|  |
| --- |
| Харьковский национальный университет радиоэлектроники |
| (полное название высшего ученого заведения) |
| Кафедра «Электронных вычислительных машин» |
| (полное название кафедры) |

**КУРСОВОЙ ПРОЕКТ**

**(РАБОТА)**

|  |  |
| --- | --- |
| по курсу: | Системное программное обеспечение |
|  | (название дисциплины) |
| на тему: | Разработка программного обеспечение для управления |
|  | запущенными на локальном и удаленном компьютере процессами. |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Студентки | 3 | | | курса | | КИ-15-4 | группы |
|  | направления подготовки | | | | Компьютерная инженерия и управление | | | |
|  | специальность | | Компьютерная инженерия | | | | | |
|  | Розенберг М. Э. | | | | | | | |
|  | (фамилия и инициалы) | | | | | | | |
|  | Руководитель |  | | | | | | |
|  |  | | | | | | | |
|  | (должность, ученое звание, научная степень, фамилия и инициалы) | | | | | | | |
|  | Национальная шкала | | | |  | | | |
|  | Количество балов: | | |  | | Оценка: ECTS | |  |
|  |  | | | | | | | |

г. Харьков – 2017 год

Харьковский национальный университет радиоэлектроники

(наименование высшего учебного заведения)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кафедра: | Электронных вычислительных машин | | | | |
| Дисциплина: | Системное программное обеспечение | | | | |
| Специальность: | Компьютерная инженерия | | | | |
| Курс: | 3 | Группа: | КИ-15-4 | Семестр: | 5 |

**З А Д А Н И Е**

на курсовой проект (работу) студента

Розенберг Мария Эдуардовна

(фамилия, имя, отчество)

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Тема проекта (работы) | Разработка программное обеспечение для управления | | |
| запущенными на локальном и удаленном компьютере процессами | | | |
| 2. Срок сдачи студентом законченного проекта (работы) | | |  |
| 3. Исходные данные к проекту (работе) | | Разработать программное обеспечение | |
| для управления запущенными на локальном и удаленном компьютере процессами. | | | |
| Программное обеспечение предоставляет возможности по просмотру списка | | | |
| запущенных процессов, созданию новых и удаления (остановки) выполняемых. | | | |
| Пользователь может задать произвольное количество удаленных компьютеров, за | | | |
| которыми производится наблюдение. Для каждого компьютера пользователь может | | | |
| определить список «запрещенных» процессов, которые останавливаются сразу как | | | |
| только обнаруживаются. | | | |
| 4. Содержание курсовой работы: | | | |
| Разработка механизма взаимодействие с процессами на локальном и удаленном | | | |
| компьютере. Разработка графического пользовательского интерфейса, с | | | |
| обеспечивающего взаимодействие пользователем. | | | |
| 5. Дата выдачи задания: |  | | |

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № этапа | Содержание выполняемых работ | Период | Отметка |
| 1 | Выдача заданий | 2-я неделя | выполнено |
| 2 | Ознакомление с литературными источниками, анализ и выбор метода решения поставленной задачи | 3-4 неделя | выполнено |
| 3 | Разработка алгоритмов решения, выбор системных средств решения задач курсового проектирования | 5-6 неделя | выполнено |
| 4 | Проектирование приложения | 7-11 неделя | выполнено |
| 5 | Отладка и тестирование приложения | 12-14 неделя | выполнено |
| 6 | Оформление пояснительной записки | 15 неделя | выполнено |
| 7 | Защита курсового проекта | 16-17 неделя |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

РЕФЕРАТ

Пояснительная записка содержит 31 страницу, 3 рисунка, 4 раздела, 1 приложение, 5 источников литературы.

Цель работы — разработать программное обеспечение для управления запущенными на локальном и удаленном компьютере процессами.

Метод исследования – изучение литературы, написание и отладка программы на компьютере.

Программа написана на языке C# в среде разработки Visual Studio 2015.

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ, VISUAL STUDIO 2015, ОПЕРАЦИОННАЯ СИСТЕМА, C#, WMI, WINDOWS FORMS.

