­Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Приложение «Диспетчер задач»

БГУИР КП 1-40 01 01 014  ПЗ

Выполнил

студент: гр. 151004 Дубовский А.В.

Проверил: Деменковец Д.В.

Минск 2023

содержание

[Введение 3](#_Toc153758842)

[1 Аналитический обзор литературы 4](#_Toc153758843)

[1.1 Обзор существующих аналогов 4](#_Toc153758844)

[1.2 Перечень функциональных требований 7](#_Toc153758845)

[1.3 Состав и параметры технических и программных средств 7](#_Toc153758846)

[2 Моделирование предметной области 8](#_Toc153758847)

[2.1 WinApi 8](#_Toc153758848)

[2.2 C#....................................................................................................................8](#_Toc153758849)

[3 Проектирование программного средства 9](#_Toc153758850)

[3.1 Структура программы 9](#_Toc153758851)

[3.2 Проектирование интерфейса программного средства 9](#_Toc153758852)

[3.3 Проектирование функционала программного средства 9](#_Toc153758853)

[4 Разработка программного средства 11](#_Toc153758854)

[4.1 Обновление информации о глобальном использовании ЦП 11](#_Toc153758855)

[4.2 Получение информации о использовании ЦП процессам 11](#_Toc153758856)

[4.3 Получение имени пользователя создавшего процесса 12](#_Toc153758857)

[5 Тестирование программного средства 13](#_Toc153758858)

[6 Руководство пользователя 15](#_Toc153758859)

[6.1 Минимальные системные требования 15](#_Toc153758860)

[Заключение 16](#_Toc153758861)

[Список использованной литературы 17](#_Toc153758862)

[Приложение А 18](#_Toc153758863)

[Приложение Б 20](#_Toc153758864)

[Приложение В 23](#_Toc153758865)

[Приложение Г 27](#_Toc153758866)

# Введение

В эпоху информационных технологий, когда многозадачность стала обыденностью, управление задачами и процессами становится ключевым для эффективности. Диспетчер задач - это инструмент, контролирующий процессы в операционной системе, предоставляющий данные о производительности и фоновых службах.

Цель этого проекта - разработать функциональный диспетчер задач с применением Windows API, который будет отображать список текущих процессов, использование ЦПУ, использование памяти и другую полезную информацию.

Windows API (WinAPI) - это набор функций, предоставляемых операционной системой Windows, которые позволяют разработчикам взаимодействовать и манипулировать различными компонентами операционной системы.

# Аналитический обзор литературы

## Обзор существующих аналогов

### Диспетчер задач Windows 11

Диспетчер задач Windows 11 изображен на рисунках 1.1 и 1.2. Является наиболее известным диспетчером задач. Основной конкурент.

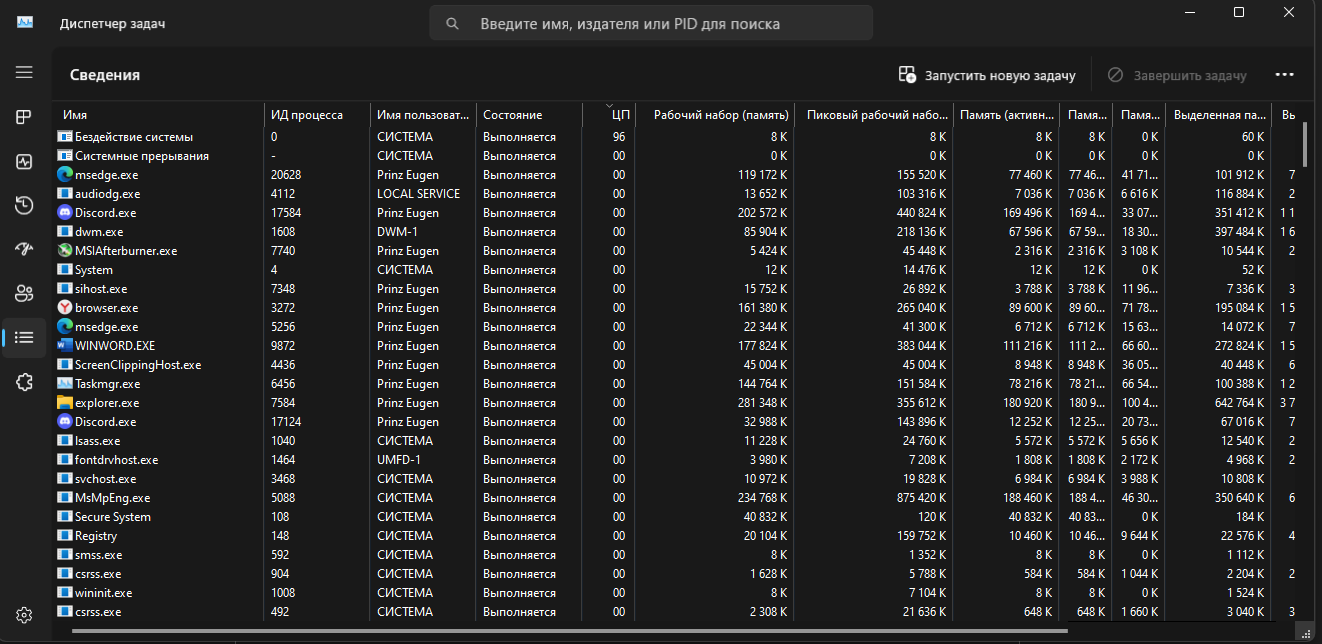


Рисунок 1.1 – Диспетчер задач Windows 11

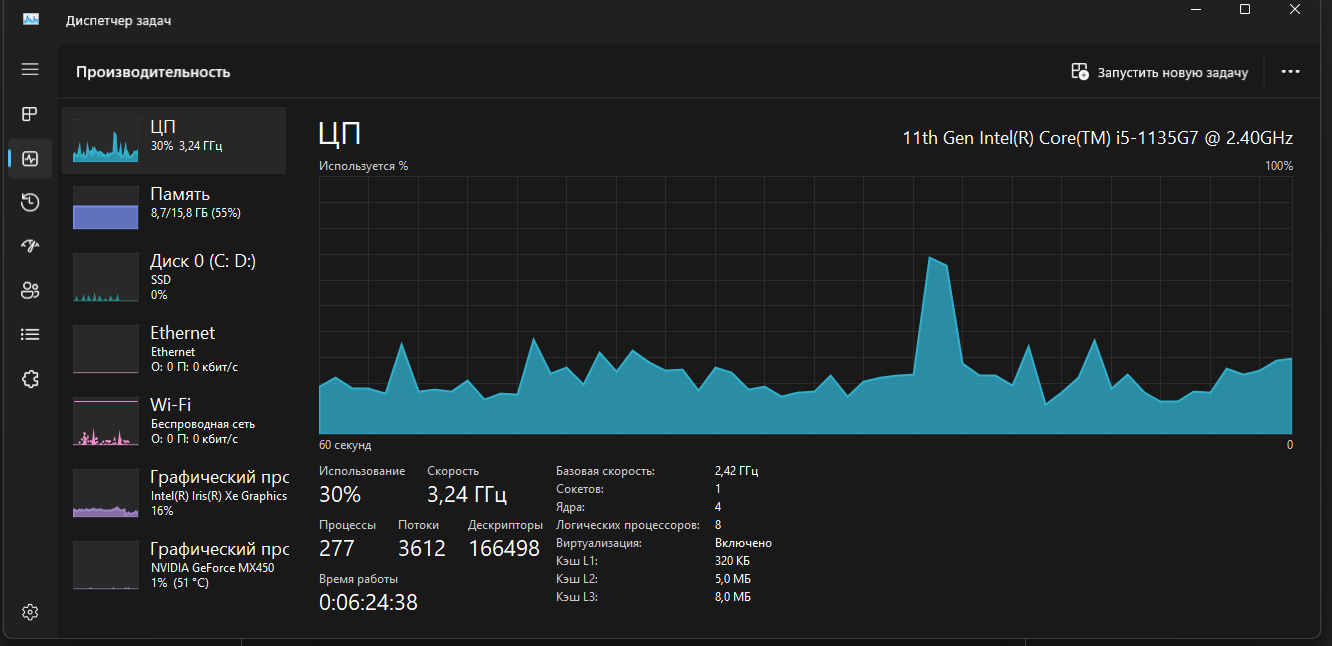


Рисунок 1.2 – Диспетчер задач Windows 11

Список выполняемых функций:

* Иформация о процессах;
* Графики производительности;
* Завершение процессов;
* Запуск новых процессов.

Рассмотрим достоинства данного приложения:

* Встроен в операционную систему, не требует дополнительной установки;
* Предоставляет детальную информацию о системе и процессах;
* Простой и понятный интерфейс;

Рассмотрим недостатки данного приложения:

* Интерфейс может быть сложным для новых пользователей.

