УДК 004

Создание приложения для управления процессами ОС Windows на языке программирования C#

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс создания оконного приложения для управления процессами ОС Windows. Программа написана на языке программирования C# с использованием библиотеки WinForms. Результатом исследования будет являться программа «Диспетчер задач C#» и описание её работы.
Ключевые слова: C#, WinForms, приложение

Creating an application for managing Windows OS processes in the C programming language#

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the process of creating a window application for managing Windows OS processes. The program is written in the C# programming language using the WinForms library. The result of the study will be the program "C# Task Manager" and a description of its work.
Keywords: C#, WinForms, application

1 Введение

1.1 Актуальность

Разработка программ типа «диспетчер задач» является важным компонентом повышения производительности компьютерных систем. Данные программы позволяют пользователям отслеживать, управлять и завершать запущенные процессы и приложения в их системе. В последние годы наблюдается значительный рост спроса на «диспетчеры» из-за растущей сложности компьютерных систем и необходимости эффективного управления системными ресурсами. Язык программирования C# широко используется при разработке программ благодаря простоте использования, объектно-ориентированному подходу и наличию многочисленных библиотек и фреймворков. В данной статье исследуется разработка программы «диспетчер задач» на языке программирования C# с использованием популярной библиотеки для создания оконных приложений WinForms, принцип её работы, а также практическое применение создаваемого приложения.

1.2 Обзор исследований

Т.В. Ромашкина и В.И. Шагалиев в своей работе рассмотрели основополагающие возможности языка C# и принцип создания Win-приложений [1]. В.М. Куприенко, Н.В. Нечитайло в своей статье описали процесс разработки приложений для Windows на языке C# [2]. И.И.Баранкова, У.В. Михайлова и Г.И. Лукьянов в своей исследование описали процесс разработки разработка приложения, для которого использовался язык программирования C# и СУБД MS SQL Server [3]. Д.Ж. Ликнесс в своём труде изложил практическое руководство по созданию приложений на языке C# для Windows 8, охватывая весь жизненный цикл программ [4]. С. М. Баженов в своей работе представил преимущества языка программирования C# в разработке приложений [5]. А.С. Ерохин в своей статье рассмотрел процесс создания графической части приложения и написания кода к различным элементам приложения [6].

1.3 Цель исследования

Целью исследования является создание приложения, позволяющего управлять процессами в системе с ОС Windows, с помощью языка программирования C# и библиотеки WinForms.

2 Материалы и методы

Для создания программы используется язык программирования C# и библиотека WinForms, которая позволяет создавать настольные приложения с помощью интерфейса перетаскивания. В качестве IDE выступает Visual Studio 2022.

3 Результаты и обсуждения

Разработка диспетчера задач начинается с создания графического интерфейса так, как в дальнейшем будет удобно и видно какие именно элементы нужно программировать. Внешний вид приложения был выбран простой и минималистический (рис. 1).

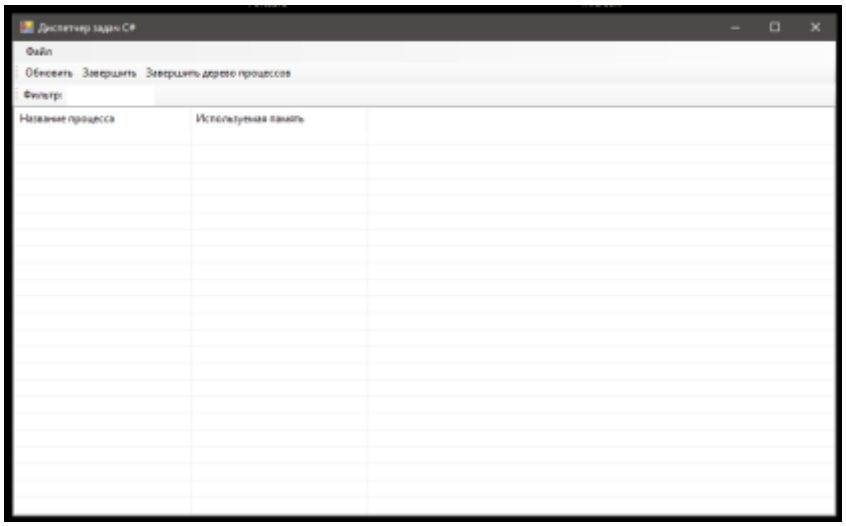

Рисунок 1. Внешний вид приложения

Интерфейс содержит следующие элементы: menuStrip, два элемента toolStrip, contextMenuStrip и listView. Элемент menuStrip отвечает за верхнее меню программы. В данном элементе имеется одна главная вкладка с названием «Файл» и её дочерние вкладки «Выход» и «Запустить задачу». Данный элемент позволяет осуществлять закрытие программы. Первый элемент toolStrip содержит три кнопки, которые в дальнейшем будут иметь функции для работы с процессами – «Обновить», «Завершить» и «Завершить дерево процессов». Второй toolStrip включает в себя элемент TextView и TextBox, который будет представлять из себя фильтр по названию процесса. Последний элемент listView предназначен для вывода списка процессов и информации о них. В listView создаются две колонки, которые имеют название «Название процесса» и «Используемая память».

