

**СОДЕРЖАНИЕ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

ДП2218.02.102.081ПЗ

Разраб.

Володько Н.

Провер.

Захарич В.

Реценз.

Н. Контр.

Усенко И.

Утверд.

Платонова Т.

Программное средство защиты от файловых вирусов

Лит.

Листов

67

НГАЭК, 2021

[ВВЕДЕНИЕ 3](#_Toc65658945)

[ОСНОВНАЯ ЧАСТЬ 5](#_Toc65658946)

[1. Назначение и область применения 5](#_Toc65658947)

[1.1 Характеристика предметной области 5](#_Toc65658948)

[1.2 Назначение программного средства 17](#_Toc65658949)

[1.3 Описание аналогов 17](#_Toc65658950)

[2 Технические характеристики 21](#_Toc65658951)

[2.1 Постановка задачи на разработку программного средства 21](#_Toc65658952)

[2.2 Описание программного средства 21](#_Toc65658953)

[2.2.1 Общие сведения 21](#_Toc65658954)

[2.2.2 Функциональное назначение 22](#_Toc65658955)

[2.2.3 Описание логической структуры 23](#_Toc65658956)

[2.2.4 Используемые технические средства 35](#_Toc65658957)

[2.2.5 Вызов и загрузка 36](#_Toc65658958)

[2.2.6 Входные данные 37](#_Toc65658959)

[2.2.7 Выходные данные 40](#_Toc65658960)

[2.2.8 Рекомендации по использованию 42](#_Toc65658961)

[3 Ожидаемые технико-экономические характеристики 49](#_Toc65658962)

[3.1 Экономический раздел 49](#_Toc65658963)

[3.2 Охрана труда 55](#_Toc65658964)

[3.2.1 Характеристика трудовой деятельности и факторов среды при работе оператора 55](#_Toc65658965)

[3.2.2 Организация общественного контроля условий и безопасности труда 56](#_Toc65658966)

[3.2.3 Общественный контроль за состоянием охраны труда на объекте при проектируемой деятельности 58](#_Toc65658967)

[3.2.4 Ответственность за необеспечение трудоохранных требований 60](#_Toc65658968)

[ЗАКЛЮЧЕНИЕ 66](#_Toc65658969)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 67](#_Toc65658970)

ПРИЛОЖЕНИЯ



# **ВВЕДЕНИЕ**

Все возрастающие усилия по созданию технологий защиты данных их уязвимость в современных условиях не только не уменьшается, но и постоянно возрастает. Актуальность проблем, связанных с защитой информации все более усиливается. Это требует от пользователя персонального компьютера знаний о природе вирусов, способах заражения вирусами и защиты от них. Проблема защиты информации является многоплановой и комплексной и охватывает ряд важных задач. Например, конфиденциальность данных, которая обеспечивается применением различных методов и средств. Интенсивное развитие современных информационных технологий, и в особенности сетевых технологий, создает для этого все предпосылки.

Вирусы, получившие широкое распространение в компьютерной технике, взбудоражили весь мир. Все чаще в средствах массовой информации появляются сообщения о различного рода проделках компьютерных злоумышленников, о появлении все более совершенных, саморазмножающихся программ. Несмотря на принятые во многих странах законы о борьбе с компьютерными преступлениями и разработку специальных программных средств защиты от вирусов, количество новых программных вирусов постоянно растет.

В связи с этим требуется разработка всё более новых и изощрённых методов защиты от вирусов. Существует огромное количество компаний, которые занимаются разработкой самого различного антивирусного программного обеспечения, начиная с простых детекторов вирусов для обычных пользователей, и заканчивая мощными распределенными антивирусными сетями на основе искусственного интеллекта для крупных коммерческих компаний. Но ни одно антивирусное средство не сможет защитить в полную силу, так как каждый день в мире создаётся бесчисленное множество вирусов, 99% из которых обнаруживаются меньше чем за минуту.

Объект исследования – программное средство.

Предмет исследования – программное средство защиты от файловых вирусов.

Цель дипломного проекта – разработать программное средство защиты операционной системы от файловых вирусов.

Задачи на дипломный проект:

* изучить методы защиты операционной системы от файловых вирусов;
* разработать структуру программного средства;
* разработать и протестировать программное средство;
* рассчитать себестоимость и отпускную цену программного средства;
* изучить тему по охране труда «Организация общественного контроля за состоянием охраны труда».

Пояснительная записка к дипломному проекту содержит 22 таблицы, 35 рисунков, 3 диаграммы, 19 источников и 5 приложений.

В введении указана актуальность, объект, предмет исследования, цель и задачи дипломного проекта.

В введении раскрыта актуальность темы, определен объект исследования, предмет исследования, цель и задачи дипломного проекта.

В разделе «Назначение и область применения» раскрыта актуальность темы дипломного проекта, описаны виды вирусов, методы и способы борьбы с ними, проведен анализ аналогов.

В разделе «Проектирование» описана структура программного средства, выбор аппаратных средств, основные алгоритмы, методы разработки программных модулей, реализация взаимосвязи компонентов программного средства, а также рекомендации по использованию и описание графического интерфейса.

В экономическом разделе произведен расчет экономических характеристик.

В разделе «Охрана труда» изучена тема «Организация общественного контроля за состоянием охраны труда», в которой раскрыты такие вопросы как организация общественного контроля условий безопасности труда и общественный контроль за состоянием охраны труда на объекте при проектируемой деятельности.

В заключении подведены итоги разработки дипломного проекта, выделены преимущества, указаны перспективы модернизации и сопровождения разработанного программного средства.

# **ОСНОВНАЯ ЧАСТЬ**

1. Назначение и область применения

1.1 Характеристика предметной области

15 февраля 1946 года, была представлена новейшая и прогрессивная ЭВМ Eniac. Спустя 5 лет после её представления, американец Джон фон Нейман, заложил основы теории самовоспроизводящихся механизмов, он предложил метод создания таких механизмов, эти основы были опубликованы. Спустя некоторое время, учёный Ф. Ж. Шталь, используя материалы статьи Джона фон Неймана, запрограммировал на машинном языке биокибернетическую модель, в которой существа двигались, питаясь ненулевыми словами. Так же, в 1961 году, в фирме Bell Labs изобрели игру, в которой несколько ассемблерных программ, называемых организмами, загружались в память компьютера. Организмы, созданные одним игроком (то есть принадлежащие к одному виду), должны были уничтожать представителей другого вида и захватывать жизненное пространство. Победителем считался тот игрок, чьи организмы захватывали всю память или набирали наибольшее количество очков.

В сентябре 1984 года, был предложен новый термин «вирус». А уже зимой того же года, были опубликованы первые антивирусные утилиты. Они позволяли проанализировать текст загрузочного модуля и выявлять все текстовые сообщения и «подозрительные» участки кода (команды прямой записи на диск и др.). Благодаря своей простоте (фактически использовался только контекстный поиск) и эффективности они получили значительную популярность. Так же эти утилиты перехватывали операции записи и форматирования, выполняемые через [BIOS](https://ru.wikipedia.org/wiki/BIOS). Спустя некоторое время, в начале 1985 года, был создан первый резидентный антивирус, перехватывающий попытки записи на [дискеты](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D0%B5%D1%82%D0%B0) и [винчестер](https://ru.wikipedia.org/wiki/%D0%96%D1%91%D1%81%D1%82%D0%BA%D0%B8%D0%B9_%D0%B4%D0%B8%D1%81%D0%BA), и осуществляющий блокировку всех операций (запись, [форматирование](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), выполняемые через BIOS.

Очередным этапом развития вирусов считается [1987 год](https://ru.wikipedia.org/wiki/1987_%D0%B3%D0%BE%D0%B4). К этому моменту получили широкое распространение сравнительно дешёвые компьютеры IBM PC, что привело к резкому увеличению масштаба заражения компьютерными вирусами. Именно в 1987 вспыхнули сразу три крупные эпидемии компьютерных вирусов. Первая эпидемия была вызвана вирусом Brain. За один год, этот вирус смог поразить более 18 тысяч компьютеров по всему миру. Вирус перезагружал компьютеры, на которых использовалось не лицензионное программное обеспечение. Он так же являлся первым невидимым вирусом, при попытке чтения заражённого сектора, он подставлял его незаражённый оригинал.

Вторая вирусная эпидемия была связанна с развитием глобальной сети интернет. В 1988 году, Робертом Моррисом был создан первый сетевой червь. Занимал червь всего 60 килобайт. Проникая в компьютер жертвы, червь подключал компоненты, позволяющие раскрывать пароли, имеющиеся в системе, что в свою очередь, позволяло программе маскироваться под задачу легальных пользователей системы, на самом деле занимаясь размножением и рассылкой копий. Он поразил свыше 6200 компьютеров. В результате вирусной атаки большинство сетей вышло из строя на срок до пяти суток. Компьютеры, выполнявшие коммутационные функции, работавшие в качестве файл-серверов или выполнявшие другие функции обеспечения работы сети, также вышли из строя.

Третья вирусная эпидемия была наиболее крупная. Появился вирус Datacrime, спящий вирус. Вирус распространялся и никак не выдавал себя, но начиная с 12 октября производил разрушение файловой системы на заражённых компьютерах. Всего было заражено более 100 тысяч ПЭВМ.

