процесса. Шаг 3

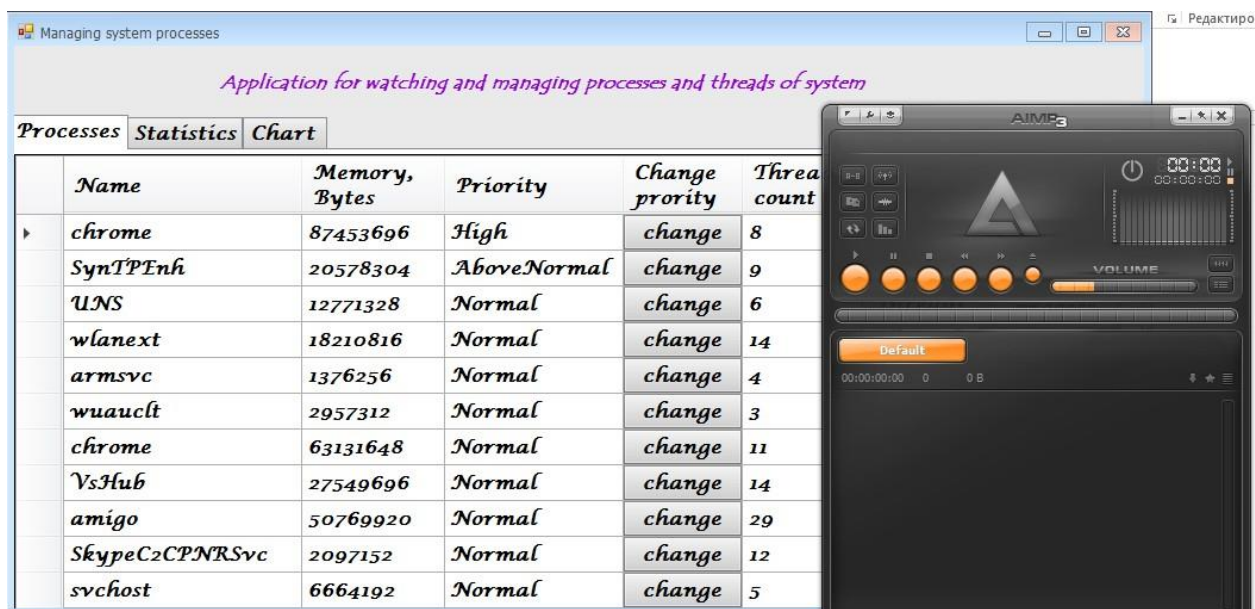


Рисунок 21. Запуск нового процесса. Шаг 4

3 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Дж. Рихтер, CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд.

[2] Ч. Петцольд, Программирование для Microsoft Windows 8. 6-е изд.

[3] [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://habrahabr.ru/>

[4] [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://msdn.microsoft.com/ru-ru/default.aspx>

[5] [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://rstdn.ru/>

