Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

МОНИТОРИНГ И УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ СИСТЕМЫ

Студент: гр. 253501 Янковский А.В.

Руководитель: Ассистент, магистр технических наук Сергейчик В. В.

Минск 2015

**СОДЕРЖАНИЕ**

[1. Цель и задачи курсового проекта 3](#_Toc373928182)

[3. Содержание курсового проекта 4](#_Toc373928184)

[3.Список использованной литературы 27](#_Toc373928184)

**1 ЦЕЛИ И ЗАДАЧИ КУРСОВОГО ПРОЕКТИРОВАНИЯ**

Цели:

- закрепление и углубление теоретических и практических знаний по избранной специальности и применение их для решения конкретных задач;

- формирование навыков ведения самостоятельной проектно-конструкторской или исследовательской работы и овладение методикой проектирования или научного исследования и эксперимента;

- приобретение навыков обобщения и анализа результатов, полученных другими разработчиками или исследователями;

Задачи:

- построение архитектуры приложения

- реализация основного функционала для работы с процессами системы

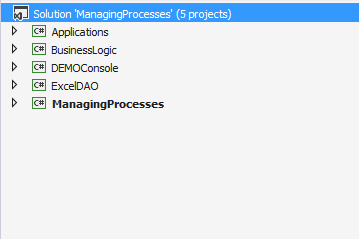
- построения удобного для пользователя графического интерфейса

- тестирование приложения

**2 СОДЕРЖАНИЕ КУРСОВОГО ПРОЕКТА**

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

Полное решение проекта представлено в виде 5 сборок.



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;

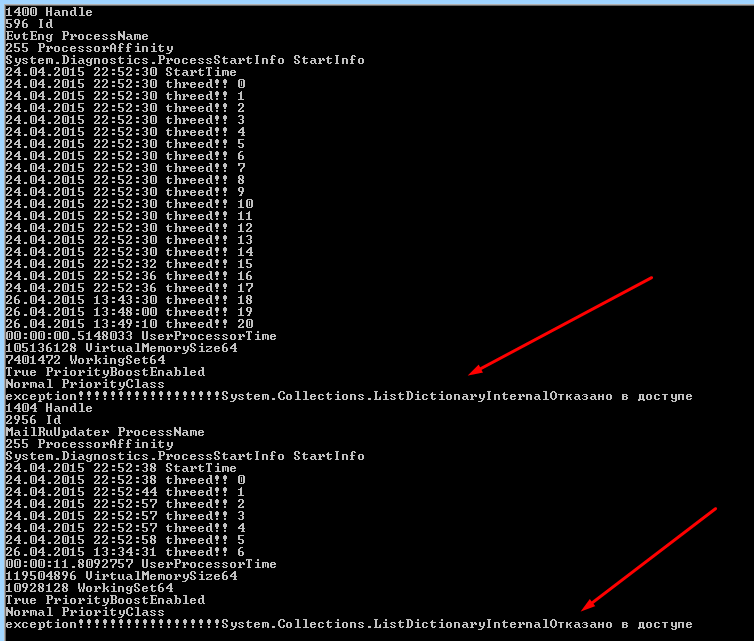
}

}

}

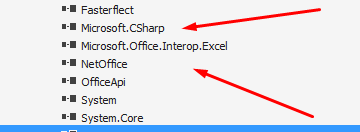
}

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



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

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



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++;

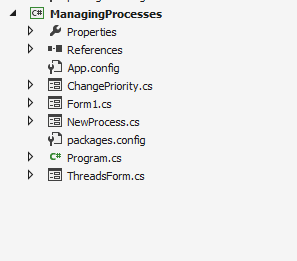
});