После создания интерфейса осуществляется программирование приложения. Для этого используется класс Form1.cc, то есть главный класс формы. Перед написанием функций необходимо задействовать несколько пространств имён, помимо автоматически задействованной WinForms. Первым пространством имён является модуль System.Diagnostics, который предоставляющее классы для взаимодействия со средствами и процессами диагностики системы. Второе пространство – это модуль System.Management, который предоставляет классы для взаимодействия с управляющей информацией и событиями в операционной системе Windows. Последний модуль Microsoft.VisualBasic позволяет в элемент MessageBox, который предназначен для обычного оповещения, вводить данные.

В ходе работы понадобится поле «private List<Process> processes», которое является главным списком, хранящим все процессы. В данный список загружается обновлённое состояние процессов, то есть будут учитываться процессы, которые были закрыты или были запущены. Для списка создаётся метод, который будет с ним работать – обновлять и заполнять список (рис. 2).

```
private void GetProcesses()
{

    processes.Clear();
    processes = Process.GetProcesses().ToList<Process>();

}
```

Рисунок 2. Метод GetProcesses() для работы со списком

Данный метод позволяет очищать список, а также заполнять его. Данный метод не работает с графическим интерфейсом приложения, а только со списком List<Process>, в котором хранятся процессы. Для взаимодействия с интерфейсом приложения, то есть чтобы список процессов отображался визуально, создаётся метод, работающий с графическим элементом ListView. Данный метод реализует заполнение ListView данными из списка List<Process>(рис. 3).

```
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);
        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();
}
```

Рисунок 3. Метод RefreshProcessesList() заполнения ListView

В данном методе происходит обращение к элементу ListView и его очистка. Переменная memSize хранит в себе объём памяти, занимаемый процессом. С помощью оператора цикла foreach осуществляется перебор всех процессов. В ходе перебора осуществляется проверка процесса на значение «null». В случае, если процесс не имеет значение «null», то будет происходить вычисление памяти, которое потребляет процесс. Также создаётся ListViewItem, который хранит в себе значение ячеек в ListView. В качестве колонок в ListViewItem передаётся массив строк. Потребляемая память будет отображаться в мегабайтах. В заголовке окна приложения будет выводиться количество запущенных процессов.

Для фильтрации по имени процессов применяется перегрузка вышеописанного метода. Реализуется перегрузка в отдельном методе (рис. 4). Метод перегрузки имеет аналогичное название метода заполнение ListView. В качестве параметров метода используются список List<Process> и строка string keyword, которая хранит ключевое слово и по которому осуществляется поиск(фильтрация) процессов. В параметрах заголовка окна добавляется значение из строкового параметра метода.

Следующие методы отвечают за завершение процессов – по одному выбранному процессу и целое дерево процессов, то есть процесса и его дочерних связанных с ним процессов (рис. 5). Первый метод получает входной параметр «Process». В данном методе используется две функции, одна из которых непосредственно завершает процесс, а вторая создаёт поток ожидать завершения связанного с ним процесса для того, чтобы численное отображение процессов совпадало со списком названий процессов.

```
private void RefreshProcessesList(List<Process> processes, string keyword)
{
    try
    {

        listView1.Items.Clear();

        double memSize = 0;

        foreach (Process p in processes)
        {
            if (p != null)
            {
                memSize = 0;

                PerformanceCounter pc = new PerformanceCounter();
                pc.Category Name = "Process";
                pc.CounterName = "Working Set - Private";
                pc.Instance Name = p.ProcessName;

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

Рисунок 4. Метод перегрузки

```
private void KillProcess(Process process)
{

    process.Kill();
    process.WaitForExit();

}

                                0 references
private void KillProcess And Children(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 object Collection)
    {
        Kill Process And Children(Convert.ToInt32(obj["ProcessID"]));
    }

    try
    {

        Process p = Process.GetProcessById(pid);
        p.Kill();
        p.WaitForExit();
    }

    catch (ArgumentException) { }

}
```

Рисунок 5. Методы завершения одного процесса и дерева процессов

Второй метод, отвечавший за завершение дерева процессов, принимает в качестве входного параметра id процесса. В случае, если id равен нулю, то осуществляется выход из метода. С помощью инструмента ManagementObjectSearcher библиотеки System.Management осуществляется поиск связанных процессов по id, и с помощью рекурсии происходит поочередное завершение найденных процессов. Для того, чтобы получить id родительского процесса, создаётся метод GetParentProcessId (рис. 6).