Начиная с 2010 года, проблема вирусов начинает принимать глобальный масштаб. 17 июня 2010 года, был обнаружен ранее не известный вирус Stuxnet. Вся проблема заключалась в том, что он был обнаружен не только на персональных компьютерах пользователей, но и в промышленных системах, управляющих автоматизированными производственными процессами. Это первый в истории вирус, который перехватывал и модифицировал информационный поток между программируемыми логическими контроллерами и рабочими станциями. Уникальность заключалась в том, что вирус мог физически разрушать инфраструктуру. Поэтому он мог быть использован как средство несанкционированного сбора данных и диверсий. Размер его, составлял всего 500 КБ.

Существует огромное количество различных вирусов. В таблице 1.1.1 приведена общая классификация типов вирусов.

Таблица 1.1.1 ­­– Общая классификация типов вирусов

|  |  |
| --- | --- |
| Наименование | Описание |
| 1 | 2 |
| Анти-антивирусный вирус | Компьютерная вирусная программа, объектом нападения которой являются антивирусные программы |
| Вариант вируса, штамм, модификация | Модифицированный вариант одного и того же вируса |
| Вирусная программа-червь | Паразитическая программа, обладающая механизмом саморазмножения. Программа способна размножать свои копии, но не поражать другие компьютерные программы. Проникает на компьютер из сети и рассылает свои функциональные копии на другие компьютерные сети |

Продолжение таблицы 1.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| Вирусный мистификатор | Не являющееся вирусом почтовое сообщение. На компьютер пользователя мистификация приходит в виде письма, написанного в подчеркнуто нейтральном тоне, в котором указывается на якобы распространяющийся новый вирус. Пользователю предлагается найти некий файл с помощью поискового средства Windows и удалить его с диска |
| Вирусы-спутники | Формально являются файловыми вирусами. Не внедряются в исполняемые программы. Такие вирусы используют особенность системы, позволяющую программному файлу с тем же названием, но другим расширением действовать с разными приоритетами. Такие вирусы могут быть резидентными и маскировать файлы-двойники |
| Дроппер | Файл-носитель, устанавливающий вирус в систему. Техника, иногда используемая вирусописателями для сокрытия вирусов от антивирусных программ |
| Зоологический вирус | Вирусы, которые существуют только в антивирусных лабораториях, в коллекциях исследователей вирусов |
| Полиморфные вирусы | Вирусы с самомодифицирующимися расшифровщиками, использующие помимо шифрования кода специальную процедуру расшифровки, изменяющую саму себя в каждом новом экземпляре вируса, что ведет к отсутствию у него байтовых сигнатур |
| MtE вирусы | Полиморфные вирусы, созданные с помощью генератора полиморфизма |

Окончание таблицы 1.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| Резидентный вирус | Постоянно присутствующий в памяти вирус, написанный, как правило на низкоуровневом языке. Такие вирусы обладают возможностью эффективно заражать программы и противодействовать антивирусным средствам. |
| Скрипт-вирус | Вирус, написанный на языках Visual Basic, Basic Script, Java Script. На компьютер пользователя чаще всего проникают в виде почтовых сообщений, содержащих во вложениях файлы-сценарии. Программы на языках Visual Basic и Java Script могут располагаться как в отдельных файлах, так и встраиваться с HTML документ и в таком случае интерпретироваться с браузером, причем не только с удаленного сервера, но и с локального диска |
| Стелс-вирус | Вирусные программы, предпринимающие специальные действия для маскировки своей деятельности с целью сокрытия своего присутствия в зараженных объектах. Делятся на 2 типа: руткит и буткит |
| Шифрованные вирусы | Вирусы, которые сами шифруют свой код для затруднения их дизассемблирования и обнаружения в файле, памяти или секторе. Каждый экземпляр такого вируса будет содержать только короткий общий фрагмент – процедуру расшифровки – который можно выбрать в качестве сигнатуры. В случае каждого инфицирования он автоматически зашифровывает себя, и каждый раз по-разному. Таким способом вирус пытается избежать обнаружения вирусными программами. |

В таблице 1.1.2 приведена классификация именований вирусов

Таблица 1.1.2 – Классификация наименований вирусов

|  |  |
| --- | --- |
| Наименование | Описание |
| 1 | 2 |
| High Level Language вирусы | В данную группу объединяют вирусы, написанные на языках программирования высокого уровня, таких как C#, Java, Pascal |
| Троянские программы | В данную группу объединяют вирусы, которые осуществляют несанкционированные пользователем действия на его компьютере. |
| Silly-вирусы | Вирусы, которые не обладают никакими особенными характеристиками, вследствие чего нет необходимости присвоить таким вирусам особенные названия |
| Макровирусы | В данную группу объединяют вирусы, использующие особенности файлов офисных программ, а так же встроенные макроязыки данных приложений. |
| Скрипт-вирусы | В данную группу объединяют вирусы, написанные на различных интерпретируемых языках. Разделяют на 9 классов: VBS, JS, Wscript, HTML, Perl, PHP, IRC, Java, Bat. |
| Узкоспециализированное ПО | В данную группу объединяют вирусы, разработанные для инфицирования файлов конкретных программ. Выделяют 3 класса: ACAD, AutoLisp, SWF |
| Суффиксы | В эту группу входит 4 класса: вирусный конструктор, инсталлятор вируса. |
| Для DoS-атак | В данную группу объединяют вредоносные программы, разработанные для осуществления DoS атак с целью довести вычислительную технику до отказа в обслуживании |

Окончание таблицы 1.1.2

|  |  |
| --- | --- |
| 1 | 2 |
| Потенциально опасное ПО | В данную группу объединяют программы, неспособные к самовоспроизведению и представляющие потенциальную опасность. В группу входит 4 классов: программы-шутки, хакерские утилиты, рекламные вирусы, программы, которые могут быть использованы злоумышленниками |
| Вирусы для разных ОС | В данную группу объединяют вредоносные программы, разработанные для определения ОС. Выделяют 19 классов. |
| Опасное ПО | Программы данной группы характеризируют как опасные и разделяют на 3 класса: Backdoor, Trojan, Exploit |
| Инструменты создания вредоносного ПО | В данную группу объединяют программы, которые используются для генерации или создания вирусов (вирусные конструкторы). |

В таблице 1.1.3 приведена классификация вирусов по видам заражаемых объектов.

Таблица 1.1.3 – Классификация вирусов по видами заражаемых объектов.

|  |  |
| --- | --- |
| Наименование | Описание |
| 1 | 2 |
| Файловые вирусы | Вирусы, заражающие двоичные файлы (в основном исполняемые файлы и динамические библиотеки). Чаще всего, такие файлы имеют расширение EXE, COM, DLL, SYS. Так же могут инфицировать файлы с расширениями DRV, BIN, OVL, OVY. Такие вирусы внедряются в файлы операционной системы, активируются при запуске пораженной программы и затем распространяются |

Окончание таблицы 1.1.3

|  |  |
| --- | --- |
| 1 | 2 |
| Загрузочные (бутовые) вирусы | Вирусы, которые заражают загрузочные записи дискет, разделов жестких дисков, а также MBR жестких дисков |
| Макрокомандные вирусы (макровирусы) | Вирусы, заражающие файлы документов, используемые приложениями Microsoft Office и другими программами, допускающие наличие макрокоманд. Благоприятным фактором распространения вируса служит то, что все основные компоненты Microsoft Office могут содержать встроенные программы на полнофункциональном языке программирования, а в Microsoft Word эти макросы автоматически запускаются при открытии любого документа, его закрытии, сохранении и т.д. Учитывая то, что копирование макросов из документа в документ (в частности в общий шаблон) выполняется всего одной командой |

В зависимости от среды обитания вирусы можно разделить на:

– сетевые;

– файловые;

– загрузочные;

– файлово-загрузочные.

Сетевые вирусы распространяются по различным компьютерным сетям.

Файловые вирусы внедряются главным образом в исполняемые модули, а именно в файлы, имеющие расширения com и exe. Загрузочные вирусы внедряются в загрузочный сектор диска (Boot-сектор) или в сектор, содержащий программу загрузки системного диска (Master Boot Record). Файлово-загрузочные вирусы заражают как файлы пользователя, так и загрузочные сектора подключенных жестких дисков.

По способу заражения вирусы делятся на:

– резидентные;

– нерезидентные.

Резидентный вирус при заражении (инфицировании) компьютера оставляет в оперативной памяти свою резидентную часть, которая потом перехватывает обращение операционной системы к объектам заражения (файлам, загрузочным секторам дисков) и внедряется в них. Резидентные вирусы находятся в памяти и являются активными вплоть до выключения или перезагрузки компьютера.

Нерезидентные вирусы не заражают память компьютера и являются активными ограниченное время.

По степени воздействия вирусы можно разделить на следующие виды:

– неопасные, не мешающие работе компьютера, но уменьшающие объем свободной оперативной памяти и памяти на дисках, действия таких вирусов проявляются в каких-либо графических или звуковых эффектах;

– опасные вирусы, которые могут привести к различным нарушениям в работе компьютера;

– очень опасные, воздействие которых может привести к потере программ, уничтожению данных, стиранию информации в системных областях диска.