}

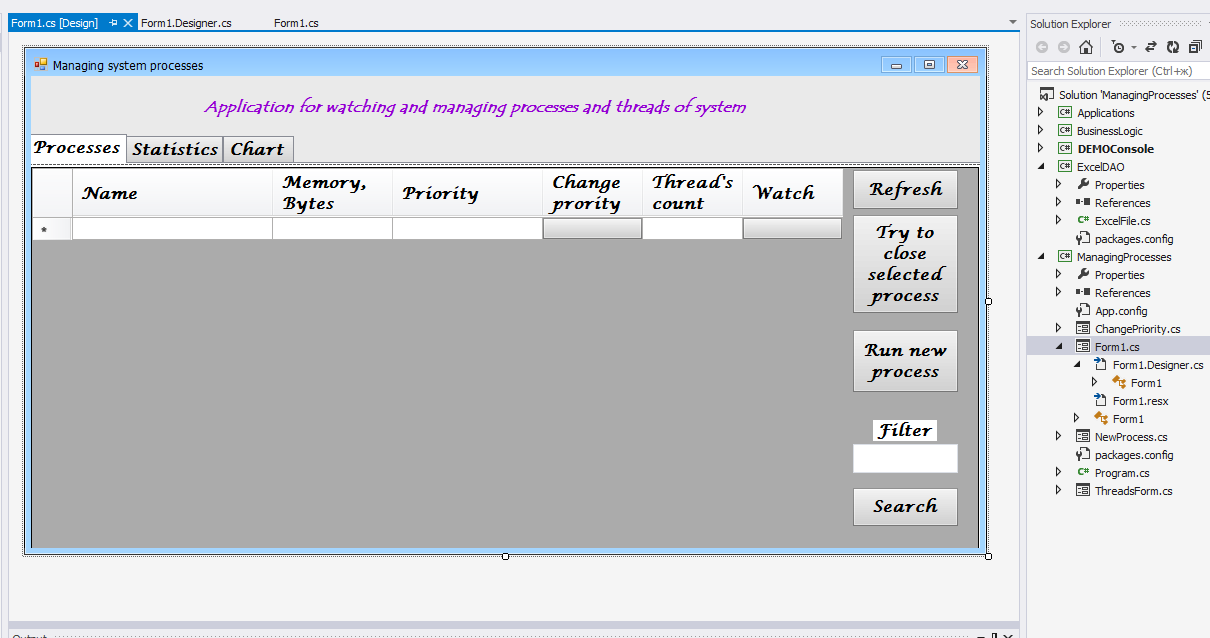
}

}

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.



Класс Form1



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 = "% загруженности 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 beacause 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)

{

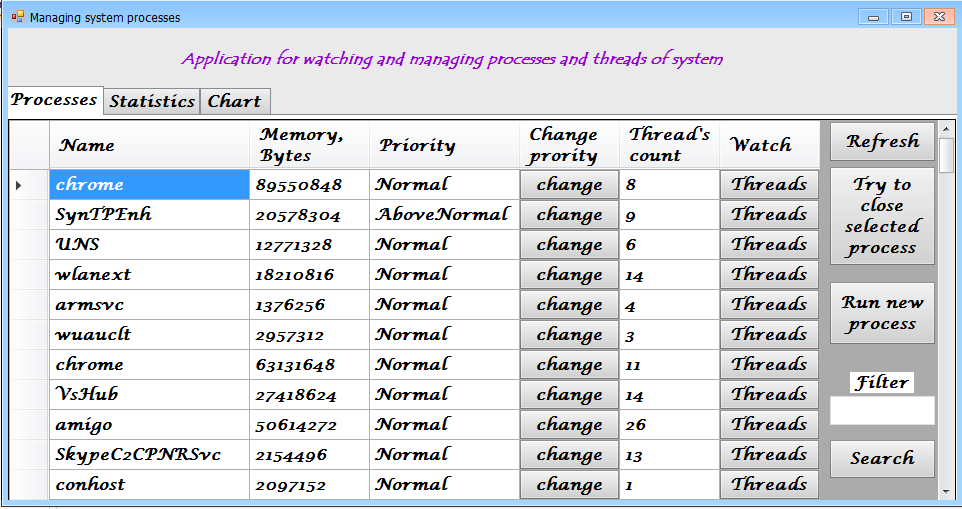
new NewProcess().Show();