Оглавление

[Введение 6](#_Toc502754266)

[1 Анализ предметной области и постановка задач 8](#_Toc502754267)

[2 Разработка структуры и алгоритма программы 10](#_Toc502754268)

[3 Опис програмної реалізації 13](#_Toc502754269)

[4 Інструкція користувача 18](#_Toc502754270)

[Висновки 21](#_Toc502754271)

[Перелік посилань 22](#_Toc502754272)

[Додаток А Лicтинг чacтини кoду 23](#_Toc502754273)

# Введение

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. Процесс (или по-другому, задача) — абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами. Обычно при загрузке ОС создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, обеспечивающими взаимодействие с пользователями и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции – например, связанные с электронной почтой, Web-страницами, выводом на печать, передачей файлов по сети, периодическим запуском программ (например, дефрагментации дисков) и т.д.

Диспетчер задач — компьютерная программа (утилита) для вывода на экран списка запущенных процессов и потребляемых ими ресурсов. В качестве дополнительных функций, диспетчер задач может предложить возможность завершить один из процессов или присвоить ему другой приоритет. На данный момент сложно представить использование компьютера без диспетчера задач, так как очень часто возникает необходимость просмотреть список запущенных в данный момент задач, создать новую задачу или остановить ненужную задачу. Windows Task Manager в Windows NT можно вызвать, одновременно нажав клавиши Ctrl+Shift+Esc. В Windows NT и в Windows XP существует более известная комбинация клавиш — Ctrl+Alt+Del. Диспетчер задач можно также запустить в командной строке, введя имя его исполняемого файла (taskmgr.exe) или выбрав соответствующий пункт в контекстном меню панели задач.

Однако встроенная утилита позволяет просматривать только задачи, запущенные на данном компьютере. Это становится не очень удобным, когда необходимо управлять сразу несколькими компьютерами и следить за их работой и запускать там новые задачи.

Поэтому целью данной курсовой работы, является описание разработки приложения, которое позволит пользователю управлять процессами как на своем локальном компьютере, так и на любом другом удаленном компьютере, к которому у него есть доступ. Данное приложение является довольно актуальным так как существует мало приложений способных управлять процессами на удаленном компьютере, а так же следить за списком запущенных задач и закрывать «запрещенные» приложения.

# 1 Анализ предметной области и постановка задач

В данной работе рассматривается вопрос создания приложения, которое позволит пользователю управлять процессами как на своем локальном компьютере, так и на любом другом удаленном компьютере, к которому у него есть доступ.

Сферой использования программы может быть, как обычное использование пользователем для просмотра списка доступных процессов на компьютере, так и использование данной утилиты администраторами серверов для мониторинга доступных процессов на нескольких удаленных компьютерах сразу. А также данное приложение может быть использовано в качестве приложения для контроля за работой пользователя и запрета для него использования сторонних приложений, которые не разрешены.

В результате выполнения данной курсовой работы программа должна обеспечивать следующий функционал:

* отображать запущенные процессы на выбранном компьютере;
* запускать заданные приложение на выбранном компьютере;
* останавливать выбранный процесс;
* добавлять настройки для подключения приложения к другим компьютерам для мониторинга запущенных задач;
* мониторить список запущенных задач и останавливать задачи, которые входят в список «запрещенных» задач.

Для разработки данной программы использовалось Windows Forms приложение, написанное на языке C#. Основное окно состоит из кнопок меню для настройки приложения и из вкладок с информацией по процессам для каждого настроенного подключения к компьютеру. Для работы со список задач операционной системы будет использоваться технология Windows Management Instrumentation (WMI). Это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows.

Для постоянного обновления списка задач будет использован стандартный компонент Windows Forms — Timer, он позволяет выполнять любое действие через определенные промежутки времени.

Список доступных задач будет отображаться на экране с помощью встроенного элемента управления — DataGridView. Это обычная таблица, которая имеет возможность сортировки колонок, а также позволяет изменять ширину и высоту ячеек таблицы.