Заражению часто подлежат файлы программ из пакета Microsoft office. В эти файлы при заражении всегда вставляются макросы, которые создаются на языке Visual Basic, который поддерживается в подавляющем большинстве продуктов Microsoft office.

По особенностям алгоритма вирусы трудно классифицировать из-за большого разнообразия. Простейшие вирусы – паразитические, они изменяют содержимое файлов и секторов диска и могут быть достаточно легко обнаружены и уничтожены. Вирусы-репликаторы, называемые червями, которые распространяются по компьютерным сетям, вычисляют адреса сетевых компьютеров и записывают по этим адресам свои копии. Вирусы-невидимки, называемые стелс-вирусами, которые очень трудно обнаружить и обезвредить, так как они перехватывают обращения операционной системы к пораженным файлам и секторам дисков и подставляют вместо своего тела незараженные участки диска. Наиболее трудно обнаружить вирусы-мутанты (полиморфные вирусы), содержащие алгоритмы шифровки-расшифровки, благодаря которым копии одного и того же вируса не имеют ни одной повторяющейся цепочки байтов. Имеются и так называемые квазивирусные или «троянские» программы, которые хотя и не способны к самораспространению, но очень опасны, так как, маскируясь под полезную программу, могут разрушить загрузочный сектор и файловую систему дисков.

Так же следует не забывать про атаки с использованием внешних аппаратных устройств. Примером такой атаки является атака BadUSB, основанная на уязвимости USB устройств. Она устроена следующим образом: для определённого контроллера съёмного накопителя создаётся специальная прошивка, которая содержит код вируса, и уязвимый контроллер (съёмный накопитель) прошивается этой прошивкой. Модифицированная прошивка может исполнять любые команды злоумышленника, при этом оставаясь незамеченным в системе, так как все действия идут от эмулированного устройства ввода-вывода, например, сетевой карты, клавиатуры, мыши, или к примеру загрузочным устройством. Атака действует на все устройства, в которых есть USB порт, вне зависимости от установленной операционной системы. Единственное условие – должен быть установлен драйвер поддержки USB устройств.

Вирусы способны заражать такие объекты, как:

– исполняемые файлы;

– загрузчик системы;

– файлы документов.

Вирусы, которые заражают файлы, называются файловыми. Вирус в зараженных исполняемых файлах начинает свою работу при запуске той программы, в которой он находится.

Вирусы, которые заражают загрузчик операционной системы и главную загрузочную запись, называются загрузочными. Обычно загрузочные вирусы разделены на 2 части: первая часть находится в главной загрузочной записи, вторая находится в другом участке диска, или в кластере в области данных диска. Разделение на 2 части обусловлено тем, что в главную загрузочную запись невозможно записать что-либо большое.

С ростом угрозы, появлялось всё больше программ, решений и методов, для борьбы с ними, самыми актуальными методами обнаружения угроз являются:

– сигнатурный анализ;

– эвристический анализ;

– метод эмуляции исполнения программного кода.

При проверке компьютера на предмет угроз, в первую очередь применяется сигнатурный анализ. Он выполняется путем проверки содержимого анализируемого объекта на предмет наличия в нем сигнатур уже известных вирусов. Сигнатурой называется непрерывная конечная последовательность байт, необходимая и достаточная для однозначной идентификации угрозы. При этом сравнение содержимого исследуемого объекта с сигнатурами производится не напрямую, а по их контрольным суммам, что позволяет значительно снизить размер записей в вирусных базах, сохранив при этом однозначность соответствия и, следовательно, корректность обнаружения угроз и лечения инфицированных объектов. Записи в вирусных базах составляются таким образом, что благодаря одной и той же записи можно обнаруживать целые классы или семейства угроз.

Работа эвристического анализатора основывается на наборе эвристик (предположений, статистическая значимость которых подтверждена опытным путем) о характерных признаках вредоносного и, наоборот, безопасного исполняемого кода. Каждый признак кода имеет определенный вес (т. е. число, показывающее важность и достоверность этого признака). Вес может быть как положительным, если признак указывает на наличие вредоносного поведения кода, так и отрицательным, если признак не свойственен компьютерным угрозам. На основании суммарного веса, характеризующего содержимое объекта, эвристический анализатор вычисляет вероятность содержания в нем неизвестного вредоносного объекта. Если эта вероятность превышает некоторое пороговое значение, то выдается заключение о том, что анализируемый объект является вредоносным.

Эвристический анализатор также использует технологию Fly-Code – универсальный алгоритм распаковки файлов. Этот механизм позволяет строить эвристические предположения о наличии вредоносных объектов в объектах, сжатых программами упаковки (упаковщиками), причем не только известными разработчикам продукта, но и новыми, ранее не исследованными программами. При проверке упакованных объектов также используется технология анализа их структурной энтропии, которая позволяет обнаруживать угрозы по особенностям расположения участков их кода. Эта технология позволяет на основе одной записи вирусной базы произвести обнаружение набора различных угроз, упакованных одинаковым полиморфным упаковщиком.

Поскольку эвристический анализатор является системой проверки гипотез в условиях неопределенности, то он может допускать ошибки как первого (пропуск неизвестных угроз), так и второго рода (признание безопасной программы вредоносной). Поэтому объектам, отмеченным эвристическим анализатором как «вредоносные», присваивается статус «подозрительные».

Метод эмуляции исполнения программного кода используется для обнаружения полиморфных и шифрованных вирусов, когда использование поиска по контрольным суммам сигнатур неприменимо или значительно усложнено из-за невозможности построения надежных сигнатур. Метод состоит в имитации исполнения анализируемого кода при помощи эмулятора – программной модели процессора и среды исполнения программ. Эмулятор оперирует с защищенной областью памяти (буфером эмуляции). При этом инструкции не передаются на центральный процессор для реального исполнения. Если код, обрабатываемый эмулятором, инфицирован, то результатом его эмуляции станет восстановление исходного вредоносного кода, доступного для сигнатурного анализа.

Сканирование является наиболее традиционным методом поиска вирусов. Оно заключается в поиске уже ранее обнаруженных вирусов. Антивирусные программы-сканеры, способные удалить обнаруженные вирусы, обычно называют полифагами. Недостаток метода сканирования в том, что невозможно обнаружить полиморфные вирусы, полностью меняющие свой код. Для этого нужно использовать более сложные алгоритмы поиска, включающие эвристический анализ. Сканнеры не способны защитить компьютер от новых вирусов, поэтому они не эффективны.

Наиболее эффективным методом обнаружения вирусов является эвристический анализ. Эвристический анализ зачастую используется совместно с сигнатурным анализом для поиска шифрующихся и полиморфных вирусов. В большинстве случаев эвристический анализ позволяет также обнаруживать и ранее неизвестные вирусы. В этом случае, скорее всего их лечение будет невозможно.

Самым ресурсо-затратным методом, является метод обнаружения изменений файлов. Заражая компьютер, вирус делает изменения на жестком диске, на обнаружении таких изменений и основывается этот метод. В системе может существовать одновременно очень большое количество процессов, которые так или иначе вносят изменения в файловую систему компьютера. Антивирусной программе приходится проверять и фильтровать каждую такую операцию ввода-вывода.

По статистике компании Dr. Web только 30% обнаруженных угроз приходится на сигнатурный анализ, остальные 70% приходятся на метод эвристического анализа и эмуляции исполнения.

Так же, с развитием ЭВМ и появлением нового оборудования, появились аппаратные уязвимости, которые полностью можно было устранить, только заменой оборудования на более новое, зачастую более дорогостоящее.

Примером аппаратной уязвимости является уязвимость Meltdown, позволяющая любой вирусной программе, путём выполнения определённых инструкций извлечь любые данные из кэша процессора. Атака возможна благодаря манипуляции с внеочередным исполнением внутри процессора. Технология внеочередного исполнения команд процессором создана для того, чтобы определить наиболее приоритетные инструкции в конвейере обработки инструкций и выполнить их в первую очередь. Однако этим процессом можно манипулировать таким образом, что можно извлечь любую информацию, начиная с обычной строки, заканчивая ключами шифрования и пользовательскими данными. Уязвимости подвержены все процессоры, которые выпускаются с 1995 года.

Способы противодействия компьютерным вирусам можно разделить на несколько групп:

– профилактика заражения и уменьшение предполагаемого ущерба от такого заражения;

– методика использования антивирусных программ, в том числе обезвреживание и удаление известного вируса;

– способы обнаружения и удаления неизвестного вируса.

Наиболее эффективны в борьбе с компьютерными вирусами антивирусные программные средства. Однако не существует антивирусных средств, гарантирующих полную защиту от вирусов, и заявления о существовании таких систем можно расценить как-либо недобросовестную рекламу, либо непрофессионализм. Таких систем не существует, поскольку на любой алгоритм антивируса всегда можно предложить контр-алгоритм вируса, невидимого для этого антивируса.

Основными признаками заражения вирусом являются:

– замедление работы компьютера;

– невозможность загрузки операционной системы;

– частые зависания и сбои в работе компьютера;

– прекращение работы или неправильная работы ранее функционировавших программ;

– увеличение количества файлов на диске;

– изменение размера файлов;

– периодическое появление на экране монитора неуместных системных сообщений;

– уменьшение объема свободной оперативной памяти;

– заметное возрастание времени доступа к файлам на жестком диске;

– изменение даты и времени создания файлов;

– разрушение файловой структуры (исчезновение файлов, искажение каталогов и др.).