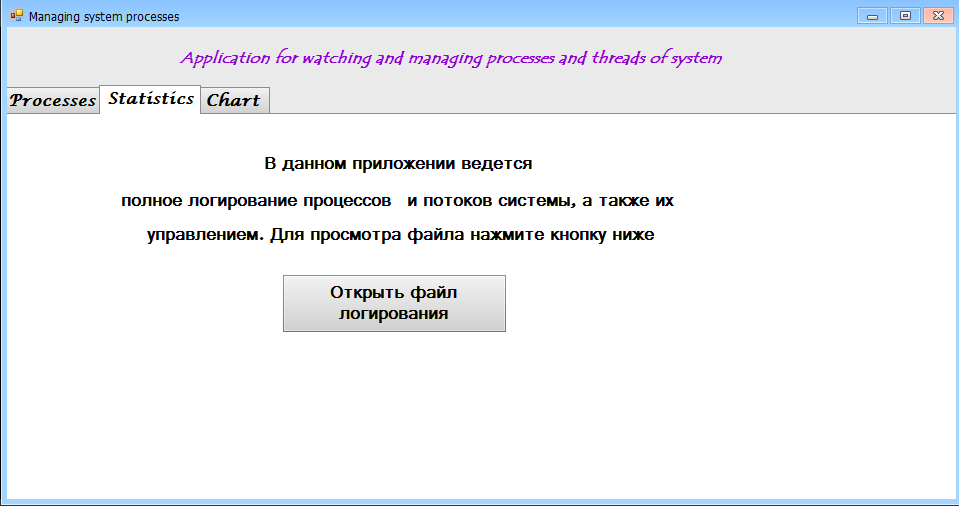
}

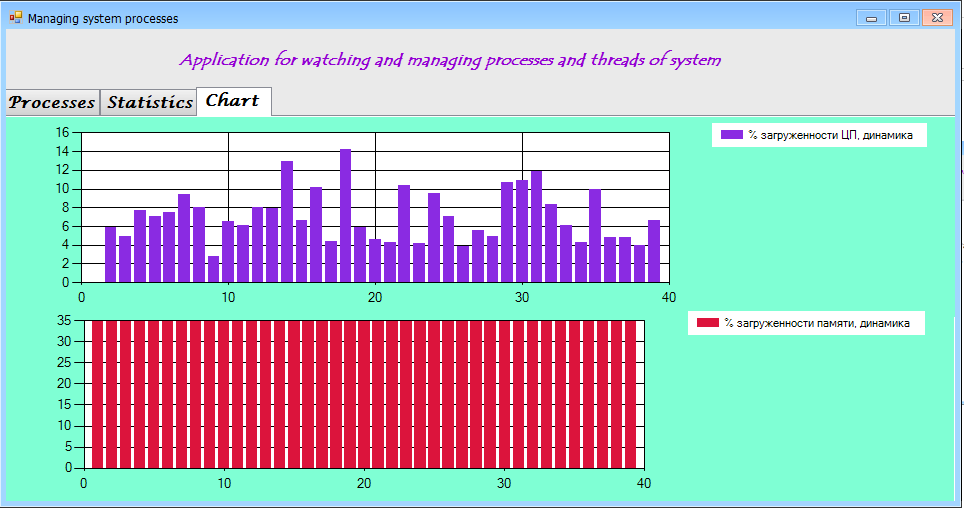
}

}

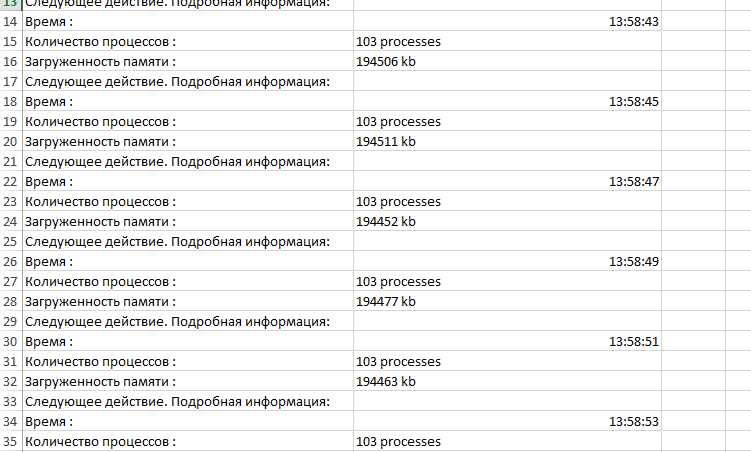
В работе главное окно выглядит следующим образом:





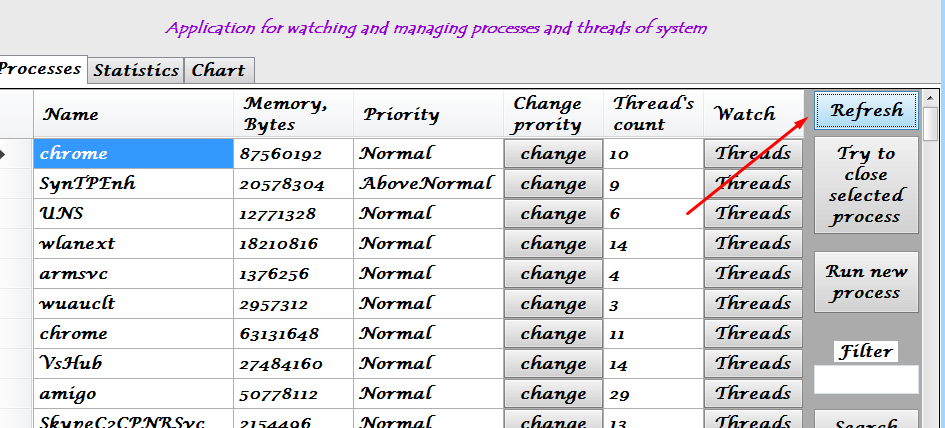


Также при просмотре статистики из Excel-файла мы можем получить следующую информацию, содержащую точное время взятия данных, количество процессов на данных момент, загруженность памяти, а также загруженность центрального процессора.

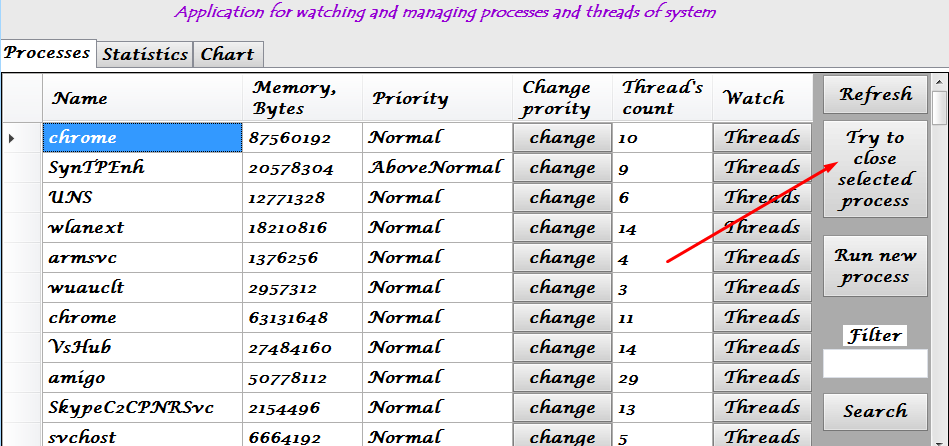


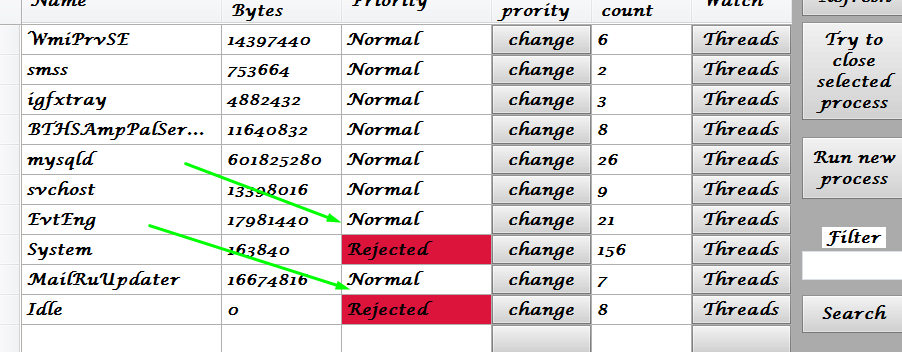
Теперь рассмотрим функции, которыми обладает программа:

1. Обновление списка процессов на текущий момент:

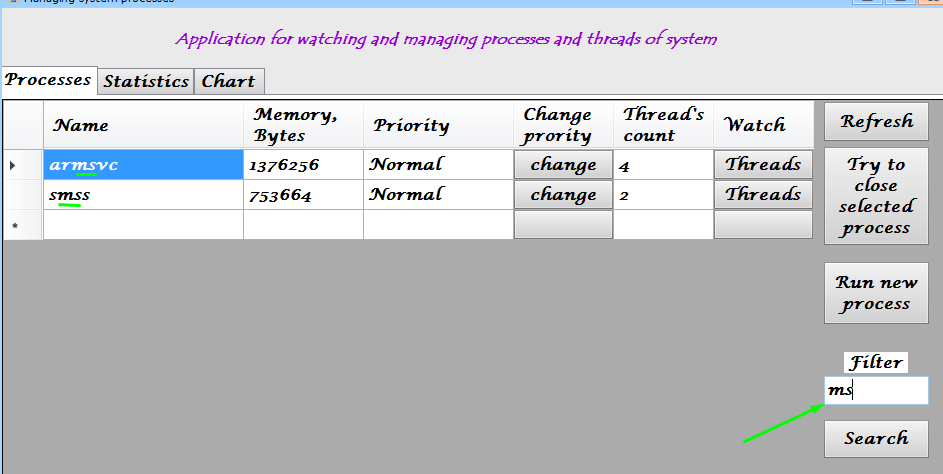


1. Закрытие выбранного процесса. «Try» значит то, что доступ к некоторым процессам может быть отклонен системой. В этом случае мы можем получить ситуацию, показанную на картинке ниже

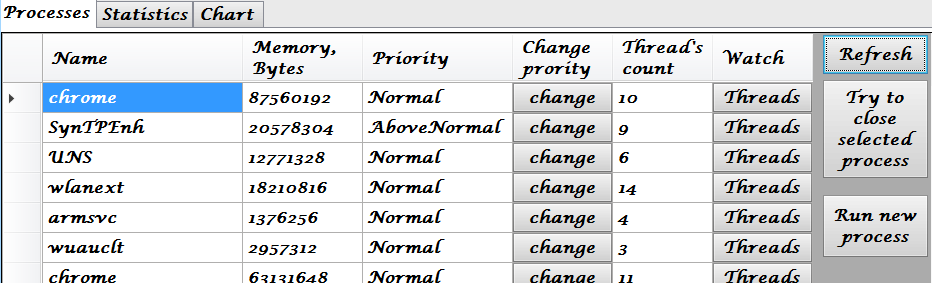


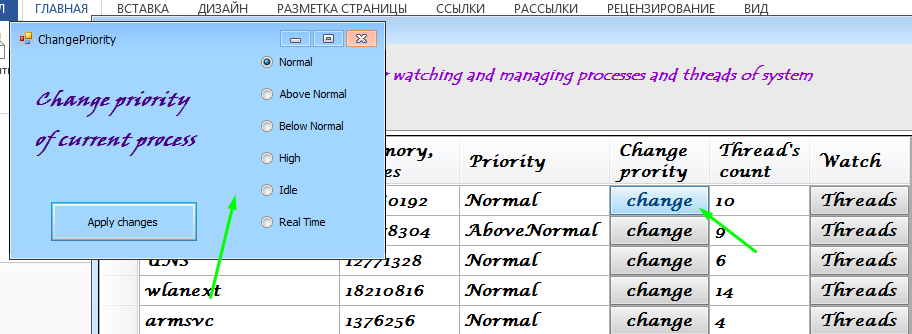


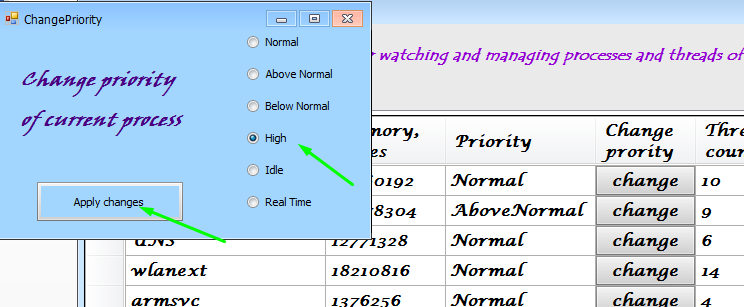
1. Фильтр по имени процесса.