В основе всей программы, лежит вполне базовая, не сложная реализация, которая состоит из множества условий, циклов и переключателей.

Для отображения всплывающих уведомлений или сообщений об ошибках подключений, используются диалоговые окна, создаваемые с помощью класса «MessageBox».

# 2 Разработка структуры и алгоритма программы

Разрабатываемое приложение состоит из одного главного окна в котором содержится вся информацию о процессах, а также программа содержит вспомогательную форму для добавления к приложению дополнительных компьютеров, за которыми необходимо вести наблюдение. Содержимое каждой вкладки с процессами отображается с помощью пользовательского элемента, который содержит необходимые функции для управления процессами и таблицы для отображения текущих задач. Данные для этих таблиц заполняются автоматически при использовании встроенного механизма в Windows Forms — Data Binding.

Для более хорошего структурирования программы, был использован объектно-ориентированный подход, а именно созданы классы для процессов, настроек локального и удаленного компьютеров, которые хранят в себе все необходимые данные. Так как есть возможность подключаться как к локальному компьютеру, так и удаленному компьютеру, то был реализован базовый класс для работы с любыми процессами. А также был реализован еще один класс для обработки процессов на локальном компьютере, который наследует базовый класс обработки задач и использует более простую логику для получения списка задач. Данная структура приложения позволит в будущем в случае необходимости добавлять новые типы компьютеров, в которых способ обработки процессов осуществляется иначе.

Так же, для более удобной отладки и использования программного кода, программа разбита на отдельные файлы:

* Program.cs — начальная точка входа в программу, это автоматически созданный файл, который отвечает за создание основного окна и отслеживания его событий.
* MainForm.cs — файл, содержащий информацию о основном окне программы, в котором можно добавлять конфигурации новых подключений и удалять ненужные;
* ComputerSettings.cs — содержит информацию о окне для редактирования и создания конфигураций подключения;
* ComputerView.cs — это пользовательский элемент управления, который отвечает за отображения процессов, их обновление, остановку и добавление новых;
* ProcessModel.cs — это класс содержащий всю необходимую информацию для какого-либо процесса;
* Computer.cs — это базовый класс для работы с процессами на компьютере;
* LocalComputer.cs — это класс для управления процессами на локальном компьютере.

Основная обработка действий программы происходит в пользовательском элементе управления ComputerView для каждого добавленного компьютера.

В разработанной программе используется базовый, не сложный алгоритм для каждого подключения к компьютеру:

* Для начала создается вкладка на главном окне, на которой будут отображаться все доступные процессы и где находится функционал для управления этими процессами.
* После этого создается таймер, который будет через равные промежутки времени опрашивать состояние процессов и обновлять их список в основной таблице.
* После запуска таймера начинается обновление всех процессов для выбранного компьютера. Для перерисовки таблицы выполняется несколько простых операций: в начале запоминается текущее положение пользователя, потом обновляется содержимое таблицы и потом положение в таблице возвращается к первоначальному.
* Когда пользователь закрывает подключение к выбранному компьютеру, с помощью меню, то освобождаются ресурсы текущего элемента управления и останавливается таймер, который отвечал за обновление процессов. И потом содержимое данной вкладки удаляется с главного окна.

В разработанной программе присутствует несложные алгоритмы для мониторинга запущенных процессов и их остановки, если они распознаны как «запрещенные». Для этого у каждой настройки подключения есть список процессов, которые запрещены. И при каждом обновления списка процессов каждый процесс проверяется не находится ли он в запрещенном списке. И если данный процесс запрещен, то посылается запрос на компьютер о его немедленной остановке.

# 3 Опис програмної реалізації

Програма, яка була створена, при виконанні курсового проекту, створювалася в середовищі розробки Visual Studio 2015 року, засобами Windows Forms, на мові С#.

Основна суть програми — це робота з процесами на локальному і віддаленому комп'ютері. Для виконання поставленого завдання була використана вбудована в Windows технологія — WMI.

Windows Management Instrumentation (WMI) — це одна з базових технологій для централізованого управління і стеження за роботою різних частин комп'ютерної інфраструктури під управлінням платформи Windows.