Следует также обратить внимание на несколько терминов, применяемых при обсуждении антивирусных программ:

– ложное срабатывание – детектирование вируса в незараженном объекте (файле, секторе или системной памяти);

– сканирование по запросу – поиск вирусов по запросу пользователя. В этом режиме антивирусная программа неактивна до тех пор, пока не будет вызвана пользователем из командной строки, командного файла или программы-расписания;

– сканирование на лету – постоянная проверка на вирусы объектов, к которым происходит обращение (запуск, открытие, создание). В этом режиме антивирус постоянно активен, он присутствует в памяти «резидентно» и проверяет объекты без запроса пользователя.

Далее приведены основные понятия и ключевые слова, необходимые для понимания предметной области.

Компьютерный вирус – это вид вредоносного программного обеспечения, способного создавать копии самого себя и внедряться в код других программ, системные области памяти, загрузочные секторы, а также распространять свои копии по разнообразным каналам связи

**Антивирусная программа (антивирус)** – изначально компьютерная программа, которая предназначена для обезвреживания вирусов и различного рода вредоносного ПО, с целью сохранности данных и оптимальной работы вашего персонального компьютера.

Сигнатура – непрерывная конечная последовательность байт, необходимая и достаточная для однозначной идентификации угрозы.

Шины устройств – подсистемы передачи данных между функциональными блоками компьютера (например, шина USB, PCI).

Хеш-сумма – уникальный идентификатор файла, представляющий собой последовательность цифр и букв заданной длины. Используется для проверки целостности данных.

Эксплойт – программа, фрагмент кода или последовательность команд, использующие уязвимости в программном обеспечении и применяемые для проведения атаки на систему.

Ядро программного средства – совокупность компонентов и связей между ними.

IRP пакет – пакет запроса ввода-вывода.

1.2 Назначение программного средства

Основное назначение антивирусных программных средств – обеспечение базовой защиты, предотвращение угроз, восстановление системы и данных, защита конфиденциальных данных.

В состав базовой защиты в первую очередь входят функции:

– защиты от вирусов, червей, эксплойтов, троянских, шпионских и рекламных программ;

– проверка файлов, почтовых сообщений, интернет в автоматическом режиме и по требованию;

– мониторинг активности (сбор данных о работе программ на компьютере);

– откат действий вредоносной программы;

– постоянная защита от фишинговых сайтов;

– постоянная проверка файлов в автономном режиме.

В состав функции предотвращения угроз входит:

– поиск уязвимостей в операционной системе и установленном ПО;

– блокирование ссылок на заражённые сайты;

– распознавание вирусов по способу их упаковки;

В состав функции восстановления системы и данных входит:

– возможность упаковки программы на заражённый компьютер;

– функция самозащиты программы от выключения или остановки;

– восстановление корректных настроек системы после удаления вредоносного ПО;

– наличие инструментов для создания диска аварийного восстановления.

В состав функции защиты конфиденциальных данных входит:

– блокирование ссылок на фишинговые сайты;

– защита от всех видов кейлоггеров и шпионских программ.

1.3 Описание аналогов

Основными критериями оценки антивирусных программных средств являются:

– стабильность и надежность работы – самый определяющий параметр, даже самый лучший антивирус окажется совершенно бесполезным, если он не сможет нормально функционировать на вашем компьютере, если в результате какого-либо сбоя в работе программы процесс проверки компьютера не пройдёт до конца;

– размер вирусной базы программы, сюда же следует отнести и возможность программы определять разнообразные типы вирусов, и умение работать с файлами различных типов (архивы, документы);

– скорость работы алгоритмов программы, к этому пункту относятся такая характеристика как скорость сканирования или проверки файла;

– кроссплатформенность.

Существует довольно много существующих разработок. Они делятся на платные и бесплатные, но самыми популярными из них, по количеству установок являются: NOD32, Kaspersky, Norton, Doctor Web, McAfee, Avira.

Самыми популярными антивирусными средствами в русскоговорящем сегменте являются Dr Web (доктор веб) и Kaspersky (антивирус Касперского).

В таблице 1.3.1 приведено сравнение антивирусных продуктов.

Таблица 1.3.1 – Сравнение антивирусных продуктов по различным параметрам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название антивирусной программы | Kaspersky | NOD 32 | Symantec | Dr. Web | 360 Total security |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Поддержка в системах семейства Windows | Имеется | Имеется | Имеется | Имеется | Имеется |
| Поддержка в системах семейства Linux | Имеется | Не имеется | Не имеется | Имеется | Не имеется |
| Время загрузки системы | До 4 минут | До 3 минут | До 3 минут | До 4 минут | До 2 минут |
| Время сканирования системных папок | Меньше 20 минут | Меньше 10 минут | Меньше 15 минут | Меньше 17 минут | Меньше 10 минут |
| Загрузка ЦП | 5.5 – 8 % | 8%–12% | 4% –10% | 3% – 8% | 2.5% – 5% |
| Использование ОЗУ в простое | 210 мб | 138 мб | 157 мб | 120 мб | 107 мб |
| Антивирусный сканер и монитор | Имеется | Имеется | Имеется | Имеется | Имеется |
| Защита персональных данных | Имеется | Имеется | Имеется  (в платной версии) | Имеется | Имеется |
| Защита почтового ящика | Имеется | Имеется | Имеется | Имеется | Не имеется |
| Работа в облаке | Не имеется | Не имеется | Имеется | Не имеется | Имеется |

Окончание таблицы 1.3.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Обнаружение вторжения | Имеется | Имеется | Имеется | Имеется | Не имеется |
| Система обновлений | Имеется | Имеется | Имеется | Имеется | Имеется |
| Веб-защита | Имеется | Имеется | Не имеется | Имеется | Имеется |
| Антиспам | Имеется | Имеется | Имеется | Имеется | Не имеется |
| Поддержка большинства языков | Имеется | Имеется | Имеется | Имеется | Имеется |

Интерфейсы всех антивирусных средств ориентированы на обычного пользователя, поэтому являются интуитивно понятными.

На рисунке 1.3.1 изображен интерфейс антивирусного программного средства Dr Web.

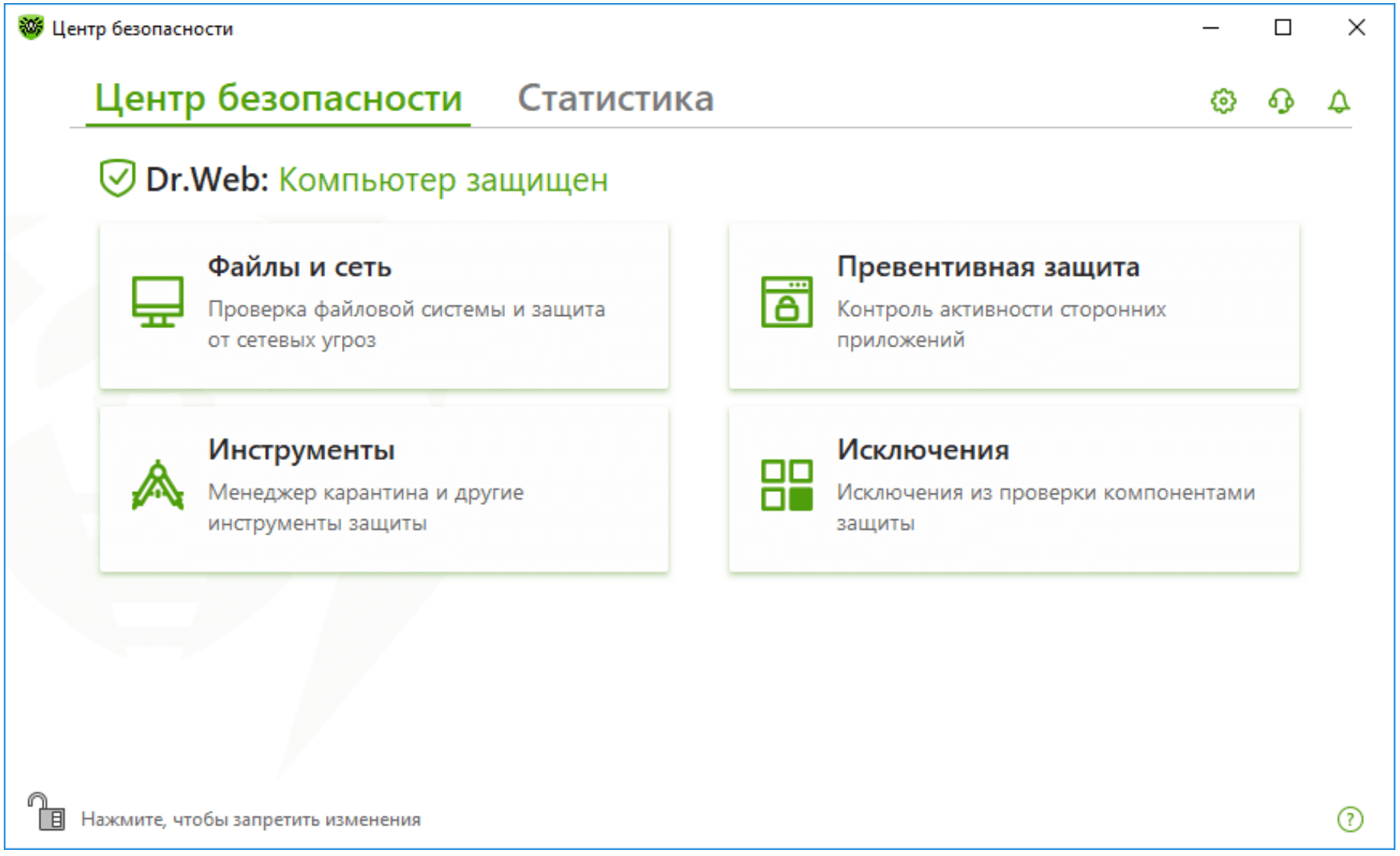


Рисунок 1.3.1 – Интерфейс Dr Web

На рисунке 1.3.2 изображен интерфейс антивирусного программного средства от лаборатории Касперского.