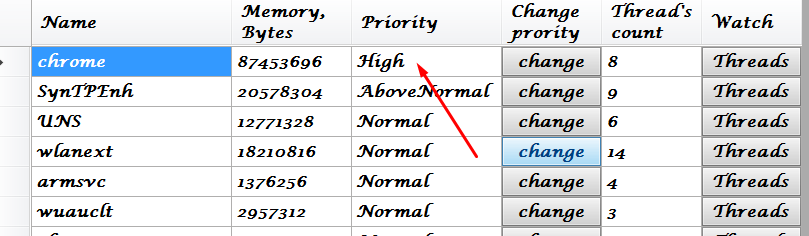


1. Также мы можем изменить приоритет каждого из процессов. Для этого достаточно кликнуть по кнопке «Change». Продемонстрируем пример:









Код данного окна изменения приоритета 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;

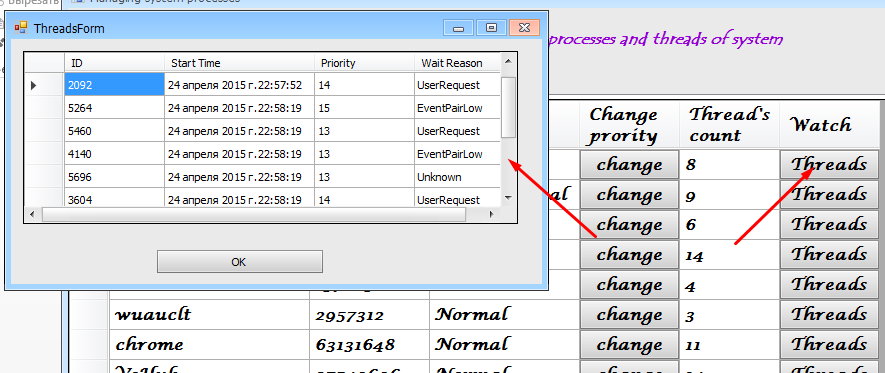
}

}

}

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.

1. Просмотр потоков, которые содержит определенный процесс:



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

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();