### System Explorer

System Explorer изображен на рисунках 1.3. Является наиболее известным диспетчером задач.

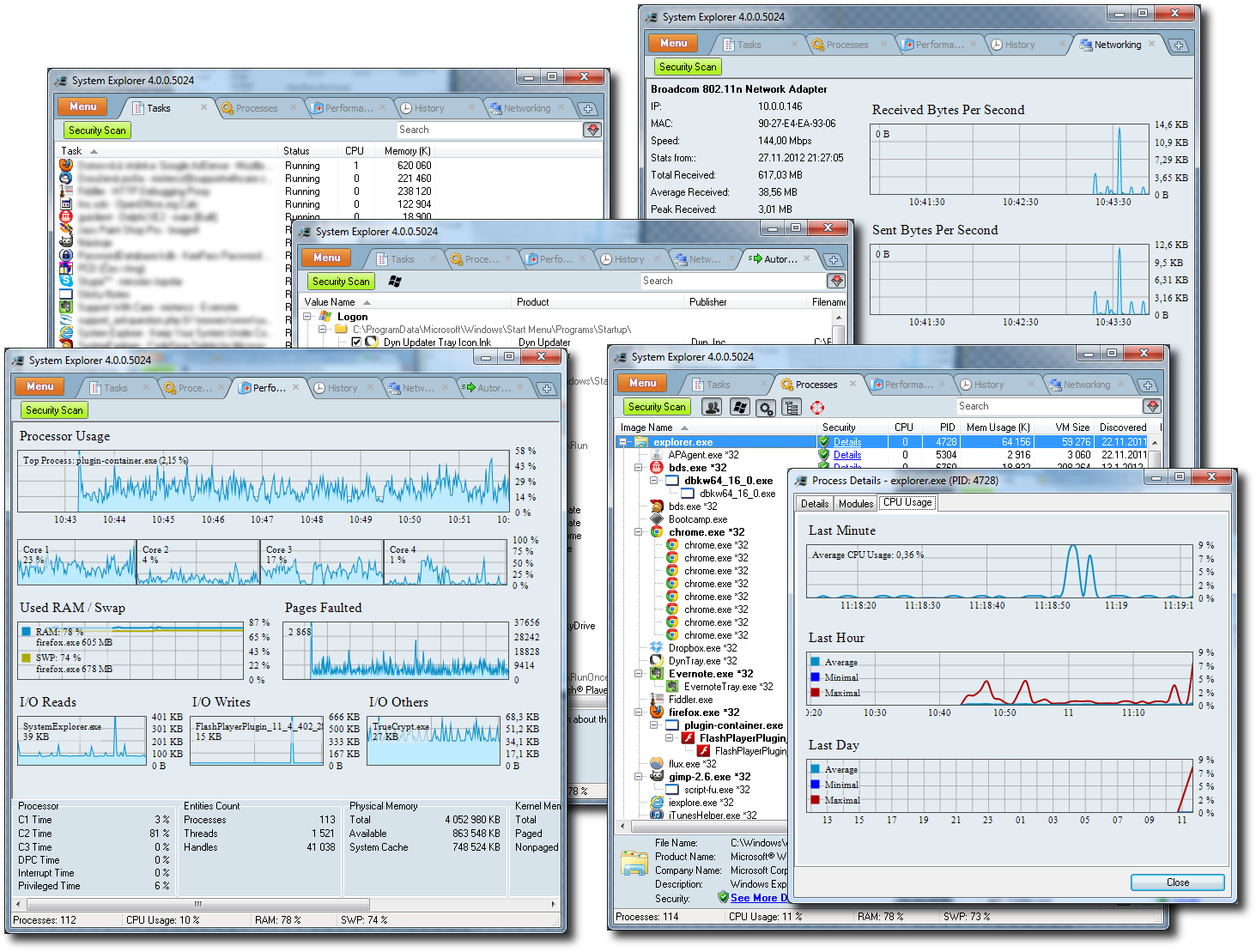


Рисунок 1.3 – System Explorer

Список выполняемых функций:

* Иформация о процессах;
* Графики производительности;
* Запуск новых процессов.
* Завершение процессов;

Рассмотрим достоинства данного приложения:

* Предоставляет более детальную информацию о системе и процессах;
* Предоставляет детальную информацию о системе и процессах;

Рассмотрим недостатки данного приложения:

* Сложный и перегружанный интерфейс.

## Перечень функциональных требований

После анализа аналогов был составлен следующий список выполняемых функций:

* Информация о процессах;
* Завершение процессов;
* Поиск по процессам;
* Графики производительности.

Для разработки программного средства будет использоваться язык программирования C# и среда разработки Visual Studio 2022, а также Windows Form и WinApis.

Windows Form используется с целью создания графического интерфейса.

WinApi используется с целью получения системной информации.

## Состав и параметры технических и программных средств

Приложение должно функционировать на персональных компьютерах со следующими характеристиками:

* процессор Intel i3-4000m или лучше;
* оперативная память 4 GB или лучше;
* свободное место на диске 1 гигабай или больше;
* операционная система Windows 7 или новее.

В данном разделе указаны минимальные технические требования для запуска программного средства. Для эксплуатации в реальных могут потребоваться более мощные технические средства. Программное средство должно корректно функционировать и на более мощном оборудовании.

# Моделирование предметной области

## WinApi

### Получение информации о процессах в Windows

WinAPI предоставляет функции, такие как EnumProcesses, GetProcessMemoryInfo и GetProcessTimes, которые позволяют получить информацию о запущенных процессах в системе, включая их идентификаторы, использование памяти и времени процессора.

### Получение информации о ресурсах

Функции WinAPI, такие как GlobalMemoryStatusEx и GetSystemTimes, позволяют получить информацию о ресурсах системы, включая общее и доступное количество физической памяти и загрузку процессора.

## C#

### Обёртки в C# для использования WinAPI

В C# можно использовать P/Invoke (Platform Invocation Services) для вызова функций WinAPI. Это позволяет использовать функции WinAPI прямо из кода на C#, обеспечивая мощные возможности для взаимодействия с операционной системой.

[StructLayout(LayoutKind.Sequential)]

public struct THREADENTRY32

{

public uint dwSize;

public uint cntUsage;

public uint th32ThreadID;

public uint th32OwnerProcessID;

public uint tpBasePri;

public uint tpDeltaPri;

public uint dwFlags;

public THREADENTRY32()

{

dwSize = (uint)Marshal.SizeOf(typeof(THREADENTRY32));

}

}

[DllImport("kernel32.dll")]

public static extern bool Thread32First(IntPtr hSnapshot, ref THREADENTRY32 lpte);

# Проектирование программного средства

## Структура программы

При разработке приложения будет использовано 13 модулей:

* GlobalStatistics – сбор информации о использовании ресурсов;
* ListProcesses – группировка процессов;
* Process – сбор информации о процессе;
* HAPI – C# обёртка для WinApi модуля HandleapiDLL;
* PSAPI – C# обёртка для WinApi модуля PsapiDLL;
* PTAPI – C# обёртка для WinApi модуля ProcessthreadsapiDLL;
* SIAPI – C# обёртка для WinApi модуля SysinfoapiDLL;
* THAPI – C# обёртка для WinApi модуля Tlhelp32DLL;
* Localization – изменение языка программы;
* ChangeVisableColumns – изменение отображения колонок в таблице информации о процессах;
* ProcessesColumns – вспомогательный модуль колонок в таблице информации о процессах;
* TaskManager – главное окно и код обработки;
* TextGui – вспомогательный модуль изменения языка программы.

## Проектирование интерфейса программного средства

### Вкладка процессы

Содержит основные функции работы с процессами к кнопки навигации по вкладкам. Макет представлен на рисунке 3.1.