Технологія WMI — це розширена і адаптована під Windows реалізація стандарту WBEM, прийнятого багатьма компаніями, в основі якого лежить ідея створення універсального інтерфейсу моніторингу та управління різними системами і компонентами розподіленого інформаційного середовища підприємства з використанням об'єктно-орієнтованих ідеологій і протоколів HTML і XML.

В основі структури даних в WBEM лежить Common Information Model (CIM), що реалізує об'єктно-орієнтований підхід до подання компонентів системи. CIM є розширюваної моделлю, що дозволяє програмам, системам і драйверам додавати в неї свої класи, об'єкти, методи і властивості.

WMI, заснований на CIM, також є відкритою уніфікованою системою інтерфейсів доступу до будь-яких параметрах операційної системи, пристроїв і додатків, які функціонують в ній.

Так як WMI побудований за об'єктно-орієнтованому принципом, то всі дані операційної системи представлені у вигляді об'єктів і їх властивостей і методів.

Всі класи групуються у просторі імен (namespace), які ієрархічно впорядковані і логічно пов'язані один з одним за певною технологією або галуззю управління. У WMI є один кореневий простір з іменем Root, який в свою чергу має 4 підпростори: CIMv2, Default, Security і WMI.

В ході виконання курсового проекту використовувалося одне з цих підпросторів - CIMv2.

Класи мають властивості і методи і знаходяться в ієрархічній залежності один від одного, тобто класи-нащадки можуть успадковувати або перевизначати властивості класів-батьків, а також додавати свої властивості. Властивості класів використовуються для однозначної ідентифікації примірника класу і для опису стану використовуваного ресурсу. Зазвичай всі властивості класів доступні тільки для читання, хоча деякі з них можна модифікувати певним методом. Методи класів дозволяють виконати дії над керованим ресурсом.

Для звернення до об'єктів WMI використовується специфічна мова запитів WMI Query Language (WQL), яка є одним із різновидів SQL. Основна її відмінність від SQL - це неможливість зміни даних, тобто за допомогою WQL можлива лише вибірка даних за допомогою команди SELECT. Крім обмежень на роботу з об'єктами, WQL не підтримує такі оператори як DISTINCT, JOIN, ORDER, GROUP, математичні функції.

WMI надає простий інтерфейс для взаємодії з даними про управління компонентів ОС, який включає:

* Повнофункціональну і узгоджену модель поведінки, конфігурацій і стану операційної системи Windows.
* API COM, який надає єдину точку доступу до всієї інформації про управління.
* Взаємодія з іншими службами управління Windows.
* Гнучку архітектуру, що дозволяє постачальникам поширити інформаційну модель на нові пристрої, додатки і служби.
* Повнофункціональна мова запитів, за допомогою якої можна створювати засновані на SQL запити інформаційної моделі.
* API з можливістю створення сценаріїв, що забезпечує просте локальне та віддалене адміністрування систем.

Основний клас, з яким доведеться працювати за допомогою WQL запитів це Win32\_Process. Win32\_Process — це WMI клас, який відображає процес в операційній системі. Даний клас має набір властивостей, за допомогою яких можна отримати всю інформацію про процес, наприклад, ім'я запущеного файлу, аргументи, з якими він був запущений, статус процесу, його Id, дату його створення, обсяг займаної віртуальної пам'яті і багато іншого. Також даний клас містить набір метод для роботи з процесами: методи для створення процесу, зупинки, отримання власника даного процесу і метод для виставлення пріоритету для обраного процесу.