```
private int GetParentProcessId(Process p)
{

    int parentID = 0;

    try
    {

        ManagementObject managementObject = new ManagementObject("win32_process. handle='" + p.Id + "managementobject.Get();
        parentID = Convert.ToInt32(management Object["Parent Process Id"]);

    }
    catch (Exception) { }
    return parentID;
}
```

Рисунок 6. Метод получения id родительского процесса

После реализации главных функций и методов, осуществляется программирование графического интерфейса, а именно нажатие на кнопки и ввод текста.

Первый метод нажатия на кнопку «Обновить» позволяет получить данные процессов из списка (List<Process>), а также данные об используемой ими памяти системы(рис. 7).

```
private void toolStripButton1_Click(object sender, EventArgs e)
{
    GetProcesses();
    RefreshProcessesList();
}
```

Рисунок 7. Обработчик события кнопки «Обновить»

Второй метод нажатие на кнопку «Завершить» имеет условие проверки выделения процесса. В случае, если процесс выделен в элементе ListView, то вызывается метод Where(), с помощью которого происходит получение процесса в списке(List) и сравнение значения списка и значения, которое находящееся в ListView. Для срабатывания функции завершение процесса используется перевод результата запроса к типу List. После завершения процесса происходит обновление списка List<Process> и списка ListView(рис. 8). Аналогичным образом прописывается метод нажатия на кнопку «Завершить дерево процессов». С помочью данных методов была запрограммирована верхняя панель кнопок (рис. 9).

```
private void toolStripButton2_Click(object sender, EventArgs e)
{
    try
    {
        if (listView1.SelectedItems[0] != null)
        {

            Process process To Kill = processes.Where((x) => x.Process Name ==
            listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

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

Рисунок 8. Обработчик события кнопки «Завершить»

```
private void toolStripButton3_Click(object sender, EventArgs e)
{
    try
    {
        if (listView1.SelectedItems[0] != null)
        {

            Process process To Kill = processes.Where((x) => x.Process Name ==
            listView1.SelectedItems[0].SubItems[0].Text).ToList()[0];

            KillProcessAndChildren(GetParentProcessId(processToKill));

            GetProcesses();

            RefreshProcesses List();
        }
    }
```

Рисунок 9. Обработчик события кнопки «Завершить дерево процессов»

Кнопки в элементе contextMenuStrip (вызываемые кнопки по нажатию правой кнопкой мыши на ячейку в ListView) – «Завершить» и «Завершить дерево процессов» имеют идентичные методы, как и у кнопок панели.

Для запуска нового процесса используется обработчик нажатия на кнопку «Запустить задачу». В данном методе вызывается окно с подсказкой для ввода имени процесса, а также запуск процесса по имени (рис. 10).

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

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

Рисунок 10. Обработчик нажатия кнопки «Запустить задачу»

Для поиска процесса по имени используется toolStripTextBox. В данном элементе имеется событие TextChanged, которое отвечает за изменение текста в элементе. Для данного события создаётся обработчик события (рис. 11).

```
private void toolStripTextBox1_TextChanged(object sender, EventArgs e)
{
    GetProcesses();
    List<Process> filteredprocesses = processes.Where((x) =>
    x.ProcessName.ToLower().Contains(toolStripTextBox1.Text.ToLower())).ToList<Process>();

    RefreshProcessesList(filteredprocesses, toolStripTextBox1.Text);
}
```

Рисунок 11. Обработчик события TextChanged

В данном обработчике происходит обновление списка, а также фильтрация. Для реализации фильтрации списка создаётся новый список List<Process> filteredprocesses и с помощью функции Where() осуществляется поиск по списку также, как и в методах завершение процессов находился процесс, который нужно было завершить.

Для выхода из приложения используется обработчик нажатия на кнопку «Выход», в которой содержится функция «Application.Exit()».

После завершения программирования функционала приложения происходит тестирование на практике (рис. 12, 13).