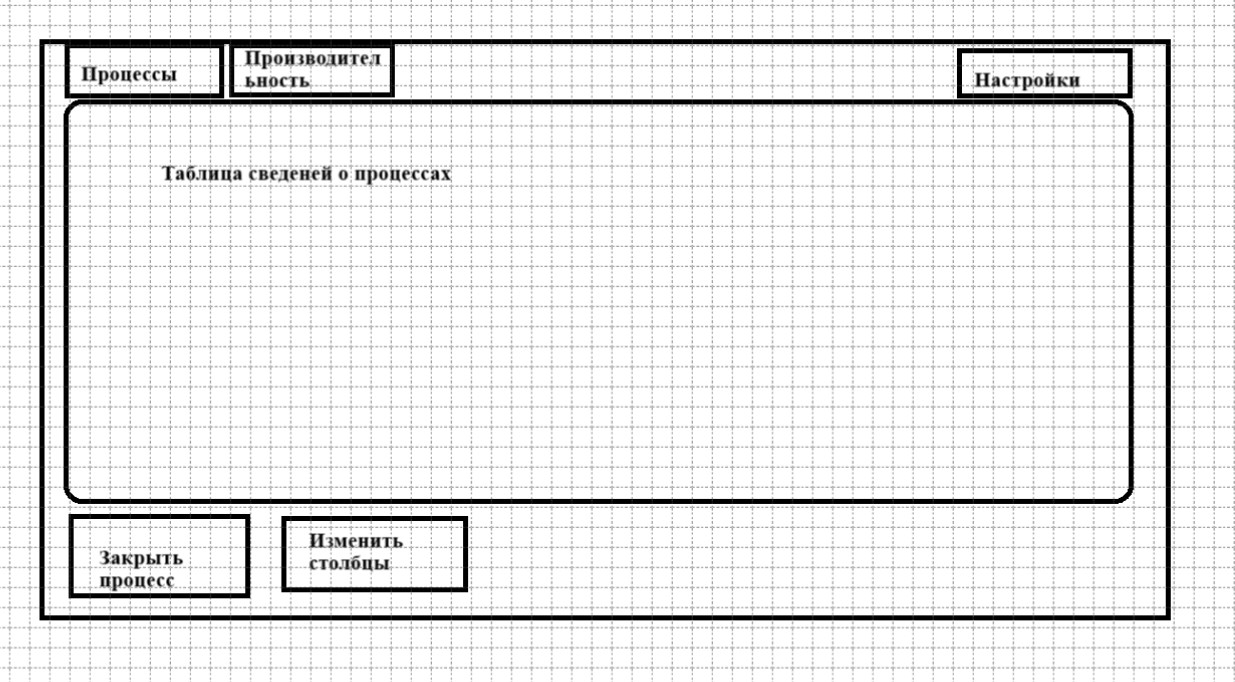


Рисунок 3.1 – Вкладка процессы

### Окно редактировать столбцы

Содержит элементы выбора столбцов и кнопку подтверждения. Макет представлен на рисунке 3.2.

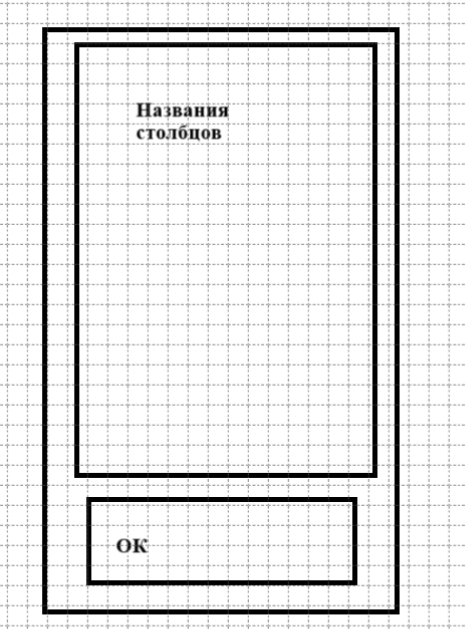


Рисунок 3.2 – Окно редактировать столбцы

### Вкладка производительность

Содержит графики системных ресурсов и кнопки навигации. Макет представлен на рисунке 3.3.

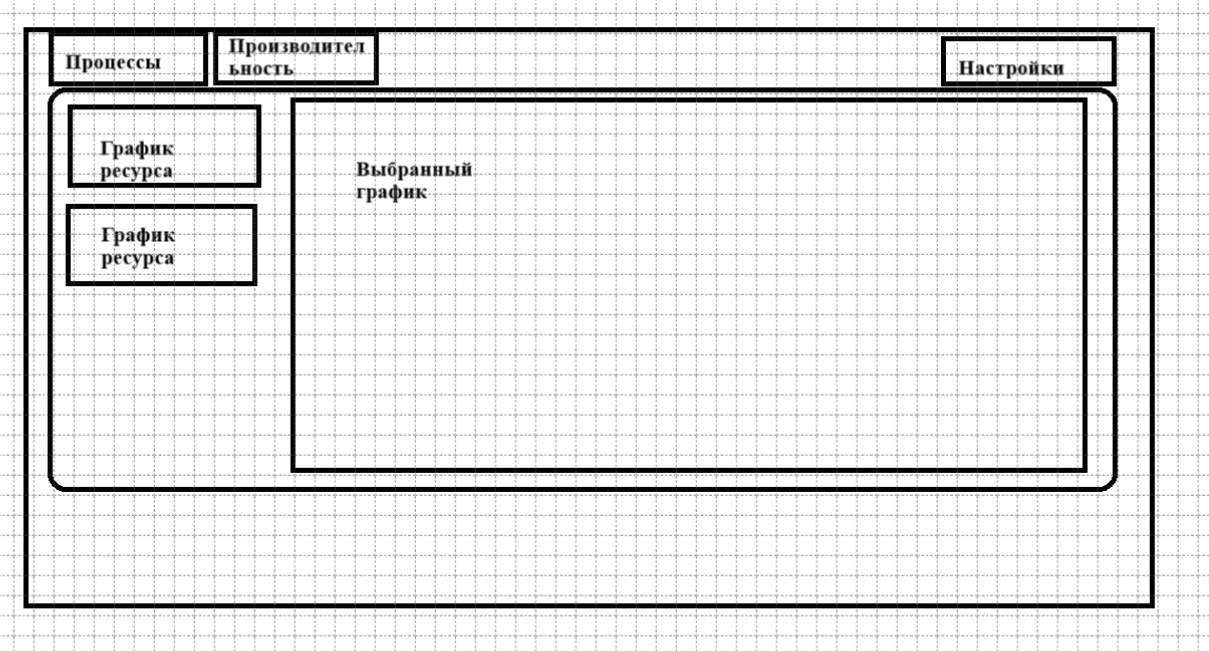


Рисунок 3.3 – Вкладка производительность

### Вкладка настройки

Содержит элементы настроек и кнопки навигации. Макет представлен на рисунке 3.4.

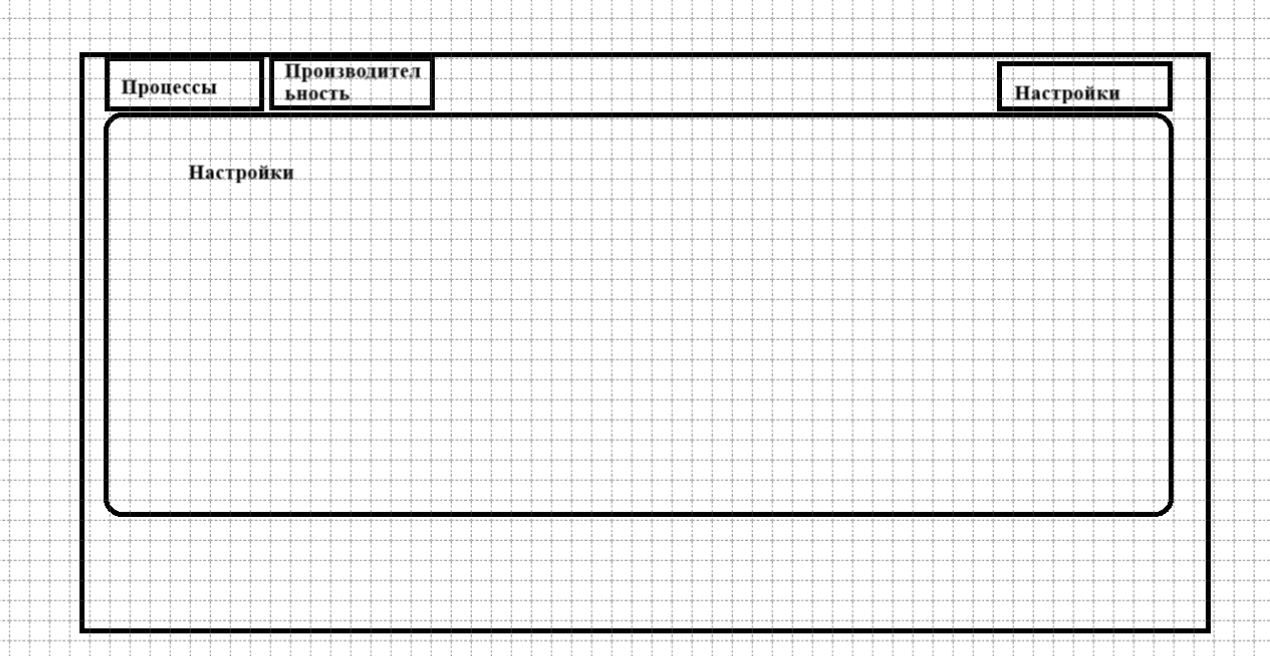


Рисунок 3.4 – Вкладка настройки

## Проектирование функционала программного средства

При создании приложения очень важно сразу определить задачу и цели, а также хорошо составить рабочие алгоритмы. Искать ошибки в коде придётся в любом случае, но хорошо написанный алгоритм упрощает это в разы. В программе должны быть использованы следующие алгоритмы:

* Получение количества потоков процесса;
* Обновление информации о глобальном использовании ОЗУ;
* Завершение дерева процессов.

