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

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 ОПИСАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Программа, которая была создана, при выполнении курсового проекта, создавалась в среде разработки Visual Studio 2015, средствами Windows Forms, на языке C#.

Основная суть приложения — это работа с процессами на локальном и удаленном компьютере. Для выполнения поставленной задачи была использована встроенная Windows технология — WMI.

Windows Management Instrumentation (WMI) — это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows.

Технология WMI — это расширенная и адаптированная под Windows реализация стандарта WBEM, принятого многими компаниями, в основе которого лежит идея создания универсального интерфейса мониторинга и управления различными системами и компонентами распределенной информационной среды предприятия с использованием объектно-ориентированных идеологий и протоколов HTML и XML.

В основе структуры данных в WBEM лежит Common Information Model (CIM), реализующая объектно-ориентированный подход к представлению компонентов системы. CIM является расширяемой моделью, что позволяет программам, системам и драйверам добавлять в неё свои классы, объекты, методы и свойства.

WMI, основанный на CIM, также является открытой унифицированной системой интерфейсов доступа к любым параметрам операционной системы, устройствам и приложениям, которые функционируют в ней.

Так как WMI построен по объектно-ориентированному принципу, то все данные операционной системы представлены в виде объектов и их свойств и методов.

Все классы группируются в пространства имен, которые иерархически упорядочены и логически связаны друг с другом по определенной технологии или области управления. В WMI имеется одно корневое пространство имен Root, которое в свою очередь имеет 4 подпространства: CIMv2, Default, Security и WMI.

В ходе выполнения курсового проекта использовалось одно из этих подпространств — CIMv2.

Классы имеют свойства и методы и находятся в иерархической зависимости друг от друга, то есть классы-потомки могут наследовать или переопределять свойства классов-родителей, а также добавлять свои свойства. Свойства классов используются для однозначной идентификации экземпляра класса и для описания состояния используемого ресурса. Обычно все свойства классов доступны только для чтения, хотя некоторые из них можно модифицировать определенным методом. Методы классов позволяют выполнить действия над управляемым ресурсом.

Для обращения к объектам WMI используется специфический язык запросов WMI Query Language (WQL), который является одной из разновидностей SQL. Основное его отличие от SQL — это невозможность изменения данных, то есть с помощью WQL возможна лишь выборка данных с помощью команды SELECT. Помимо ограничений на работу с объектами, WQL не поддерживает такие операторы как DISTINCT, JOIN, ORDER, GROUP, математические функции.

WMI предоставляет простой интерфейс для взаимодействия с данными об управлении компонентов ОС, который включает:

— Полнофункциональную и согласованную модель поведения, конфигурации и состояния операционной системы Windows.

— API COM, который предоставляет единую точку доступа ко всей информации об управлении.

— Взаимодействие с другими службами управления Windows.

— Гибкую архитектуру, позволяющую поставщикам распространить информационную модель на новые устройства, приложения и службы.

— Полнофункциональный язык запросов, с помощью которого можно создавать основанные на SQL запросы информационной модели.

— API с возможностью создания сценариев, обеспечивающий простое локальное и удаленное администрирование систем.

Основной класс, с которым предстоит работать посредством WQL запросов это Win32_Process. Win32_Process — это WMI класс, который отображает процесс в операционной системе. Данный класс имеет набор свойств, с помощью которых можно получить всю информацию о процессе, например, имя запущенного файла, аргументы, с которыми он был запущен, статус процесса, его Id, дату его создания объем занимаемой виртуальной памяти и многое другое. Также данный класс содержит набор метод для работы с процессами: методы для создания процесса, остановки, получения владельца данного процесса и метод для выставления приоритета для выбранного процесса.

Для начала рассмотрим созданные в программе для удобства дополнительные файлы классов объектов:

Файл ProcessModel.cs содержит класс для хранения всей необходимой информации для процесса, а также в нем переопределен метод Equals, что позволяет сравнивать объекты этого типа между собой.

```
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 содержит базовый класс для управления процессами и для хранения настроек подключения.