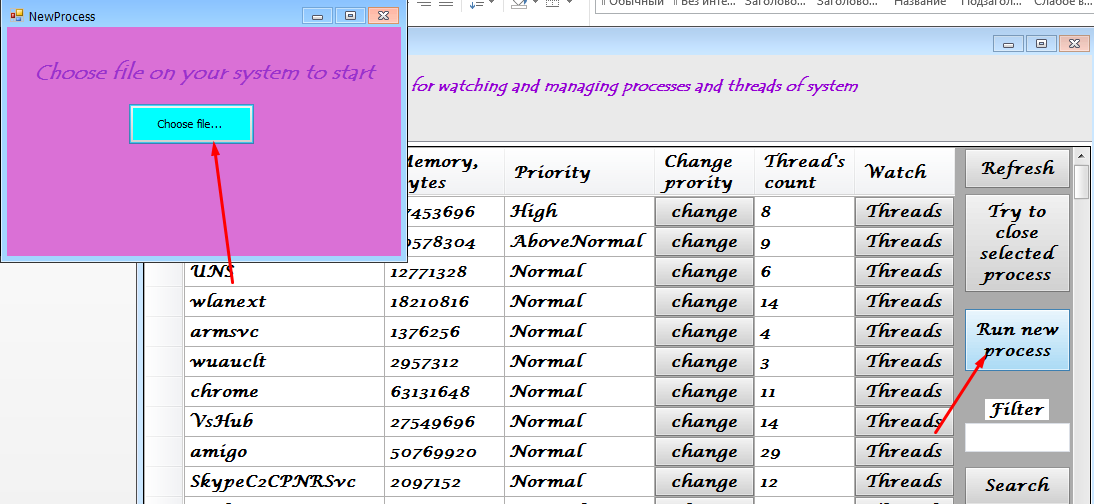
}

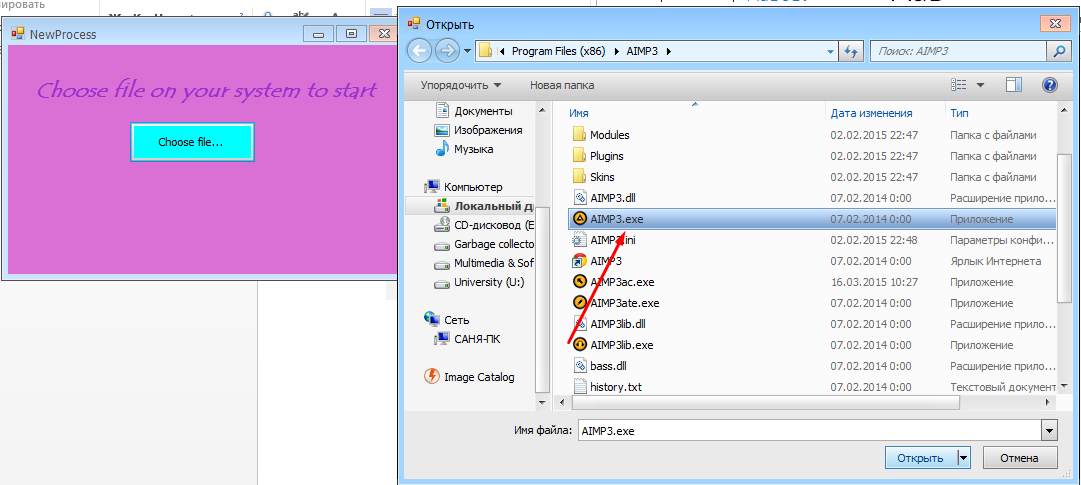
}

}

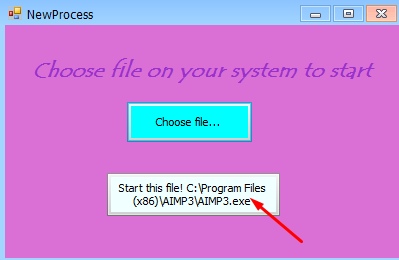
Как видно, аналогично классу изменения приоритета, процесс передается по ссылке в конструктор.

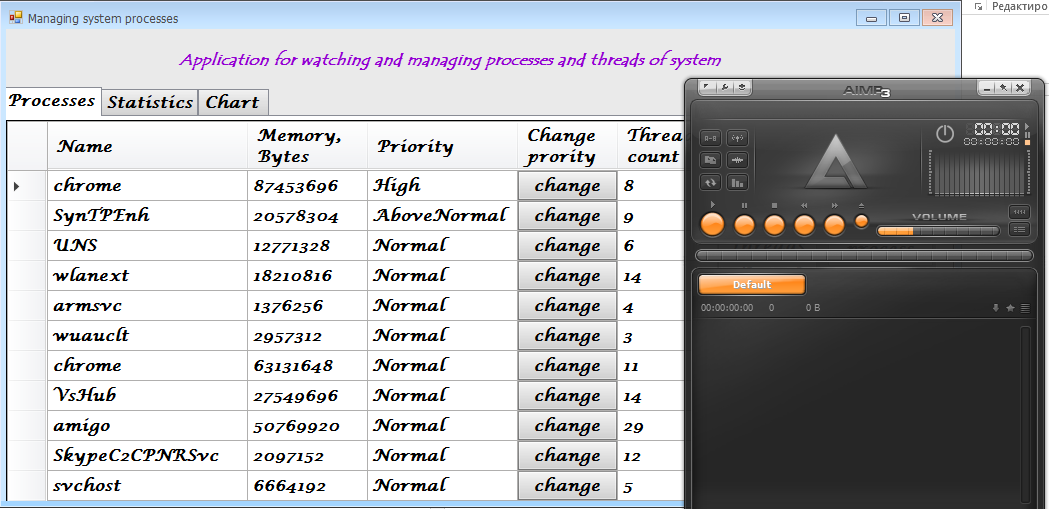
1. И, наконец, создание нового процесса. Продемонстрируем работу и с этим функционалом:





При выборе приложения становится активной кнопка запуска процесса:





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();

}

}

}

1. **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: <https://msdn.microsoft.com/ru-ru/default.aspx>

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