### Получение количества потоков процесса



Рисунок 3.5 – Получение количества потоков процесса

### Обновление информации о глобальном использовании ОЗУ



Рисунок 3.6 – Обновление информации о глобальном использовании ОЗУ

### Завершение дерева процессов



Рисунок 3.7 – Завершение дерева процессов

# Разработка программного средства

## Обновление информации о глобальном использовании ЦП

Вызов процедуры осуществляется с заданной переодичностью. Процедура использует новое и предыдущее значения использования времени процессора, для вычесления загруженость процессора в момент вызова. Код процедуры будет предоставлен ниже.

private void ChangeCpuUsePercent()

{

var timeAll = PTAPI.GetSystemTimesAll();

var timeUse = PTAPI.GetSystemTimesUse();

var delta = ((timeUse - timeUsePred) \* 1.0) / (timeAll - timeAllPred)

\* MAX\_PERCENT;

timeAllPred = timeAll;

timeUsePred = timeUse;

if (delta > MAX\_PERCENT)

{

return;

}

CpuUsePercent.Enqueue(Convert.ToInt32(delta));

if (CpuUsePercent.Count > COUNT\_TIME)

{

CpuUsePercent.Dequeue();

}

}

## Получение информации о использовании ЦП процессам

Процедура выполняет запрос общего времени процессора и запрос времени для каждого из процессов. Потом ожидает 100мс и повторяет запросы. В результате на основе данных из 2 запросов получает процент использования процессора процессом. Код процедуры будет предоставлен ниже.

private void CalcCpuUse()

{

var processesTime = new ProcessTime[processes.Count];

// Подсчет 1вого системного времени процесса

var systemTime1 = PTAPI.GetSystemTimesAll();

// Подсчет 1вого времени процесса

int i = 0;

foreach (Process process in processes.Values)

{

processesTime[i].ProcessTime1 =

PTAPI.GetProcessTimesUse(process.Handle);

i++;

}

Thread.Sleep(100);

// Подсчет 2вого системного времени процесса

var systemTime2 = PTAPI.GetSystemTimesAll();

// Подсчет 2вого времени процесса

i = 0;

foreach (Process process in processes.Values)

{

processesTime[i].ProcessTime2 =

PTAPI.GetProcessTimesUse(process.Handle);

i++;

}

// Подсчет загрузки цп процессами

var systemTimeDelta = systemTime2 - systemTime1;

i = 0;

foreach (var process in processes)

{

var delta = (processesTime[i].ProcessTime2 –

processesTime[i].ProcessTime1) \* 1.0;

process.Value.SetCpu(Convert.ToUInt32(delta / systemTimeDelta

\* 100));

i++;

}

}

## Получение имени пользователя создавшего процесса

Процедура получает токен. Затем понему получает информацию. Затем получает токен роли в сестеме и преобраует из него имя роли. Код процедуры будет предоставлен ниже.

private string GetUserName()

{

var tokenHandle = PTAPI.OpenProcessToken(Handle,

PTAPI.TokenDesiredAccess.Query);

var tokenInfo = PTAPI.GetTokenInformation(tokenHandle,

PTAPI.TOKEN\_INFORMATION\_CLASS.TokenUser);

TOKEN\_USER tokenUser = (TOKEN\_USER)Marshal.PtrToStructure(tokenInfo,

typeof(TOKEN\_USER));

SecurityIdentifier sid = new SecurityIdentifier(tokenUser.User.Sid);

Marshal.FreeHGlobal(tokenInfo);

HAPI.CloseHandle(tokenHandle);

return sid.Translate(typeof(NTAccount)).ToString();

}

# Тестирование программного средства

## Запуск программы

Исходный набор данных: запуск программы.

Ожидаемый результат: Запуск программы на окне процессов.

Очевидно, что тест пройден успешно.

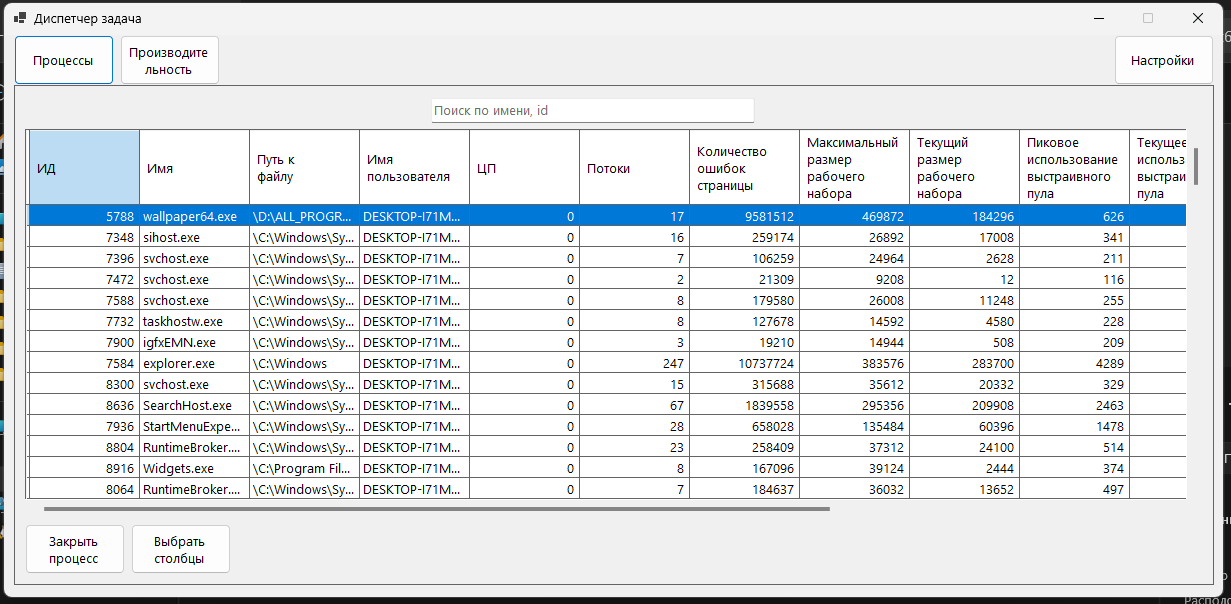


Рисунок 5.1 – Запуск программы

## Переключение вкладок

Исходный набор данных: переключение вкладок.

Ожидаемый результат: открытие вкладки производительность.

Очевидно, что тест пройден успешно.

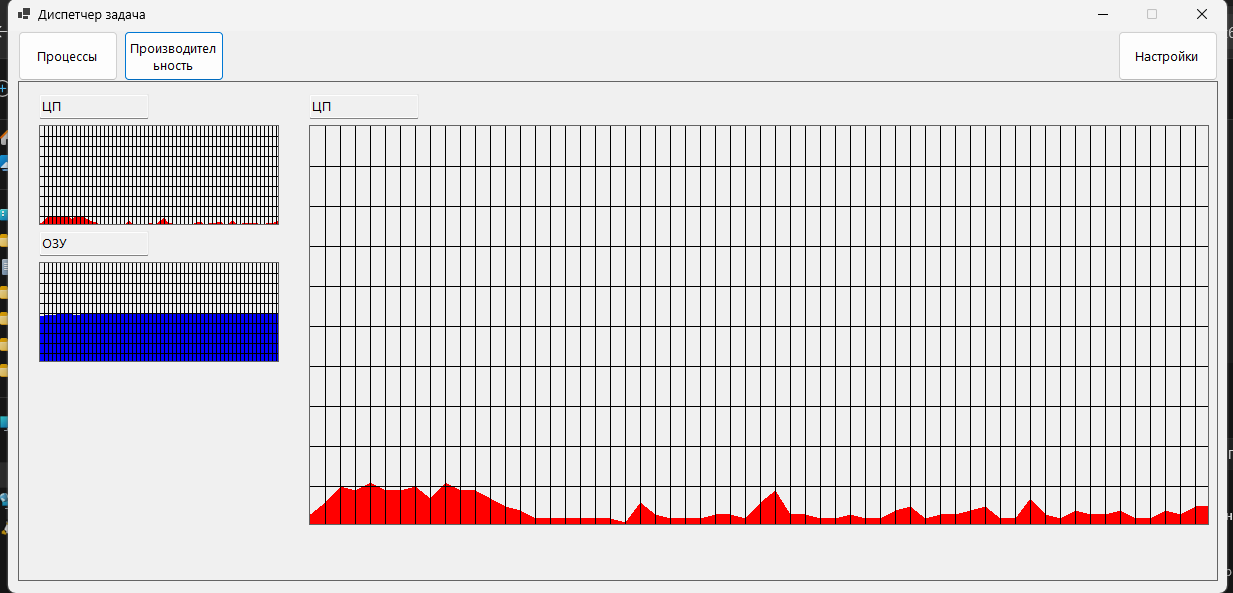


Рисунок 5.2 – Переключение вкладок

## Переключение языка

Исходный набор данных: переключение языка.