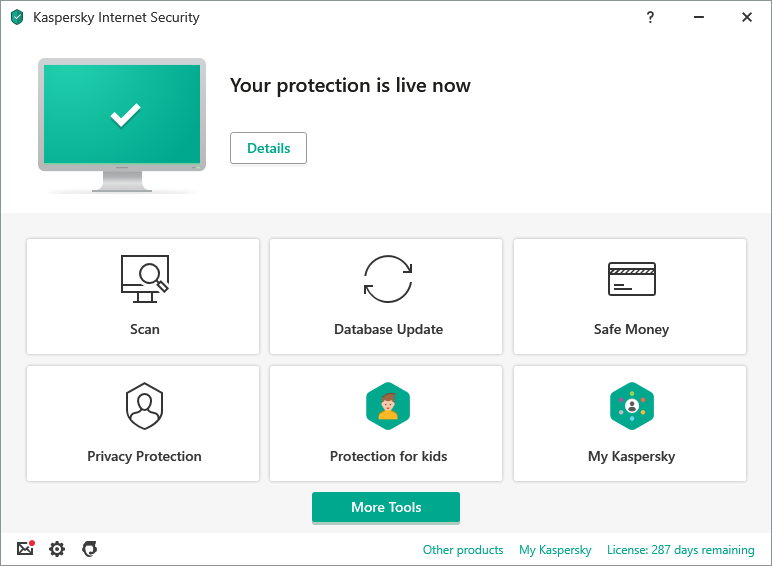


Рисунок 1.3.2 – Антивирус Касперского

У большинства антивирусных средств графический интерфейс очень простой по причине того, что вмешивание пользователя в работу практически не требуется, за исключением простой настройки, всё производится автоматически.

На основании проведенного сравнения, можно сделать вывод что одним из лучших антивирусных средств, является антивирус Касперского. Именно он обеспечивает наилучшую защиту от взломщиков, троянов и других видов вирусных программ.

2 Технические характеристики

2.1 Постановка задачи на разработку программного средства

Целью дипломного проекта является разработка программного средства, позволяющего защитить операционную систему от файловых вирусов.

Для достижения поставленной цели необходимо выполнить задачи:

– разработать логическую структуру программного средства;

– разработать и протестировать библиотеки, модули и драйвер операционной системы.

Программное средство должно обеспечивать следующий функционал:

– защита операционной системы от файловых вирусов;

– хранение файловых вирусов в защищенном хранилище (карантин);

– ручное сканирование;

– настройка исключений;

– защита важных файлов с помощью криптографа;

– своевременное автоматическое обновление базы сигнатур;

– автоматическая проверку файлов в фоновом режиме.

2.2 Описание программного средства

2.2.1 Общие сведения

Программное средство защиты от вирусных файлов разработано на языке C# с использованием платформы .NET Framework.

.NET Framework – [программная платформа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0#%D0%BA%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), выпущенная компанией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в [2002 году](https://ru.wikipedia.org/wiki/2002_%D0%B3%D0%BE%D0%B4). Основой платформы является общеязыковая среда исполнения [Common Language Runtime (CLR)](https://ru.wikipedia.org/wiki/Common_Language_Runtime), которая подходит для различных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Плюсы использования платформы .NET:

– управление памятью, во многих языках разработчики должны предусматривать выделение и освобождение памяти, а также управлять временем жизни объектов, в приложениях .NET Framework эти функции выполняет среда CLR;

– общая система типов, в традиционных языках программирования базовые типы определяются компилятором, что осложняет взаимодействие между языками, в .NET Framework базовые типы определяются системой типов;

– обширная библиотека классов, разработчикам не требуется писать код для выполнения стандартных низкоуровневых операций программирования, так как они используют удобную библиотеку типов и членов, входящую в библиотеку классов .NET Framework;

– платформы и технологии разработки, платформа .NET Framework включает библиотеки для конкретных областей разработки приложений;

– взаимодействие языков, языковые компиляторы, ориентированные на .NET Framework, выдают промежуточный код, называемый языком CIL (Common Intermediate Language), который, в свою очередь, компилируется во время выполнения средой CLR;

– совместимость версий, приложения, разработанные на основе конкретной версии платформы .NET Framework, могут выполняться без доработок и на более поздних версиях платформы;

– параллельное выполнение, платформа .NET Framework помогает разрешать конфликты версий, поскольку на компьютере могут быть установлены несколько версий среды CLR.

Программное средство имеет модульную структуру. Все модули изолированы от ядра и не могут ему навредить каким-либо образом. Преимущества такого подхода в том, что для обновления программного средства не нужно его полностью выключать, что могло бы ослабить защиту во время обновления. Для обновления достаточно выключить нужный модуль и заменить его библиотеку на жестком диске.

2.2.2 Функциональное назначение

Программное средство позволяет защитить операционную систему от файловых вирусов, которые могут попасть на компьютер пользователя через различные источники, такие как съемные носители, дискеты, флеш-накопители, электронная почта, веб-страницы, интернет и локальные сети, а также через системы обмена мгновенными сообщениями.

Программное средство предоставляет пользователю следующий функционал:

– обнаружение и обезвреживание угроз в пассивном режиме;

– принудительное сканирование файлов и папок;

– хранение зараженных файлов в карантине;

– настройка исключений;

– защита файлов и папок шифрованием.

2.2.3 Описание логической структуры

На рисунке 2.2.3.1 изображена логическая структура программного средства. В приложении В изображена диаграмма компонентов программного средства.

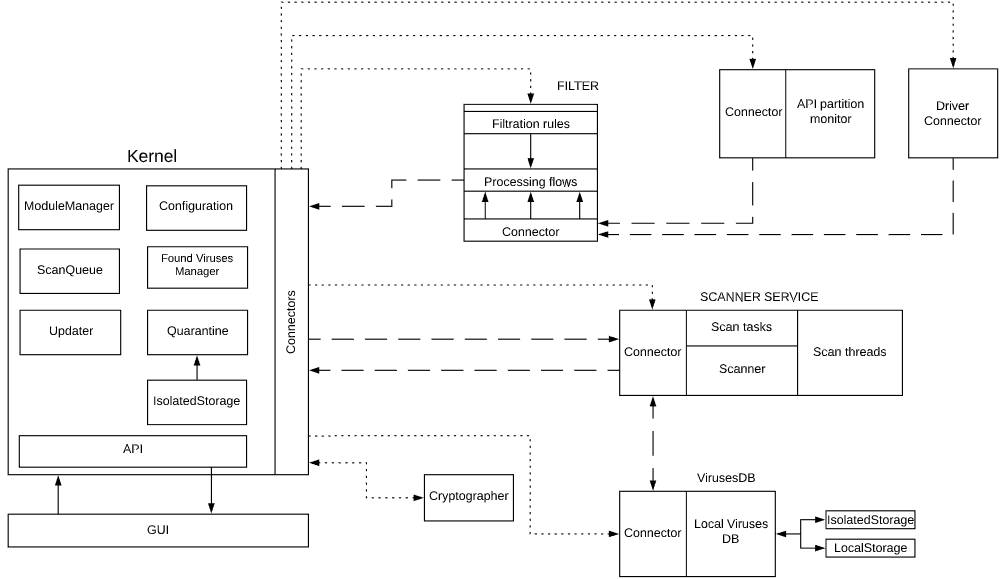


Рисунок 2.2.3.1 – Логическая структура программного средства

В таблице 2.2.3.1 приведено описание всех модулей программного средства.