Для початку розглянемо створені в програмі для зручності додаткові файли класів об'єктів:

Файл ProcessModel.cs містить клас для зберігання всієї необхідної інформації для процесу, а також в ньому перевизначений метод Equals, що дозволяє порівнювати об'єкти цього типу між собою.

public class ProcessModel

{

public int Id { get; set; }

public string Name { get; set; }

public string Path { get; set; }

public string Arguments { get; set; }

public override bool Equals(object obj)

{

if (obj is ProcessModel)

return Equals((ProcessModel)obj);

return false;

}

private bool Equals(ProcessModel other)

{

return Id == other.Id

&& string.Equals(Name, other.Name)

&& string.Equals(Path, other.Path)

&& string.Equals(Arguments, other.Arguments);

}

}

Пример 3.1

Файл Computer.cs містить базовий клас для управління процесами і для зберігання налаштувань підключення.

public class Computer

{

public Computer(string name, string userName, string password, int loadInterval)

{

Name = name;

UserName = userName;

Password = password;

LoadInterval = loadInterval;

RestrictedProcesses = new string[0];

}

public string Name { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string[] RestrictedProcesses { get; set; }

public int LoadInterval { get; set; }

...

}

Приклад 3.2

Цей клас також містить і набір методів для управління процесами. Базова реалізація цих методів використовує WMI для управління процесами. Цю реалізацію при необхідності можна перевизначити в дочірньому класі, як наприклад це зроблено для підключення до локального комп'ютера. І для локального підключення вже не використовується технологія WMI, а використовується стандартний клас .Net Framework: Process, який вміє створювати процеси, зупиняти процеси і відображати список запущених процесів.

Розглянемо метод для відображення списку увімкнених процесів.

public virtual ProcessModel[] GetAllProcesses()

{

var result = new List<ProcessModel>();

var scope = GetManagementScope();

var query = new ObjectQuery("Select ProcessId, ExecutablePath, Name, CommandLine From Win32\_Process");

using (var searcher = new ManagementObjectSearcher(scope, query))

{

foreach (ManagementObject details in searcher.Get())

{

result.Add(new ProcessModel

{

Id = Convert.ToInt32(details["ProcessId"].ToString()),

Name = details["Name"].ToString(),

Arguments = details["CommandLine"]?.ToString(),

Path = details["ExecutablePath"]?.ToString()

});

}

}

return result.ToArray();

}

Приклад 3.3

На початку цього методу виконується створення необхідних змінних для його виконання: створюється список для зберігання запущених процесів, відкривається підключення до віддаленого комп'ютерів, а також формується SQL-запит для отримання списку процесів. Після цього виконується сформований запит і за допомогою foreach циклу обробляється кожен рядок результату і перетворюється в модель ProcessModel, яка потім буде відображатися в таблиці користувачу.

# 4 Інструкція користувача

Для коректної роботи програми та моніторингу процесів на локальному комп'ютері бажано запускати додаток від імені адміністратора для запобігання можливих проблем з правами доступу до процесів. А для управління процесами на віддаленому комп'ютері необхідно знати його IP-адресу або ім'я комп'ютера, так само потрібно мати логін і пароль від облікового запису, який має доступ до віддаленого виконання WMI запитів. Якщо всі ці умови виконані, то додаток зможе управляти процесами на віддаленому комп'ютері.

При виконанні програми автоматично додається вкладка з налаштуваннями для управління локальним комп'ютером. Основне вікно програми представлено на рис 4.1.

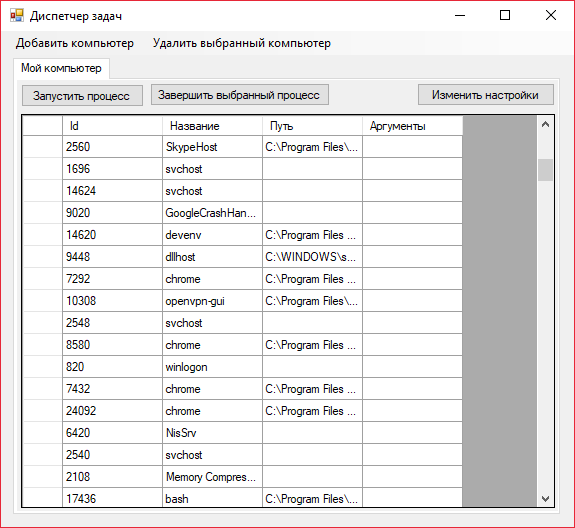


Рисунок 4.1 — Головне вікно програми

Даний список процесів можна впорядкувати за будь-якою з колонок. Так само на цій сторінці можна завершити обраний в таблиці процес. Запустити новий процес можна при натисканні на кнопку «Запустити процес». Після цього відкриється діалогове вікно (див. рис. 4.2) для введення шляху до додаток, який буде виконуватися і при натисканні ОК даний процес буде негайно запущений.