Ожидаемый результат: программа на англиском.

Очевидно, что тест пройден успешно.

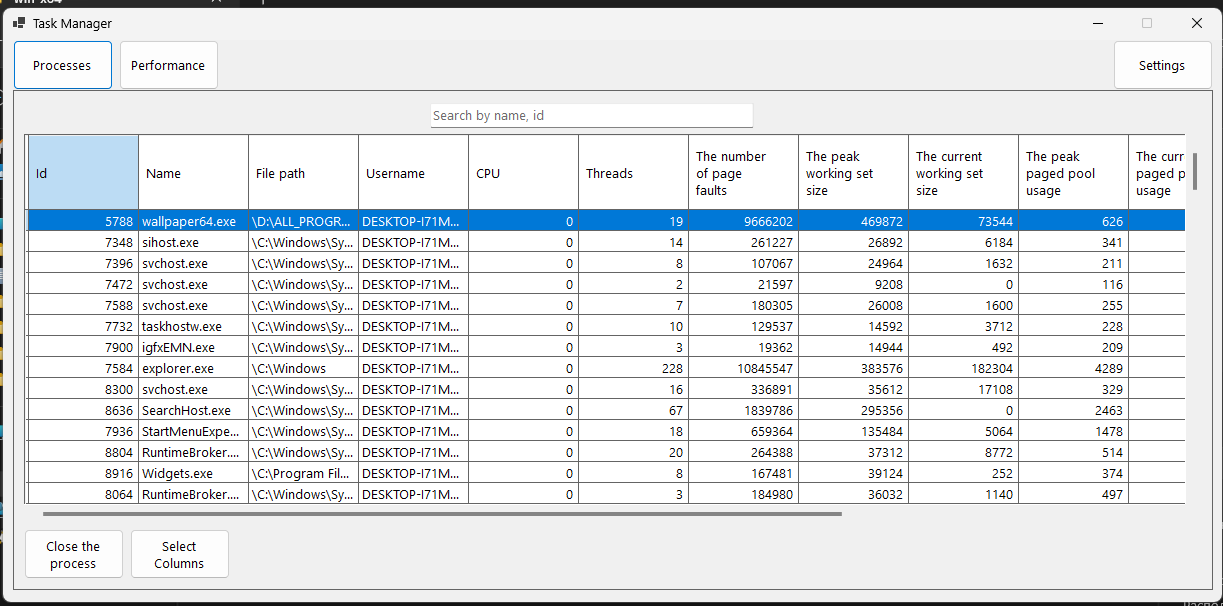


Рисунок 5.2 – Переключение языка

# Руководство пользователя

## Минимальные системные требования

Приложение должно функционировать на персональных компьютерах со следующими характеристиками:

* процессор Intel i3-4000m или лучше;
* оперативная память 4 GB или лучше;
* свободное место на диске 1 гигабай или больше;
* операционная система Windows 7 или новее.

## Установка

На установочном диске находится zip архив. После его распоковки программа пригодна к использованию.

### Вкладка процессов

При нажатии на название столбца происходит сортировка. Для закрытия процесса выбрать прцоесс и на нажать кнопку Закрытия процесса.

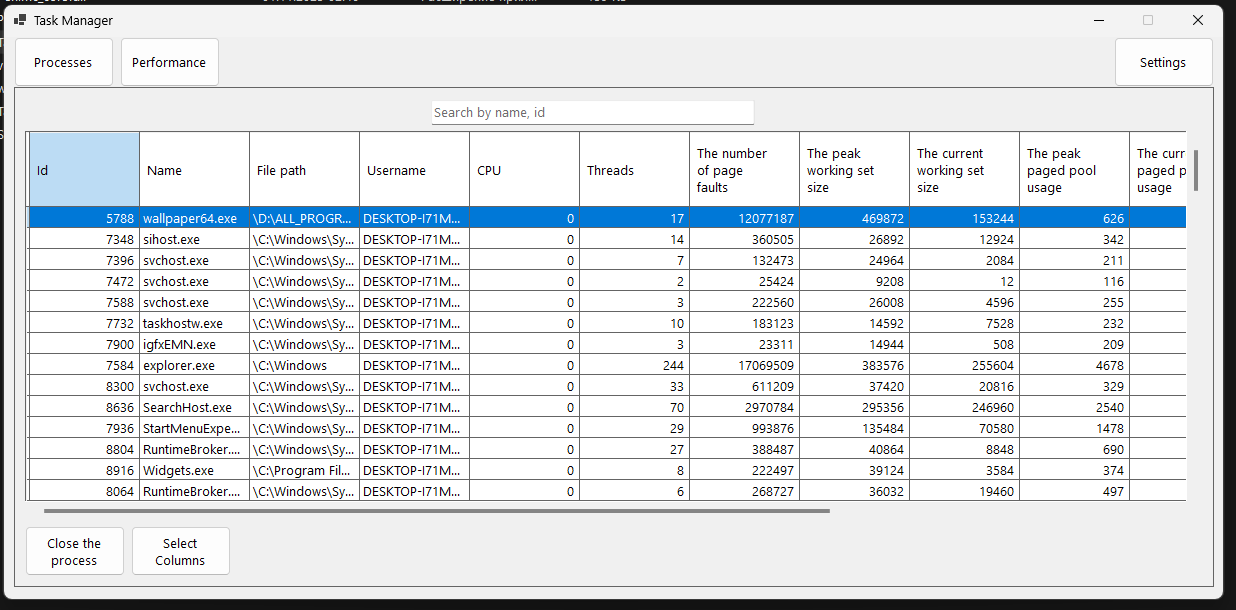


Рисунок 6.1 – Вкладка процессов

### Окно изменения столбцов

Выбрать нужные колонки и нажать сохранить.

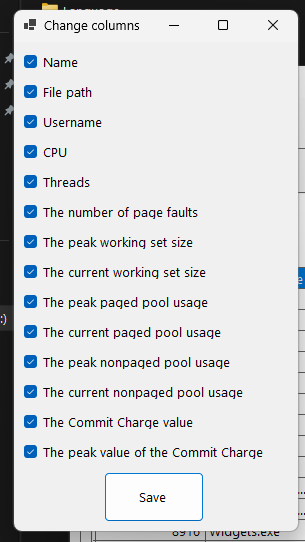


Рисунок 6.2 – Окно изменения столбцов

### Вкладка производительности

Для смены графика справо нажать на 1 из левых графиков

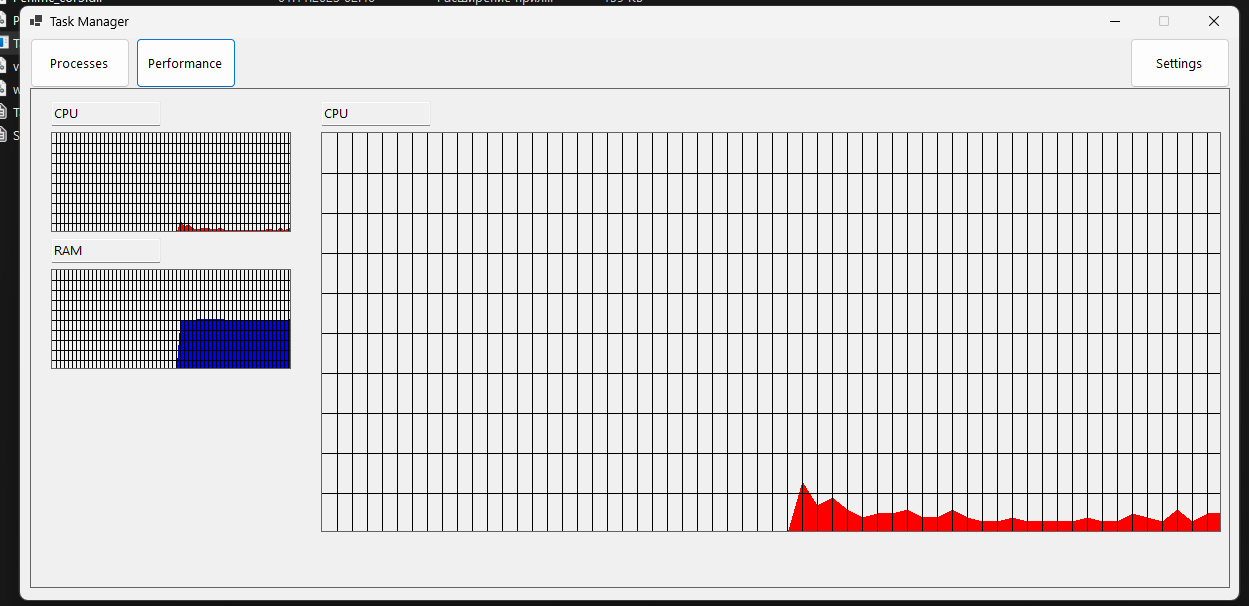


Рисунок 6.3 – Вкладка производительности

### Вкладка настроек

Для смены языка выбрать из выпадающего списка язык. Для изменения скорости обновления данных выбрать из выподающего списка число.