Таблица 2.2.3.1 – описание компонентов и модулей программного средства

|  |  |
| --- | --- |
| Компонент | Описание |
| 1 | 2 |
| Kernel | Ядро программного средства, связывает воедино все модули, состоит из 8 компонентов |
| Filter | Модуль фильтрации, позволяет отфильтровать непригодные для сканирования файлы и пути, состоит из 3 компонентов |
| Api partition monitor | Модуль обнаружения создания или изменения файлов на жестком диске |
| Scanner service | Модуль сканирования файлов, состоит из 4 компонентов |
| Viruses DB | Модуль локальной базы сигнатур |
| Cryptographer | Модуль криптографа |

Окончание таблицы 2.2.3.1

|  |  |
| --- | --- |
| 1 | 2 |
| GUI | Модуль графического интерфейса |
| Driver connector | Модуль коннектора, позволяет соединить ядро программного средства с драйвером ядра операционной системы |

В таблице 2.2.3.2 приведено описание внутренних компонентов ядра программного средства

Таблица 2.2.3.2 – Компоненты ядра программного средства

|  |  |
| --- | --- |
| Компонент | Описание |
| ModuleManager | Менеджер модулей, предназначен для загрузки и управления работой сторонних DLL модулей |
| ScanQueue | Менеджер задач сканирования, хранит в себе активные задачи по сканированию файлов, компонент взаимодействует с сервисом сканирования |
| Updater | Модуль обновления, позволяет получать обновления через интернет |
| Configuration | Компонент конфигурации, содержит в себе настройки позволяющие задавать работу модулей и компонентов |
| Found Viruses Manager | Менеджер обнаруженных вирусов, хранит в себе полную информацию об обнаруженных вирусах |
| Quarantine | Компонент карантина, позволяет хранить обнаруженные вирусы в защищенном хранилище |
| Connectors | Компонент соединений, позволяет другим компонентам ядра взаимодействовать с DLL модулями |
| API | Компонент позволяющий связать ядро со сторонним пользовательским приложением |

Все DLL модули имеют одну схожую черту, все они имеют в своём составе один и тот же компонент, называемый коннектором. Коннектор позволяет связать модуль с другим модулем или приложением с помощью именованных каналов. Именованный канал представляет собой поток байтов, доступ к которому может иметь несколько приложений, данные в такой поток можно записывать и считывать.

При запуске ядра первоначально производится инициализация конфигурации, затем следует загрузка DLL модулей, инициализация входящих и исходящих подключений и только после этого производится инициализация внутренних компонентов ядра. Такой порядок обусловлен тем, что компоненты ядра работают с DLL модулями, чтобы удостоверится в работоспособности модулей, необходимо создать подключение к ним, если подключение успешно, то производится инициализация внутренних компонентов. Процесс полной инициализации изображен на блок-схеме 2.2.3.2.

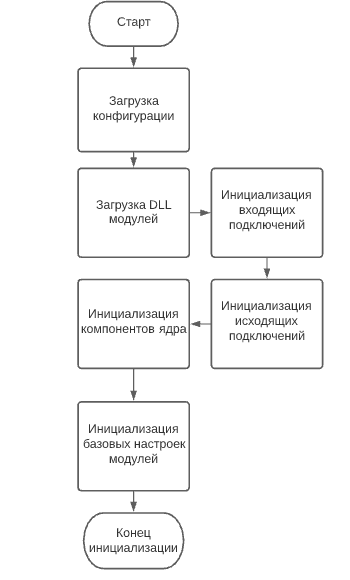


Рисунок 2.2.3.2 – Блок-схема процесса полной инициализации ядра

Менеджер модулей хранит описание и различные данные о DLL модуле. В таблице 2.2.3.3 описаны данные, которые хранит менеджер модулей о каждом модуле.

Таблица 2.2.3.3 – Описание хранимых данных о DLL модуле

|  |  |
| --- | --- |
| Наименование поля | Описание |
| ModuleName | Наименование модуля |
| ModuleAssembly | Ссылка на сборку модуля |
| IsRunning | Состояние модуля |

На рисунке 2.2.3.3 изображена блок-схема процесса загрузки DLL модуля.

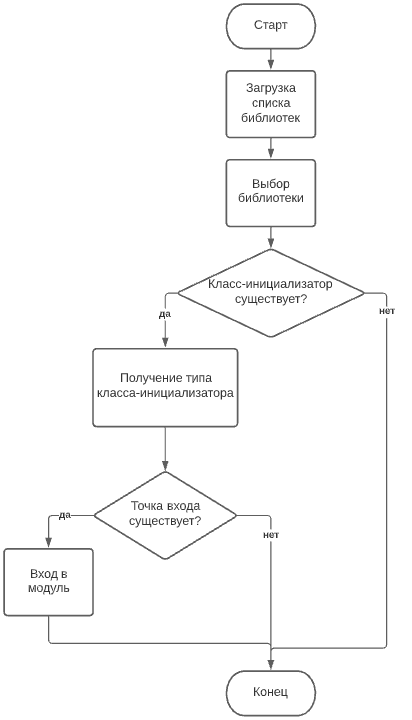


Рисунок 2.2.3.3 – Блок-схема процесса загрузки DLL модуля

Все DLL модули всегда инициализируют подключения к другим модулям. В таблице 2.2.3.4 описаны действия, выполняемые при инициализации каждого модуля.

Таблица 2.2.3.4 – Действия выполняемые при инициализации каждого модуля

|  |  |
| --- | --- |
| Наименование инициализируемого модуля | Описание действий |
| API Partition monitor | Запуск потока выполнения команд |
| Filter | Загрузка стандартных правил фильтрации, запуск потоков-обработчиков |
| Scanner service | Запуск потоков-обработчиков |
| Viruses DB | Запуск потока обработки команд, загрузка файлов базы сигнатур в защищенное хранилище, загрузка сигнатур в ОЗУ из защищенного хранилища |

В таблице 2.2.3.5 описаны действия, выполняемые при инициализации каждого компонента ядра.

Таблица 2.2.3.5 – Действия выполняемые при инициализации каждого компонента ядра

|  |  |
| --- | --- |
| Наименование инициализируемого компонента | Описание действий |
| API | Установка событий, запуск потока-обработчика запросов |
| ScannerResponseHandler | Запуск потока-обработчика |
| ScanTasks | Запуск потока-обработчика, установка событий |
| FoundVirusesManager | Запуск потока-обработчика |
| Quarantine | Установка защищенного хранилища, проверка существования директорий |

После полной инициализации производится базовая настройка подключенных модулей, а именно:

– установка проверяемых разделов в мониторе разделов;

– установка правил фильтрации;

– генерация или загрузка ключей шифрования;

– выгрузка всех сигнатур из модуля базы сигнатур в модуль сканнера.

Для мониторинга файловой системы в операционной системе используется класс FileSystemWatcher, который в свою очередь взаимодействует с Windows API (application programming interface). На рисунке 2.2.3.4 представлена блок-схема работы монитора изменений.

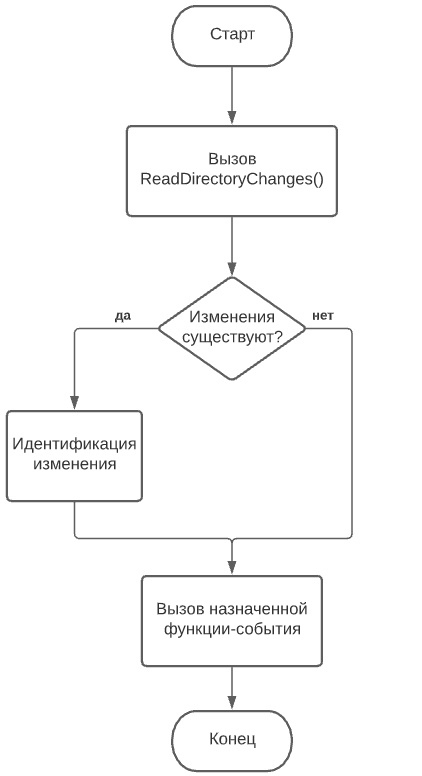


Рисунок 2.2.3.4 – Блок-схема алгоритма монитора изменений

При срабатывании события создания или редактирования файла, путь к файлу и идентификатор действия отправляются в сервис фильтрации.

Сервис фильтрации содержит в себе 3 компонента:

– правила фильтрации;

– коннектор;

– обработчики.

Компонент обработчика представляет собой 3 потока:

– поток обработки команд;

– поток обработки сообщений от модуля монитора разделов;

– поток обработки сообщений от модуля драйвер коннектора.

Существует 3 основных этапа фильтрации и 1 дополнительный. На первом этапе применяются фильтры расширения файлов, на втором фильтры путей, на третьем прочие правила фильтрации, на дополнительном этапе применяются не правила, а функции, в которые передается путь к файлу. Правило фильтрации представляет собой регулярное выражение. Поэтому в правила фильтрации можно добавить только правила, осуществляемые над строкой пути к файлу, а вот с помощью функции можно задать самые различные проверки (например проверка существования файла). На  рисунке  2.2.3.5 изображена блок схема процесса фильтрации.

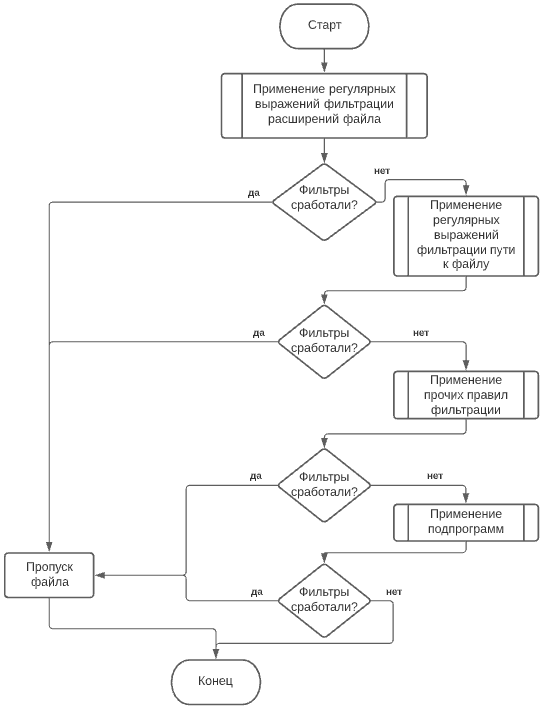


Рисунок 2.2.3.5 – Блок-схема процесса фильтрации

Если какой-либо из фильтров сработал, то файл просто пропускается, если же ни один из фильтров не сработал, то путь к файлу отправляется по именованному каналу дальше в ядро программного средства.