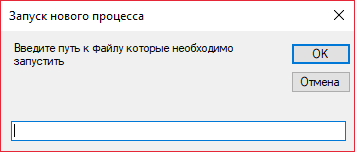


Рисунок 4.2 — Діалогове вікно для запуску процесів

Для того, щоб додати новий комп'ютер необхідно вибрати пункт меню «Додати комп'ютер». Після цього відкриється діалогове вікно, в якому необхідно заповнити конфігураційні поля для підключення (див. рис. 4.3).

Для підключення до віддаленого комп'ютера необхідно ввести реальне ім'я комп'ютера або його адресу в мережі, а також потрібно ввести облікові дані користувача, за допомогою яких можна управляти процесами на віддаленому комп'ютері. За допомогою цього вікна так само можна вказати список програм, який повинні бути негайно зупинені у разі їх виявлення. Так само тут можна вказати частоту оновлення списку завдань.

Всі ці налаштування можна буде змінити в будь-який момент при натисканні кнопки «Змінити налаштування» для конкретного комп'ютера.

Після натискання на кнопку «Зберегти» в додатку буде додана нова вкладка з тільки що введеними даними.

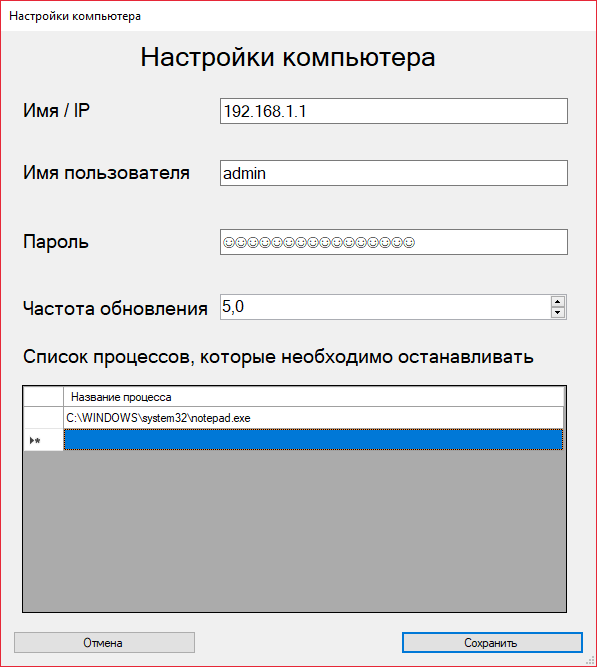


Рисунок 4.2 — Вікно налаштувань підключення

Кожне з доданих підключень можна закрити в будь-який момент. Для цього всього лише досить натиснути на кнопку «Видалити обраний комп'ютер». Після цього буде видалена вкладка з поточним комп'ютером і зупинений таймер, який оновлює список процесів.

# Висновки

В ході виконання даної роботи в середовищі Visual Studio 2015 з допомогою мови програмування C# і засобів Windows Forms була розроблена програма, яка дозволяє керувати запущеними процесами.

Використання даної програми, дає можливість заощадити багато часу на управління процесами на багатьох серверах одночасно, а також дає можливість зупиняти непотрібні процеси автоматично. Програма проста у використанні, так як в ній інтуїтивно зрозумілий інтерфейс, також була складена для користувача інструкція використання програми.

Розроблений код програми був розбитий на кілька файлових блоків, за прийнятим стилем програмування. Створено основний блок виконання програми, а також додаткові блоки, які містять в собі реалізацію опису об'єктів програми, їх методи і їх реалізацію.

На даному етапі проектування, функціонал програми логічно завершений. При продовженні роботи над даним проектом, в сторону розширення функціоналу, можливо додавання різних функцій, а саме: взаємодія з іншими операційними системами, підтримка збереження та завантаження налаштувань для підключення до комп'ютерів, відображення додаткової інформації по кожному з процесів, напр. відсотків завантаження CPU, кількість запущених потоків всередині процеса і т.д.