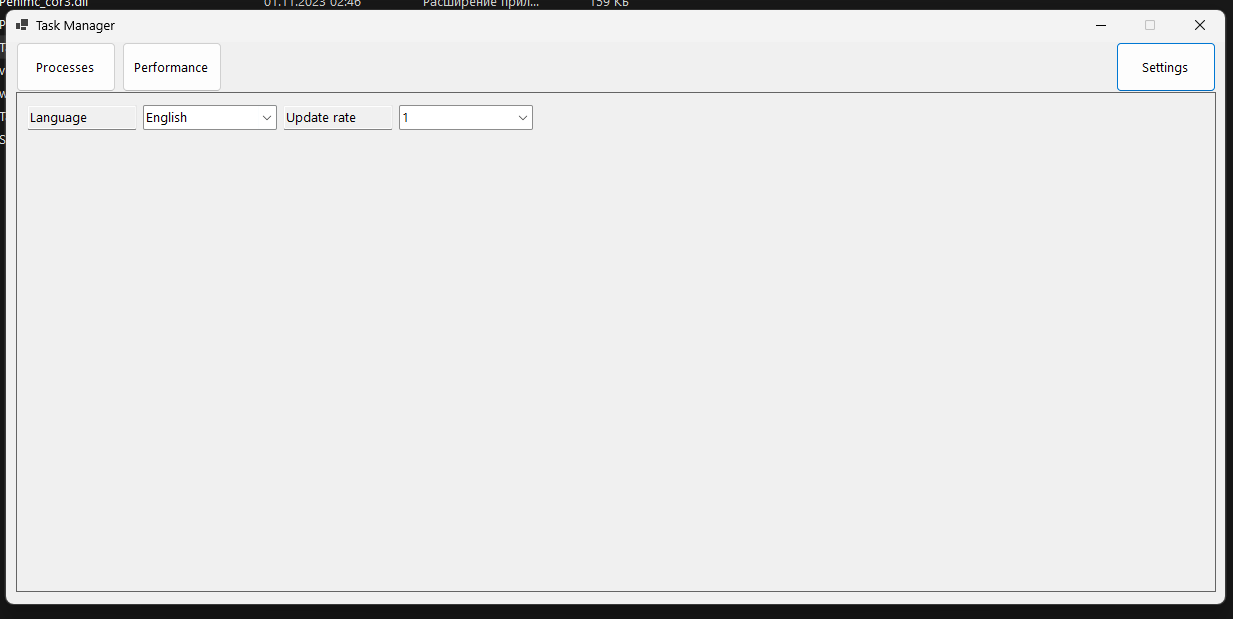


Рисунок 6.4 – Вкладка настроек

# Заключение

В ходе работы над курсовым проектом было разработано приложение - диспетчер задач, использующий WinAPI и написанный на C#. Это приложение предоставляет информацию о программном обеспечении и аппаратном обеспечении компьютера.

Среди преимуществ программы можно отметить интуитивно понятный интерфейс, простоту реализации, современный дизайн, а также низкие системные требования.

В процессе разработки были успешно выполнены следующие задачи:

* Изучены основы работы с WinAPI.
* Реализован диспетчер задач с возможностью просмотра запущенных процессов и их ресурсов.
* Реализована функциональность для управления процессами, включая завершение процессов.
* Протестирована работоспособность диспетчера задач на разных устройствах.

Приложение прошло все этапы тестирования и продемонстрировало корректную работу. Бесценным результатом курсового проектирования является полученный опыт работы с WinAPI и C#. Был получен опыт работы с графическим интерфейсом пользователя, обобщены и применены все ранее полученные знания.

В перспективе данное программное средство может быть усовершенствовано, например, путем оптимизации кода или добавления новых функций. Это может включать в себя более детальное отслеживание использования ресурсов, улучшенное управление процессами или дополнительные опции настройки для пользователей.

# Список использованной литературы

[1] Рихтер Джеффри. Windows via C/C++. - М.: Вильямс, 2009. – Дата доступа: 01.10.2023.

[2] Prosise, Jeff. Programming Windows with MFC. - Microsoft Press, 1999. – Дата доступа: 01.10.2023.

[3] Soulami, Tarik. Inside Windows Debugging: A Practical Guide to Debugging and Tracing Strategies in Windows. - Microsoft Press, 2012. – Дата доступа: 01.10.2023.

[4] Russinovich, Mark E., Solomon, David A., Ionescu, Alex. Windows Internals, Part 1: System architecture, processes, threads, memory management, and more. - Microsoft Press, 2017. – Дата доступа: 01.10.2023.

[5] Liberty, Jesse. Programming C# 8.0: Build Windows, Web, and Desktop Applications. - O’Reilly Media, 2019. – Дата доступа: 01.10.2023.

[6] Albahari, Joseph. C# 8.0 in a Nutshell: The Definitive Reference. - O’Reilly Media, 2020. – Дата доступа: 01.10.2023.

[7] Рихтер Джеффри. CLR via C#. - М.: Вильямс, 2012. – Дата доступа: 01.10.2023.

Приложение А

(обязательное)

Исходный код программы

using SystemInfo.Processes.WinApi.PTAPI;

using SystemInfo.SystemDate.WinApi.SIAPI;

namespace SystemInfo.SystemDate.GlobalStatistics

{

public class GlobalStatistics

{

public const int COUNT\_TIME = 60;

public const int MAX\_PERCENT = 100;

public enum StatisticsType

{

Cpu, Ram

}

public Queue<int> CpuUsePercent { get; private set; } = new Queue<int>();

public Queue<int> RamUsePercent { get; private set; } = new Queue<int>();

private UInt64 timeAllPred = 0;

private UInt64 timeUsePred = 0;

public GlobalStatistics()

{

}

public void Update()

{

ChangeCpuUsePercent();

ChangeRamUsePercent();

}

private void ChangeCpuUsePercent()

{

var timeAll = PTAPI.GetSystemTimesAll();

var timeUse = PTAPI.GetSystemTimesUse();

var delta = ((timeUse - timeUsePred) \* 1.0) / (timeAll - timeAllPred) \* MAX\_PERCENT;

timeAllPred = timeAll;

timeUsePred = timeUse;

if (delta > MAX\_PERCENT)

{

return;

}

CpuUsePercent.Enqueue(Convert.ToInt32(delta));

if (CpuUsePercent.Count > COUNT\_TIME)

{

CpuUsePercent.Dequeue();

}

}

private void ChangeRamUsePercent()

{

var res = SIAPI.GlobalMemoryStatusEx();

RamUsePercent.Enqueue(Convert.ToInt32(res.dwMemoryLoad));

if (RamUsePercent.Count > COUNT\_TIME)

{

RamUsePercent.Dequeue();

}

}

}

}

Приложение Б

(обязательное)

Исходный код программы

using System.Runtime.InteropServices;

using System.Security.Principal;

using SystemInfo.Processes.WinApi.HAPI;

using SystemInfo.Processes.WinApi.PSAPI;

using SystemInfo.Processes.WinApi.PTAPI;

using static SystemInfo.Processes.WinApi.PTAPI.ProcessthreadsapiDLL;

namespace SystemInfo.SystemDate.Processes

{

public class Process

{

public uint Id { get; }

public nint Handle { get; }

public string Name { get; }

public string FilePath { get; }

public string UserName { get; }

public uint Cpu { get; private set; }

public int CountThreads { get; private set; } = 0;

public SystemInfo.Processes.WinApi.PsapiDLL.PROCESS\_MEMORY\_COUNTERS Memory { get; }

public Process(uint id, nint handle)

{

Id = id;

Handle = handle;

var name = PSAPI.GetModuleFileNameEx(Handle);

var subStr = name.Split("\\");

FilePath = "";

for (int i = 0; i < subStr.Length - 1; i++)