```
    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; }
        ...
    }
```

Пример 3.2

Этот класс так же содержит и набор методов для управления процессами. Базовая реализация этих методов использует WMI для управления процессами.

Эту реализацию при необходимости можно переопределить в дочернем классе, как например это сделано для подключения к локальному компьютеру. И для локального подключения уже не используется технология WMI, а используется стандартный класс .Net Framework: Process, который умеет создавать процессы, останавливать процессы и отображать список запущенных процессов.

Рассмотрим метод для отображения списка запущенных процессов.

```
      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();
      }
```

Пример 3.3

В начале этого метода выполняется создание необходимых переменных для его выполнения: создается список для хранения запущенных процессов, открывается подключение к удаленному компьютеров, а также формируется SQL-запрос для получения списка процессов. После этого выполняется сформированный запрос и с помощью foreach цикла обрабатывается каждая строчка результата и преобразовывается в модель ProcessModel, которая потом будет отображаться в таблице пользователю.

4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Для корректной работы приложения и мониторинга процессов на локальном компьютере желательно запускать приложение от имени администратора для предотвращения возможных проблем с правами доступа к процессам. А для управления процессами на удаленном компьютере необходимо знать его IP-адрес или имя компьютера, так же нужно иметь логин и пароль от учетной записи, которая имеет доступ к удаленному выполнению WMI запросов. Если все эти условия выполнены, то приложение сможет управлять процессами на удаленном компьютере.

При запуске приложения автоматически добавляется вкладка с настройками для управления локальным компьютером. Основное окно приложения представлено на рисунке 4.1.

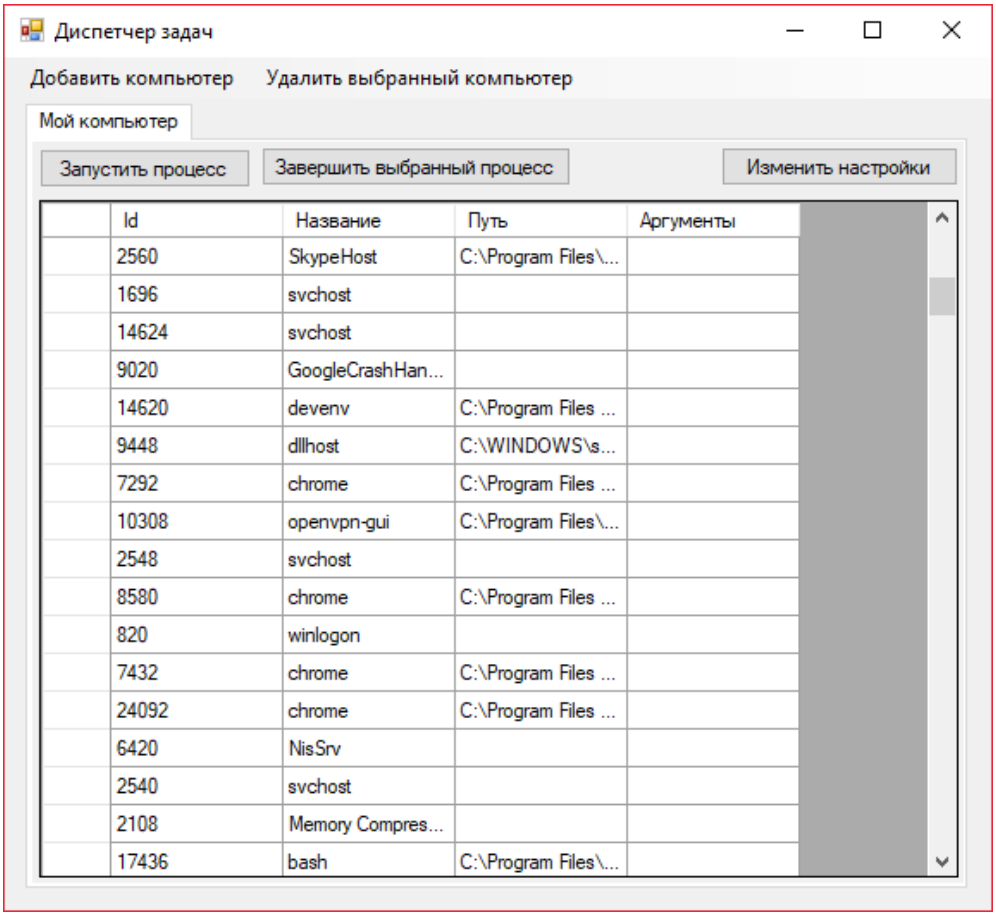

Рисунок 4.1 — Основное окно программы