В ядре программного средства обработкой сообщений от модуля фильтра занимается компонент FileQueue, а который содержит в себе специально предназначенный для этого класс FilterHandler.

В таблице 2.2.3.6 приведено описание всех классов, которые содержит в себе компонент FileQueue.

Таблица 2.2.3.6 – Описание классов

|  |  |
| --- | --- |
| Класс | Описание |
| ScannerResponseHandler | Представляет собой обработчик результатов сканирования |
| FilterHandler | Представляет собой обработчик сообщений от фильтра |
| ScanTasks | Представляет собой менеджер активных задач сканирования |
| ScanTask | Представляет собой отдельную задачу сканирования |

При поступлении сообщения с путем к файлу производится проверка на наличие этого файла в листе вирусов, после этого создается задача сканирования. В таблице 2.2.3.7 приведено описание полей класса ScanTask, который представляет собой задачу сканирования.

Таблица 2.2.3.7 – Описание полей класса ScanTask

|  |  |
| --- | --- |
| Наименование поля | Описание поля |
| TaskId | Идентификатор задачи |
| File | Путь к файлу |
| ProbesCount | Количество попыток сканирования |

Все созданные задачи сканирования передаются через именованный канал прямо в модуль сканнера (ScannerService).

Сканнер при приёме задачи сканирования по именованному каналу добавляет её в свой локальный менеджер задач. Все задачи сканирования обрабатываются и выполняются потоками по мере возможности. Количество одновременно сканируемых файлов задается с помощью конфигурации. На рисунке 2.2.3.6 изображена блок-схема алгоритма поиска сигнатуры в файле.

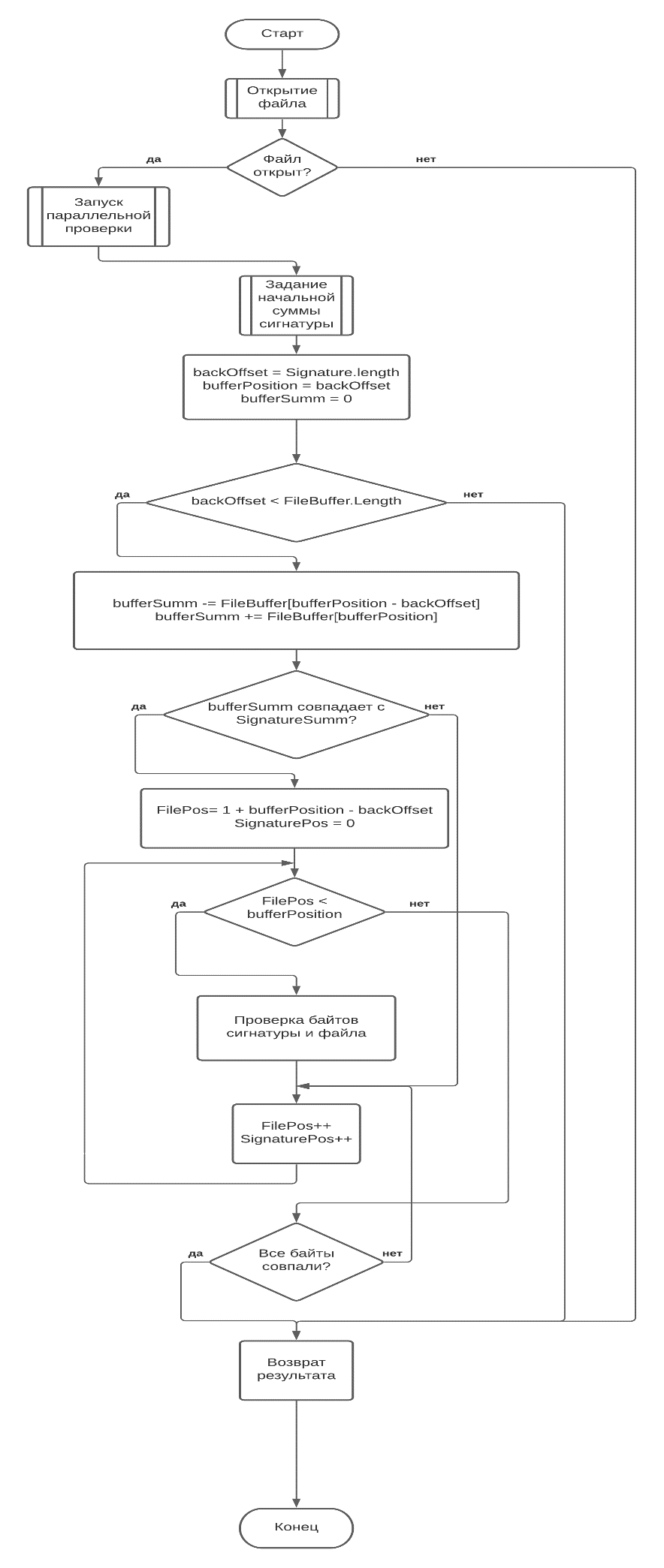


Рисунок 2.2.3.6 – Блок-схема алгоритма поиска сигнатуры

В целях увеличения производительности алгоритма, проверка байтов файла и сигнатуры может производится параллельно.

После того как задача сканирования была выполнена, результат сканирования возвращается в компонент FileQueue. Если задача была выполнена с ошибкой, то она повторяется ещё раз. Количество проб зависит от конфигурации компонента. Если задача была выполнена и в файле была обнаружена сигнатура вируса, то файл передается в компонент FoundVirusesManager. Если же в файле не была обнаружена вирусная сигнатура, то задача просто удаляется.

На рисунке 2.2.3.7 приведена блок схема алгоритма проверки файла начиная с создания задачи сканирования и заканчивая её удалением.

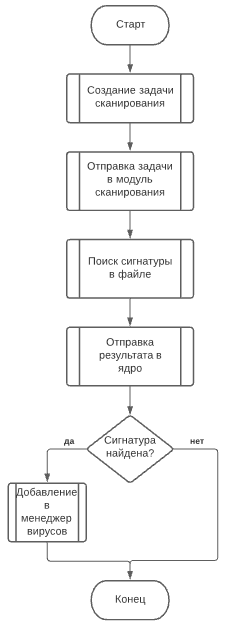


Рисунок 2.2.3.7 – Блок-схема алгоритма проверки файла

Компонент FoundVirusesManager содержит в себе полную информацию о файле и его принадлежности к вирусу. В таблице 2.2.3.8 приведено описание полей класса VirusInfo, который представляет собой информацию о вирусе.

Таблица 2.2.3.8 – Описание полей класса VirusInfo

|  |  |
| --- | --- |
| Наименование поля | Описание поля |
| Id | Идентификатор вируса в таблице менеджера |
| inQuarantine | Находится ли вирус в карантине |
| fileInQuarantine | Путь к файлу в карантине |
| File | Путь к файлу на жестком диске |
| VirusId | Идентификатор вирусной сигнатуры |

Если в настройках программного средства включена опция авто перемещения вирусов в карантин, то вирус будет немедленно перемещен в карантин. За реализацию карантина отвечает компонент Quarantine.

Карантин представляет собой изолированное хранилище, доступ к которому может получить только та программа, которая создала это хранилище. В изолированном хранилище данные всегда изолированы по пользователю и по сборке. Учетные данные, такие как происхождение или строгое имя сборки, определяют идентичность сборки. Данные также могут быть изолированы по домену приложения с использованием аналогичных учетных данных. Для того, чтобы файл поместить в карантин, необходимо его сначала скопировать в защищенное хранилище, а затем удалить оригинал на жестком диске. На рисунке 2.2.3.8 изображена блок-схема алгоритма перемещения файла в карантин.

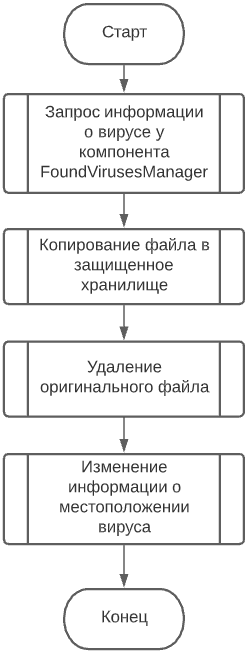


Рисунок 2.2.3.8 – Блок–схема алгоритма перемещения файла в карантин

Что бы другие программные средства могли взаимодействовать с ядром был реализован специальный API. Библиотека API связывается с ядром с помощью именованного канала. Ядро таким образом может принимать команды или уведомлять стороннее программное средство при возникновении различных событий (например событие обнаружение вируса). В приложении А изображена диаграмма последовательности действий работы ядра совместно с графическим интерфейсом пользователя.