{

FilePath += "\\" + subStr[i];

}

Name = subStr.Last();

Memory = PSAPI.GetProcessMemoryInfo(Handle);

UserName = GetUserName();

}

private string GetUserName()

{

var tokenHandle = PTAPI.OpenProcessToken(Handle, PTAPI.TokenDesiredAccess.Query);

var tokenInfo = PTAPI.GetTokenInformation(tokenHandle, PTAPI.TOKEN\_INFORMATION\_CLASS.TokenUser);

TOKEN\_USER tokenUser = (TOKEN\_USER)Marshal.PtrToStructure(tokenInfo, typeof(TOKEN\_USER));

SecurityIdentifier sid = new SecurityIdentifier(tokenUser.User.Sid);

Marshal.FreeHGlobal(tokenInfo);

HAPI.CloseHandle(tokenHandle);

return sid.Translate(typeof(NTAccount)).ToString();

}

public void IncrementThreadCount()

{

CountThreads++;

}

public void SetCpu(uint value)

{

Cpu = value;

}

public void KillProcess()

{

PTAPI.TerminateProcess(Handle, 0);

}

// Деструктор

private void ClearProcess()

{

HAPI.CloseHandle(Handle);

}

// Деструктор

~Process()

{

ClearProcess();

}

}

}

Приложение В

(обязательное)

Исходный код программы

using System.Collections;

using SystemInfo.Processes.WinApi.HAPI;

using SystemInfo.Processes.WinApi.PSAPI;

using SystemInfo.Processes.WinApi.PTAPI;

using SystemInfo.Processes.WinApi.THAPI;

namespace SystemInfo.SystemDate.Processes

{

public class ListProcesses : IEnumerable<Process>

{

private readonly Dictionary<uint, Process> processes;

public int Count { get; }

private struct ProcessTime

{

public ulong ProcessTime1;

public ulong ProcessTime2;

}

public ListProcesses()

{

var processesIds = PSAPI.EnumProcesses();

processes = new Dictionary<uint, Process>();

if (processesIds != null)

{

CalcProcessHandle(processesIds);

CalcThread();

CalcCpuUse();

}

Count = processes.Count;

}

private void CalcProcessHandle(uint[]? processesIds)

{

nint processHandle;

foreach (uint indexIds in processesIds)

{

processHandle = PTAPI.OpenProcess(PTAPI.ProcessDesiredAccess.PROCESS\_ALL\_ACCESS, false, indexIds);

if (processHandle != nint.Zero)

{

processes.Add(indexIds, new Process(indexIds, processHandle));

}

}

}

private void CalcCpuUse()

{

var processesTime = new ProcessTime[processes.Count];

// Подсчет 1вого системного времени процесса

var systemTime1 = PTAPI.GetSystemTimesAll();

// Подсчет 1вого времени процесса

int i = 0;

foreach (Process process in processes.Values)

{

processesTime[i].ProcessTime1 = PTAPI.GetProcessTimesUse(process.Handle);

i++;

}

Thread.Sleep(100);

// Подсчет 2вого системного времени процесса

var systemTime2 = PTAPI.GetSystemTimesAll();

// Подсчет 2вого времени процесса

i = 0;

foreach (Process process in processes.Values)

{

processesTime[i].ProcessTime2 = PTAPI.GetProcessTimesUse(process.Handle);

i++;

}

// Подсчет загрузки цп процессами

var systemTimeDelta = systemTime2 - systemTime1;

i = 0;

foreach (var process in processes)

{

var delta = (processesTime[i].ProcessTime2 - processesTime[i].ProcessTime1) \* 1.0;

process.Value.SetCpu(Convert.ToUInt32(delta / systemTimeDelta \* 100));

i++;

}

}

private void CalcThread()

{

var handleProcessSnap = THAPI.CreateToolhelp32Snapshot(THAPI.Flag.TH32CS\_SNAPTHREAD, 0);

for (var threadEntry32 = THAPI.Thread32First(handleProcessSnap); threadEntry32 != null; threadEntry32 = THAPI.Thread32Next(handleProcessSnap))

{

if (processes.TryGetValue(threadEntry32.Value.th32OwnerProcessID, out Process process))

{

process.IncrementThreadCount();

}

}

HAPI.CloseHandle(handleProcessSnap);

}

public void KillProcessTree(uint processId)

{

var handleProcessSnap = THAPI.CreateToolhelp32Snapshot(THAPI.Flag.TH32CS\_SNAPPROCESS, 0);

uint rootId = processId;

bool finish;

// Полученние самого главного родителя

do

{

finish = true;

for (var processEntry32 = THAPI.Process32First(handleProcessSnap); processEntry32 != null; processEntry32 = THAPI.Process32Next(handleProcessSnap))

{

if (processEntry32.Value.th32ProcessID == rootId)

{

if (processEntry32.Value.th32ParentProcessID != 0 && processes.ContainsKey(processEntry32.Value.th32ParentProcessID) && processes[processEntry32.Value.th32ParentProcessID].Name == processes[processId].Name)

{

rootId = processEntry32.Value.th32ParentProcessID;

finish = false;

}

break;

}

}

} while (!finish);

processes[rootId].KillProcess();

HAPI.CloseHandle(handleProcessSnap);

}

public Process this[uint processId]

{

get

{

return processes[processId];

}

}

public IEnumerator<Process> GetEnumerator()

{

foreach (var process in processes.Values)

{

yield return process;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

}

}

Приложение Г

(обязательное)

Исходный код программы

using System.ComponentModel;

using System.Data;

using System.Reflection;

using System.Text.Json;

using SystemInfo.SystemDate.GlobalStatistics;

using SystemInfo.SystemDate.Processes;

using Task\_manager\_WinApi.Language;

using static SystemInfo.SystemDate.GlobalStatistics.GlobalStatistics;

using static Task\_manager\_WinApi.ProcessesColumns;

namespace Task\_manager\_WinApi

{

internal partial class fTaskManager : Form

{

private ListProcesses listProcesses = new();

private GlobalStatistics globalStatistics = new();

private ProcessesColumns processesColumns;

private ProcessesColumnsName[] processesColumnWhitchVisable =

[

ProcessesColumnsName.Id,

ProcessesColumnsName.Name,

ProcessesColumnsName.FilePath,

ProcessesColumnsName.UserName,

ProcessesColumnsName.Cpu,

ProcessesColumnsName.CountThreads,

ProcessesColumnsName.PageFaultCount,

ProcessesColumnsName.PeakWorkingSetSize,

ProcessesColumnsName.WorkingSetSize,

ProcessesColumnsName.QuotaPeakPagedPoolUsage,

ProcessesColumnsName.QuotaPagedPoolUsage,

ProcessesColumnsName.QuotaPeakNonPagedPoolUsage,

ProcessesColumnsName.QuotaNonPagedPoolUsage,

ProcessesColumnsName.PagefileUsage,

ProcessesColumnsName.PeakPagefileUsage

];

public ProcessesColumnsName[]? TmpProcessesColumnWhitchVisable { get; set; } = null;

private uint? selectedProcessId = null;

private ChangeVisableColumns? subForm = null;

private string? searchProcess;

private StatisticsType statisticsType = StatisticsType.Cpu;

public fTaskManager()

{

InitializeComponent();

cbLanguage.SelectedIndex = 0;

cbUpdateTime.SelectedIndex = 0;

ProcessesUpdate();

tUpdate.Start();

HideAllPanelsExcept(pProceses);

typeof(DataGridView).InvokeMember("DoubleBuffered", BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.SetProperty, null, dgvProcesses, new object[] { true });

typeof(Panel).InvokeMember("DoubleBuffered", BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.SetProperty, null, pMainGraph, new object[] { true });

typeof(Panel).InvokeMember("DoubleBuffered", BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.SetProperty, null, pCpu, new object[] { true });

typeof(Panel).InvokeMember("DoubleBuffered", BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.SetProperty, null, pRam, new object[] { true });

}

private void ProcessesUpdate()

{

var sortedColumn = dgvProcesses.SortedColumn;

var sortOrder = dgvProcesses.SortOrder;

int indexFirstRowOnDisplay = dgvProcesses.FirstDisplayedScrollingRowIndex;

int offsetColumOnDisplay = dgvProcesses.HorizontalScrollingOffset;

listProcesses = new ListProcesses();

// Скрытие колонок

if (TmpProcessesColumnWhitchVisable != null)

{

dgvProcesses.Columns.Clear();

processesColumnWhitchVisable = TmpProcessesColumnWhitchVisable;

TmpProcessesColumnWhitchVisable = null;

}

// Создание колонок в dataTable

var dataTable = new DataTable();

foreach (var indexColumn in processesColumnWhitchVisable)

{

dataTable.Columns.Add(processesColumns.GetColumeName(indexColumn));