У своїй структурі програма не має складних алгоритмів, використані функції легкі у використанні, до того ж, функції стандартної бібліотеки Windows докладно описані в багатьох джерелах. Це дозволяє без особливих проблем написати або відредагувати код програми людині з базовими знаннями мови С#.

# Перелік посилань

1. Троелсен, Е. Мова програмування C# 5.0 і платформа .NET 4.5, 6 вид. [Текст]: пер. з англ.- М.: Вільямс, 2013 – 1312 с.
2. Уотсон, К., Нейгел, К., Педерсен, Я. Visual C# 2010: повный курс. [Текст]: Пер. з англ. – М.:ООО «И.Д. ільямс», 2011 – 960 с.
3. Шилдт Г. C# 4.0.: Повне керівництво [Текст]: пер. з англ.- М.: Вильямс, 2011 – 1056 с.
4. Попов, А., Шикин, Е. Адміністрування Windows за домогою WMI та WMIC [Текст]: 2004 – 752 с.
5. Петцольд, Ч., Програмування з використанням Microsoft Windows Forms [Текст]: 2006 – 433 с.

# Додаток А Лicтинг чacтини кoду

Файл Computer.cs

using System;

using System.Collections.Generic;

using System.Management;

namespace ProcessManager.Models

{

public class Computer

{

public Computer(string name, string userName, string password, int loadInterval)

{

Name = name;

UserName = userName;

Password = password;

LoadInterval = loadInterval;

RestrictedProcesses = new string[0];

}

public string Name { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string[] RestrictedProcesses { get; set; }

public int LoadInterval { get; set; }

public virtual ProcessModel[] GetAllProcesses()

{

var result = new List<ProcessModel>();

var scope = GetManagementScope();

var query = new ObjectQuery("Select ProcessId, ExecutablePath, Name, CommandLine From Win32\_Process");

using (var searcher = new ManagementObjectSearcher(scope, query))

{

foreach (ManagementObject details in searcher.Get())

{

result.Add(new ProcessModel

{

Id = Convert.ToInt32(details["ProcessId"].ToString()),

Name = details["Name"].ToString(),

Arguments = details["CommandLine"]?.ToString(),

Path = details["ExecutablePath"]?.ToString()

});

}

}

return result.ToArray();

}

public virtual void RunProcess(string fullName)

{

var scope = GetManagementScope();

var options = new ObjectGetOptions();

using (ManagementClass processClass = new ManagementClass(scope, new ManagementPath("Win32\_Process"), options))

{

var parameters = processClass.GetMethodParameters("Create");

var taskName = Guid.NewGuid().ToString();

var prepareCommand = $"schtasks /Create /sc daily /tn {taskName} /tr \"{fullName}\" /it";

var executeCommand = $"schtasks /run /tn {taskName} /i";

var deleteCommand = $"schtasks /delete /tn {taskName} /f";

parameters["CommandLine"] = prepareCommand;

processClass.InvokeMethod("Create", parameters, null);

parameters["CommandLine"] = executeCommand;

processClass.InvokeMethod("Create", parameters, null);

parameters["CommandLine"] = deleteCommand;

processClass.InvokeMethod("Create", parameters, null);

}

}

public virtual void KillProcess(int processId)

{

var scope = GetManagementScope();

var query = new ObjectQuery("Select \* From Win32\_Process where ProcessId = " + processId);

using (var searcher = new ManagementObjectSearcher(scope, query))

{

foreach (ManagementObject details in searcher.Get())

{

details.InvokeMethod("Terminate", null);

}

}

}

private ManagementScope GetManagementScope()

{

ConnectionOptions connection = new ConnectionOptions();

if (!string.IsNullOrWhiteSpace(UserName) || !string.IsNullOrWhiteSpace(Password))

{

connection.Username = UserName;

connection.Password = Password;