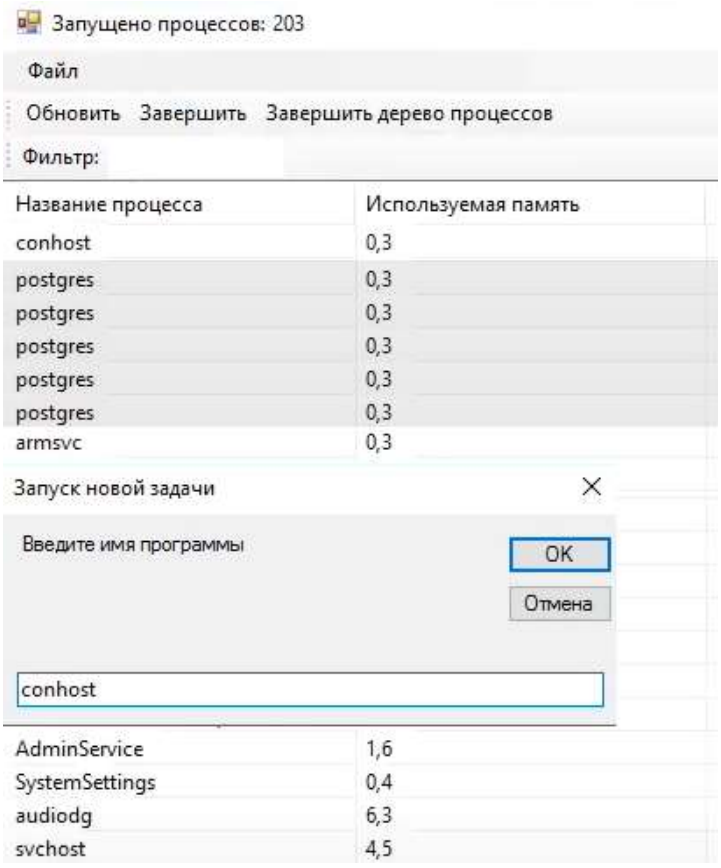

Рисунок 12. Результат работы приложения (1)

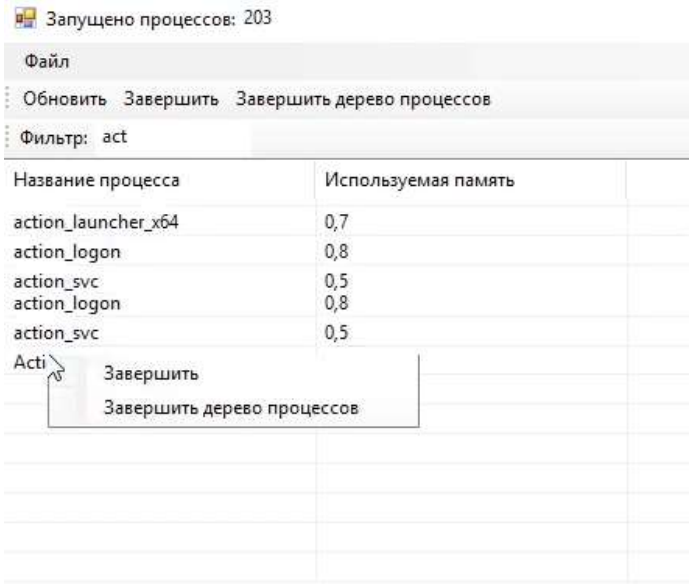

Рисунок 13. Результат работы приложения (2)

В результате исследования было создано приложения «Диспетчер задач» на языке программирования C# с функциями управления процессами ОС Windows такими, как завершение одного, нескольких и целого дерева процессов, добавление процессов и поиск по имени процесса. Также был описан процесс разработки приложения и принцип его работы.

Библиографический список

1. Ромашкина Т.В., Шагалиев В.И. Создание приложений для Windows средствами языка C# // Хроники объединенного фонда электронных ресурсов Наука и образование. 2015. № 12 (79). С. 115.
2. Куприенко В.М., Нечитайло Н.В. Разработка Windows-приложений на языке C# // В сборнике: Современные инновации в науке и технике. Сборник научных трудов 4-ой Международной научно-практической конференции: В 4-х томах. Ответственный редактор Горохов А.А., 2014. С. 340-343.
3. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. Разработка приложений на C# для работы с базами данных // Магнитогорск: Электронное издание: практикум, 2018. 170 с.
4. Ликнесс Д.Ж. Приложения для Windows 8 на C# и XAML // СПб: БХВ-Петербург, 2013. 368 с.
5. Баженов С. М. Преимущества языковой платформы C# (си шарп) в разработке многозадачных приложений для контроля качества // Редакционная коллегия. – С. 118.
6. Ерохин А.С. Создание оконного приложения на языке C# // В сборнике: Интеллектуальный потенциал XXI века инновационной России. Материалы VIII Всероссийской научно-практической конференции обучающихся и студентов, посвященной 100-летию ОГУ. 2019. С. 29-32.