}

// Простановка не изменений текста

for (int i = 0; i < dataTable.Columns.Count; i++)

{

dataTable.Columns[i].ReadOnly = true;

}

// Занос данныъ в таблицу

int index;

object[] data = new object[dataTable.Columns.Count];

foreach (var process in listProcesses)

{

index = 0;

if (searchProcess != null && !process.Id.ToString().Contains(searchProcess) && !process.Name.ToUpper().Contains(searchProcess.ToUpper()))

{

continue;

}

foreach (var indexColumn in processesColumnWhitchVisable)

{

data[index++] = GetColumeValue(indexColumn, process);

}

dataTable.Rows.Add(data);

}

// Передача данных в dvg

dgvProcesses.DataSource = dataTable;

if (dataTable.Rows.Count > 0)

{

// положение текста

for (int i = 0; i < dgvProcesses.Columns.Count; i++)

{

if (dgvProcesses.Columns[i].Name == processesColumns.GetColumeName(ProcessesColumnsName.Name) ||

dgvProcesses.Columns[i].Name == processesColumns.GetColumeName(ProcessesColumnsName.FilePath) ||

dgvProcesses.Columns[i].Name == processesColumns.GetColumeName(ProcessesColumnsName.UserName))

{

dgvProcesses.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.BottomLeft;

}

else

{

dgvProcesses.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.BottomRight;

}

}

if (sortedColumn != null && dgvProcesses.Columns.Contains(sortedColumn.Name) && sortOrder != SortOrder.None)

{

dgvProcesses.Sort(dgvProcesses.Columns[sortedColumn.Name], (ListSortDirection)sortOrder - 1);

}

for (int i = 0; i < dgvProcesses.Rows.Count; i++)

{

if (IsSelectedProcess(i))

{

break;

}

}

if (indexFirstRowOnDisplay > -1 && offsetColumOnDisplay > -1)

{

if (dgvProcesses.Rows.Count > indexFirstRowOnDisplay)

{

dgvProcesses.FirstDisplayedScrollingRowIndex = indexFirstRowOnDisplay;

}

dgvProcesses.HorizontalScrollingOffset = offsetColumOnDisplay;

}

}

}

private void GlobalStatisticsUpdate()

{

globalStatistics.Update();

}

private void HideAllPanelsExcept(Panel panel)

{

pProceses.Visible = false;

pGlobalStatistics.Visible = false;

pSetting.Visible = false;

panel.Visible = true;

}

private bool IsSelectedProcess(int index)

{

if (Convert.ToUInt32(dgvProcesses.Rows[index].Cells[processesColumns.GetColumeName(ProcessesColumnsName.Id)].Value.ToString()) == Convert.ToUInt32(selectedProcessId))

{

dgvProcesses.Rows[Convert.ToInt32(index)].Selected = true;

return true;

}

return false;

}

private void ChangeLanguage(Localization.Language language)

{

ChangeLanguageProcessesColumns(language);

ChangeLanguageTextGui(language);

}

private void ChangeLanguageProcessesColumns(Localization.Language language)

{

processesColumns = new(language);

}

private void ChangeLanguageTextGui(Localization.Language language)

{

var str = File.ReadAllText($"Language\\{Localization.GetLanguageName(language)}\\TextGui.txt");

var textGui = JsonSerializer.Deserialize<TextGui>(str);

Text = textGui.fTaskManager;

bKillProcess.Text = textGui.bKillProcess;

bChangeVisableColumns.Text = textGui.bChangeVisableColumns;

tbSearch.PlaceholderText = textGui.tbSearch;

bProcesses.Text = textGui.bProcesses;

bGlobalStatistics.Text = textGui.bGlobalStatistics;

bSetting.Text = textGui.bSetting;

tbCpu.Text = textGui.tbCpu;

tbRam.Text = textGui.tbRam;

tbLanguage.Text = textGui.tbLanguage;

tbUpdateTime.Text = textGui.tbUpdateTime;

}

private void bProcesses\_Click(object sender, EventArgs e)

{

if (!pProceses.Visible)

{

ProcessesUpdate();

HideAllPanelsExcept(pProceses);

}

}

private void bGlobalStatistics\_Click(object sender, EventArgs e)

{

if (!pGlobalStatistics.Visible)

{

HideAllPanelsExcept(pGlobalStatistics);

}

}

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

{

if (pProceses.Visible)

{

ProcessesUpdate();

}

GlobalStatisticsUpdate();

if (pGlobalStatistics.Visible)

{

pMainGraph.Invalidate();

pCpu.Invalidate();

pRam.Invalidate();

}

}

private void bKillProcess\_Click(object sender, EventArgs e)

{

if (selectedProcessId != null)

{

listProcesses.KillProcessTree(selectedProcessId.Value);

}

}

private void bChangeVisableColumns\_Click(object sender, EventArgs e)

{

subForm = new ChangeVisableColumns(processesColumnWhitchVisable, processesColumns, this);

subForm.Visible = true;

}

private void dvgProcesses\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex > -1)

{

selectedProcessId = Convert.ToUInt32(dgvProcesses.Rows[e.RowIndex].Cells[processesColumns.GetColumeName(ProcessesColumnsName.Id)].Value);

}

}

private void tbSearch\_KeyUp(object sender, KeyEventArgs e)

{

searchProcess = tbSearch.Text;

if (searchProcess == "")

{

searchProcess = null;

}

}

private void pMainGraph\_Paint(object sender, PaintEventArgs e)

{

var g = e.Graphics;

switch (statisticsType)

{

case StatisticsType.Cpu:

tbMainGraph.Text = tbCpu.Text;

PaintGraph(g, pMainGraph, globalStatistics.CpuUsePercent, Color.Red);

break;

case StatisticsType.Ram:

tbMainGraph.Text = tbRam.Text;

PaintGraph(g, pMainGraph, globalStatistics.RamUsePercent, Color.Blue);

break;

}

}

private static void PaintGraph(Graphics g, Panel panel, Queue<int> queue, Color color)

{

var brush = new SolidBrush(color);

var points = new Point[GlobalStatistics.COUNT\_TIME + 3];

var stepPaintX = panel.Width / GlobalStatistics.COUNT\_TIME;

var stepPaintY = panel.Height / (GlobalStatistics.MAX\_PERCENT);

int i = 0;

for (; i < GlobalStatistics.COUNT\_TIME - queue.Count; i++)

{

points[i] = new Point(i \* stepPaintX, ((GlobalStatistics.MAX\_PERCENT - 0) \* stepPaintY));

}

foreach (var cpu in queue)

{

points[i] = new Point(i \* stepPaintX, ((GlobalStatistics.MAX\_PERCENT - cpu) \* stepPaintY));

i++;

}

points[i] = new Point(i \* stepPaintX, points[i - 1].Y);

i++;

points[i] = new Point(i \* stepPaintX, panel.Height);

i++;

points[i] = new Point(0, panel.Height);

g.FillPolygon(brush, points);

PaintLines(g, panel);

}

private static void PaintLines(Graphics g, Panel panel)

{

var pen = new Pen(Color.Black);

var stepPaintX = panel.Width / GlobalStatistics.COUNT\_TIME;

var stepPaintY = panel.Height / 10;

for (int i = 1; i < 10; i++)

{

g.DrawLine(pen, new Point { X = 0, Y = i \* stepPaintY }, new Point { X = panel.Width, Y = i \* stepPaintY });

}

for (int i = 1; i < GlobalStatistics.COUNT\_TIME; i++)

{

g.DrawLine(pen, new Point { X = i \* stepPaintX, Y = 0 }, new Point { X = i \* stepPaintX, Y = panel.Height });

}

}

private void pCpu\_Paint(object sender, PaintEventArgs e)

{

var g = e.Graphics;

PaintGraph(g, pCpu, globalStatistics.CpuUsePercent, Color.Red);

}

private void pRam\_Paint(object sender, PaintEventArgs e)

{

var g = e.Graphics;

PaintGraph(g, pRam, globalStatistics.RamUsePercent, Color.Blue);

}

private void pCpu\_Click(object sender, EventArgs e)

{

statisticsType = StatisticsType.Cpu;

}

private void pRam\_Click(object sender, EventArgs e)

{

statisticsType = StatisticsType.Ram;

}

private void bSetting\_Click(object sender, EventArgs e)

{

HideAllPanelsExcept(pSetting);

}

private void cbLanguage\_SelectedIndexChanged(object sender, EventArgs e)

{

ChangeLanguage((Localization.Language)cbLanguage.SelectedIndex);

}

private void cbUpdateTime\_SelectedIndexChanged(object sender, EventArgs e)

{

tUpdate.Interval = Convert.ToInt32(cbUpdateTime.Text) \* 1000;

}

}

}