}

ManagementScope scope = new ManagementScope("\\\\" + Name + "\\root\\CIMV2", connection);

scope.Connect();

return scope;

}

}

}

Файл LocalComputer.cs

using System;

using System.Diagnostics;

using System.Linq;

namespace ProcessManager.Models

{

public class LocalComputer : Computer

{

public LocalComputer() : base("Мой компьютер", null, null, 5) { }

public override ProcessModel[] GetAllProcesses()

{

return Process

.GetProcesses()

.Select(p=> new ProcessModel

{

Id = p.Id,

Name = p.ProcessName,

Path = GetProcessFileName(p),

Arguments = p.StartInfo.Arguments

})

.ToArray();

}

public override void KillProcess(int processId)

{

Process.GetProcessById(processId).Kill();

}

public override void RunProcess(string fullName)

{

Process.Start(fullName);

}

private string GetProcessFileName(Process process)

{

try

{

return process.MainModule.FileName;

}

catch (Exception)

{

return string.Empty;

}

}

}

}

Файл ProcessModel.cs

namespace ProcessManager.Models

{

public class ProcessModel

{

public int Id { get; set; }

public string Name { get; set; }

public string Path { get; set; }

public string Arguments { get; set; }

public override bool Equals(object obj)

{

if (obj is ProcessModel)

{

return Equals((ProcessModel)obj);

}

return false;

}

private bool Equals(ProcessModel other)

{

return Id == other.Id

&& string.Equals(Name, other.Name)

&& string.Equals(Path, other.Path)

&& string.Equals(Arguments, other.Arguments);

}

}

}

Файл MainForm.cs

using System;

using System.Windows.Forms;

using ProcessManager.Models;

namespace ProcessManager

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

AddComputerTabPage(new LocalComputer());

}

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

{

var form = new ComputerSettings();

if (form.ShowDialog() == DialogResult.OK)

{

var computer = form.GetComputer();

AddComputerTabPage(computer);

}

}

private void AddComputerTabPage(Computer computer)

{

var page = new TabPage(computer.Name);

var view = new ComputerView(computer, page);

view.Dock = DockStyle.Fill;

page.Controls.Add(view);

tabControl1.TabPages.Add(page);

}

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

{

if (tabControl1.TabPages.Count > 0)

{

var tab = tabControl1.SelectedTab;

tabControl1.TabPages.Remove(tab);

tab.Dispose();

}

}

}

}

Файл ComputerSettings.cs

using System.Windows.Forms;

using ProcessManager.Models;

namespace ProcessManager

{

public partial class ComputerSettings : Form

{

private readonly Computer \_initialComputer;

public ComputerSettings()

{

InitializeComponent();

}

public ComputerSettings(Computer computer) : this()

{

\_initialComputer = computer;

nameTextBox.Text = computer.Name;

userNameTextBox.Text = computer.UserName;

passwordTextBox.Text = computer.Password;

loadIntervalControl.DecimalPlaces = computer.LoadInterval;

foreach (var restrictedProcess in \_initialComputer.RestrictedProcesses)

{

dataGridView1.Rows.Add(restrictedProcess);

}

}

public Computer GetComputer()

{

if (\_initialComputer != null)

{

\_initialComputer.Name = nameTextBox.Text;

\_initialComputer.Password = passwordTextBox.Text;

\_initialComputer.UserName = userNameTextBox.Text;

\_initialComputer.LoadInterval = loadIntervalControl.DecimalPlaces;

\_initialComputer.RestrictedProcesses = GetRestrictedProcesses();

return \_initialComputer;

}

var result = new Computer(nameTextBox.Text, userNameTextBox.Text, passwordTextBox.Text, loadIntervalControl.DecimalPlaces);

result.RestrictedProcesses = GetRestrictedProcesses();

return result;

}

private string[] GetRestrictedProcesses()

{

var result = new string[dataGridView1.RowCount - 1];

for (int i = 0; i < dataGridView1.RowCount - 1; i++)

{

result[i] = dataGridView1.Rows[i].Cells[0].Value.ToString().Trim().ToLower();

}

return result;

}

}

}