Данный список процессов можно отсортировать по любой из колонок. Так же на этой странице можно завершить выбранный в таблице процесс. Запустить новый процесс можно при нажатии на кнопку «Запустить процесс». После этого откроется диалоговое окно (см. рис. 4.2) для ввода пути запускаемого приложение и при нажатии ОК данный процесс будет немедленно запущен.

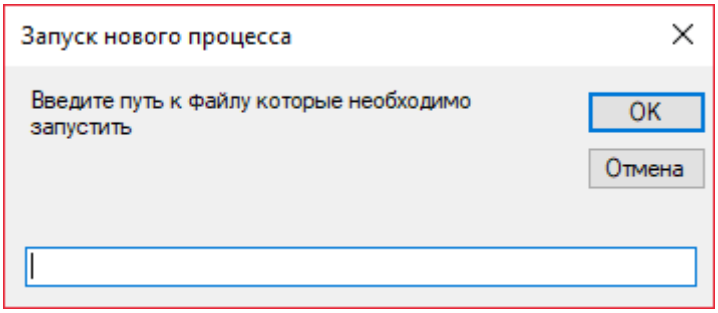

Рисунок 4.2 — Диалоговое окно для запуска процессов

Для того, чтоб добавить новый компьютер необходимо выбрать пункт меню «Добавить компьютер». После этого откроется диалоговое окно, в котором необходимо заполнить конфигурационные поля для подключения (см. рис. 4.3).

Для подключения к удаленному компьютеру необходимо ввести реальное имя компьютера или его адрес в сети, а также требуется ввести учетные данные пользователя, с помощью которого можно управлять процессами на удаленном компьютере. С помощью этого окна так же можно указать список приложений, который должны быть немедленно остановлены в случае их обнаружения. Так же здесь можно указать частоту обновления списка задач.

Все эти настройки можно будет изменить в любой момент при нажатии кнопки «Изменить настройки» для конкретного компьютера.

После нажатия на кнопку «Сохранить» в приложении будет добавлена новая вкладка с только что введенными данными.

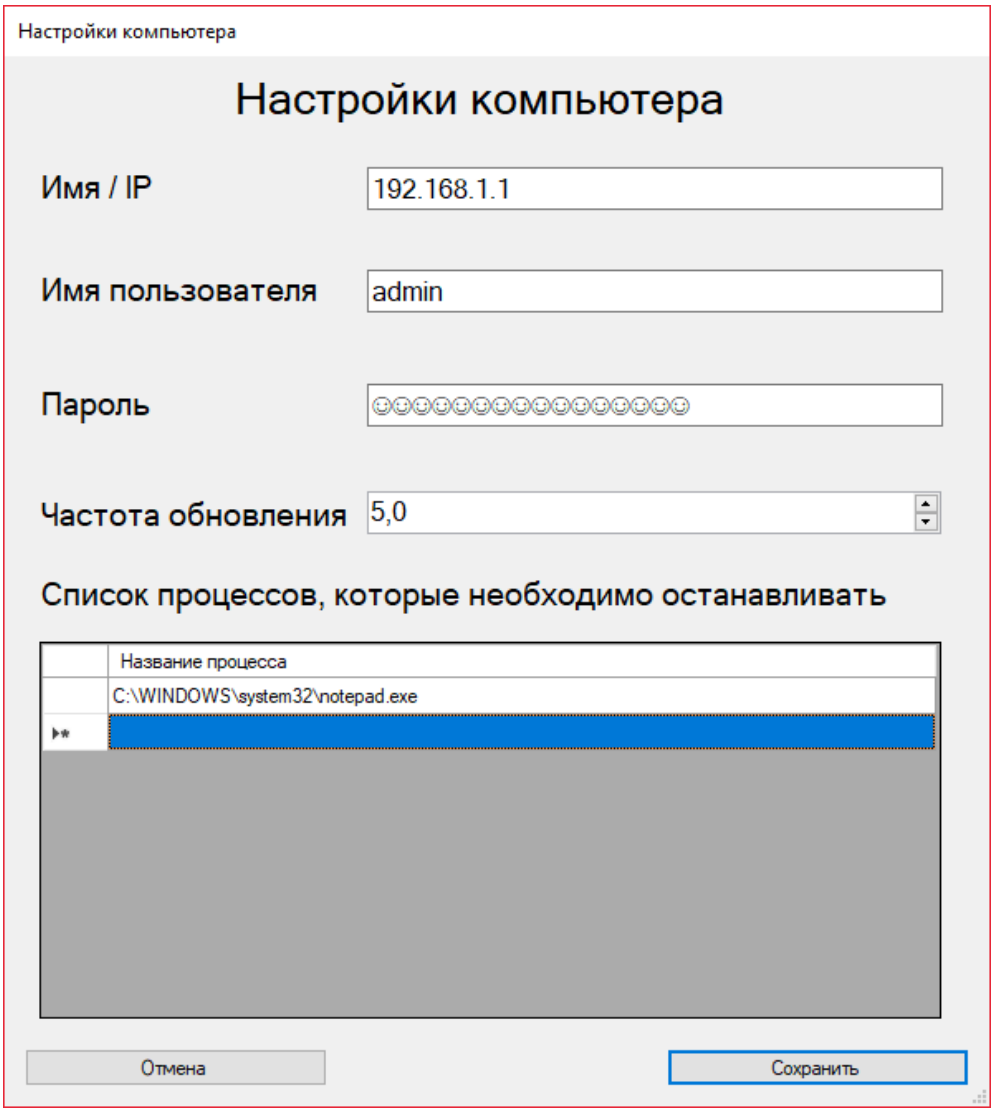

Рисунок 4.2 — Окно настройки подключения

Каждое из добавленных подключений можно закрыть в любой момент. Для этого всего лишь достаточно нажать на кнопку «Удалить выбранный компьютер». После этого будет удалена вкладка с текущим компьютером и остановлен таймер для обновления списка процессов.

ВЫВОДЫ

В ходе выполнения данной работы в среде Visual Studio 2015 с помощью языка программирования C# и средств Windows Forms была разработана программа, которая позволяет управлять запущенными процессами.

Использование данной программы, дает возможность сэкономить много времени на управления процессами на многих серверах одновременно, а так же дает возможность закрывать ненужные процессы автоматически. Программа проста в использовании, так как в ней интуитивно понятный интерфейс, так же была составлена для пользователя инструкция использования программы.

Разработанный код программы был разбит на несколько файловых блоков, по принятому стилю программирования. Создан основной блок выполнения программы, а также дополнительные блоки, которые содержат в себе реализацию описания объектов программы, их методы и их реализацию, также создан дополнительный файл, который хранит в себе идентификаторы ресурсов.

На данном этапе проектирования, функционал программы логически завершен. При продолжении работы над данным проектом, в сторону расширения функционала, возможно добавление различных функций, а именно: взаимодействие с другими операционными системами, поддержка сохранения и загрузки настроек для подключения к компьютерам, отображения дополнительной информации по каждому из процессов, напр. процентов загрузки CPU, количество запущенных потоков внутри процессора и т.д..

В своей структуре программа не имеет сложных алгоритмов, использованные функции легкие в использовании, к тому же, функции стандартной библиотеки Windows подробно описаны во многих источниках. Это позволяет без особых проблем написать или отредактировать код программы человеку с базовыми знаниями языка C#.

СПИСОК ЛИТЕРАТУРЫ

1. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. [Текст]: пер. с англ.- М.: Вильямс, 2013 – 1312 с.
2. Уотсон, К., Нейгел, К., Педерсен, Я. Visual C# 2010: полный курс. [Текст]: Пер. с англ. – М.:ООО «И.Д. ильямс», 2011 – 960 с.
3. Шилдт Г. C# 4.0.: Полное руководство [Текст]: пер. с англ.- М.: Вильямс, 2011 – 1056 с.
4. Попов, А., Шикин, Е. Администрирование Windows с помощью WMI и WMIC [Текст]: 2004 – 752 с.
5. Петцольд, Ч., Программирование с использованием Microsoft Windows Forms [Текст]: 2006 – 433 с.

Приложение А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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;
            }
        }
}
```