Для осуществления действий над обнаруженными вирусами требуется вмешательство пользователя. На рисунке 2.2.3.9 изображена блок-схема взаимодействия ядра программного средства с пользователем при обнаружении вируса.



Рисунок 2.2.3.9 – Блок–схема взаимодействия

В таблице 2.2.3.9 приведено описание функционала библиотеки API.

Таблица 2.2.3.9 – Описание функций библиотеки API

|  |  |
| --- | --- |
| Функция | Назначение |
| 1 | 2 |
| ToQuarantine | Поместить вирус в карантин |
| RestoreFile | Восстановить файл из карантина |
| GetVirusInfo | Получить информацию о вирусе |
| ApplyingActions | Применить действия к вирусам |
| ClearScanQueue | Очистить очередь сканирования |
| AddToScan | Добавить файл для проверки в очередь сканирования |
| SetAutoScanRemovableDevices | Установить опцию авто сканирования съемных носителей |

Окончание таблицы 2.2.3.9

|  |  |
| --- | --- |
| 1 | 2 |
| getAllViruses | Получить информацию о всех вирусах |
| DeleteFile | Удалить файл, где бы он не находился (в карантине или на жестком диске) |
| ClearConnectedDevices | Очистить таблицу подключенных устройств |
| AddSimpleRule | Добавить простое правило фильтрации |
| RemoveSimpleRule | Удалить простое правило фильтрации |
| ClearSimpleRules | Удалить все простые правила фильтрации |

В таблице 2.2.3.10 приведено описание событий библиотеки API.

Таблица 2.2.3.10 – Описание событий библиотеки API

|  |  |
| --- | --- |
| Событие | Назначение |
| onScanCompleted | Срабатывает при выполнении сканирования |
| onScanFound | Срабатывает при обнаружении вируса |
| onVirusInfo | Срабатывает при получении информации о вирусе |

2.2.4 Используемые технические средства

В таблице 2.2.4.1 приведены минимальные аппаратно-программные характеристики компьютера пользователя.

Таблица 2.2.4.1 – Минимальные аппаратно-программные характеристики компьютера пользователя

|  |  |
| --- | --- |
| Компонент | Требование |
| Процессор | Любой x86 совместимый с тактовой частотой не менее 1GHz |
| Оперативная память | 1 GB RAM |
| Жесткий диск | 50MB HDD |
| Операционная система | Windows 8.1 |
| .NET Framework | V4.7.2 |

В таблице 2.2.4.2 приведено описание минимальных аппаратно-программных характеристик для .NET Framework.

Таблица 2.2.4.2 – Минимальные аппаратно-программные требования для .NET Framework

|  |  |
| --- | --- |
| Компонент | Требование |
| Оперативная память | 512MB |
| Свободное место на жестком диске | 4.5GB |
| Процессор | Любой x86 совместимый с тактовой частотой не менее 1GHz |
| Операционная система | Windows 7 SP1 |

В системах Windows 10 версии 1803 .NET Framework установлен по умолчанию.

2.2.5 Вызов и загрузка

Для обеспечения максимальной безопасности загрузка программного средства производится автоматически при загрузке операционной системы.

При запуске компьютера происходит инициализация ядра операционной системы, во время которой в ядро загружается драйвер защиты программного средства. Драйвер защиты программного средства просматривает все события, связанные с процессами в операционной системе, при этом блокируя все действия, связанные с процессом программного средства.

После того, как очередь дойдёт до запуска пользовательских приложений, ядро операционной системы запустит исполняемый файл ядра программного средства. Программное средство в свою очередь инициализирует конфигурацию, проверяет настройки операционной системы, затем загружает библиотеки динамической компоновки (DLL).

После полной загрузки операционной системы, в трее появится значок программного средства.

Количество потоков внутри программного средства может быть различно, всё зависит от мощности компьютера. В таблице 2.2.5.1 приведено описание потребления оперативной памяти при различных состояниях.

Таблица 2.2.5.1 – Описание потребления оперативной памяти при различных состояниях

|  |  |  |
| --- | --- | --- |
| Состояние | Объём потребляемой физической памяти | Объём потребляемой виртуальной памяти |
| 1 | 2 | 3 |
| Программное средство запущено и работает в фоновом режиме | 36 МБ | 268 МБ |

Окончание таблицы 2.2.5.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Программное средство запущено и отображается графический интерфейс | 39МБ | 280 МБ |
| Запущена проверка папки размером 1.5 Гб | 300 МБ | 290 МБ |
| Программное средство запущено в фоновом режиме, защита компьютера отключена | 30 МБ | 268 МБ |

2.2.6 Входные данные

Все входные данные в модули передаются с помощью именованных каналов. Именованные каналы представляют односторонние или дуплексные каналы для связи между сервером канала и одним или несколькими клиентами. Именованные каналы могут использоваться для межпроцессного взаимодействия локально или по сети. Любой процесс может действовать как сервер или клиент именованного канала.

Входные именованные каналы модуля фильтра описаны в таблице 2.2.6.1.

Таблица 2.2.6.1 – Описание входных каналов модуля фильтра

|  |  |
| --- | --- |
| Именованный канал | Назначение |
| Filter.CommandPipe | Канал для команд |
| API\_MON\_FILTER | Канал для приёма сообщений от модуля мониторинга разделов жесткого диска и съемных носителей |
| DRIVER MON | Канал для приёма сообщений от драйвера мониторинга операций ввода-вывода |

Для выполнения команды в модуле фильтра, необходимо сначала отправить байт идентификатора команды, затем различные параметры, которые зависят от конкретной команды. Доступные команды модуля фильтра описаны в таблице 2.2.6.2.

Таблица 2.2.6.2 – Описание команд модуля фильтрации

|  |  |  |
| --- | --- | --- |
| Идентификатор команды | Передаваемые параметры | Назначение команды |
| 0 | Строка (регулярное выражение) | Добавить правило фильтрации расширения файла |
| 1 | Добавить правило фильтрации путей к файлам |
| 2 | Добавить прочее правило фильтрации |
| 3 | Строка | Добавить простое правило фильтрации |
| 4 | Удалить простое правило фильтрации |
| 5 | Ничего | Удалить все простые правила фильтрации |
| 6 | Ничего | Выключить модуль фильтрации |
| 7 | Очистить все правила |

Отличие обычного правила фильтрации от простого состоит в том, что обычное правило представляет собой регулярное выражение, простое правило это обычная строка. Путь считается отфильтрованным, если есть совпадение по регулярному выражению, или если строка простого правила была найдена в проверяемой строке. В таблице 2.2.6.3 приведены примеры правил фильтрации.

Таблица 2.2.6.3 – Правила фильтрации

|  |  |
| --- | --- |
| Правило | Назначение |
| ([^\s]+(?=\.(jpg|gif|png))\.\2) | Фильтрация пути к файлу с расширением jpg, png и gif |
| ^([a-zA-Z]:)?(\\[^<>:"/\\|?\*]+)+\\?$ | Фильтрация пути при присутствии в нём недопустимых символов |

В таблице 2.2.6.4 приведен пример простых правил фильтрации

Таблица 2.2.6.4 – Простые правила фильтрации

|  |  |
| --- | --- |
| Правило | Назначение |
| 1 | 2 |
| amdForever.dot | Фильтрация всех путей, ведущих к файлу amdForever.dot |

Окончание таблицы 2.2.6.4

|  |  |
| --- | --- |
| 1 | 2 |
| myName | Фильтрация папки или файла с именем myName |
| \myFolder\ | Фильтрация папки с именем myFolder |

Модули мониторинга разделов, криптографа, API и вирусной базы сигнатур в своём составе имеют только 1 именованный канал, предназначенный для приёма команд.

В таблице 2.2.6.5 приведено описание доступных команд модуля мониторинга разделов.

Таблица 2.2.6.5 – Описание команд модуля мониторинга разделов

|  |  |  |
| --- | --- | --- |
| Идентификатор команды | Передаваемые параметры | Назначение команды |
| 0 | Строка (путь к папке или накопителю) | Создать монитор изменений папки или раздела |
| 1 | Строка (путь к папке или накопителю) | Удалить монитор изменений папки или раздела |
| 2 | Ничего | Включить автопроверку съемных носителей |
| 3 | Выключить автопроверку съемных носителей |
| 4 | Очистить буфер информации о подключенных съемных носителях |
| 5 | Приостановить мониторинг разделов и съемных носителей |
| 6 | Ничего | Активировать мониторинг разделов и съемных носителей |
| 7 | Выключить модуль |

В таблице 2.2.6.6 приведено описание доступных команд модуля локальной базы вирусных сигнатур. Модуль локальной базы сигнатур единственный из всех модулей, команды в который передаются в текстовом виде.

Таблица 2.2.6.6 – Описание команд локальной базы сигнатур

|  |  |
| --- | --- |
| Команда | Назначение команды |
| /reinit\_db | Перезагрузить всю информацию о вирусах |
| /upload\_to\_scanner | Выгрузить все сигнатуры в модуль сканнера |
| /shutdown | Выключить модуль |

В таблице 2.2.6.7 приведено описание входных именованных каналов модуля сканнера.

Таблица 2.2.6.7 – Описание входных именованных каналов сканнера

|  |  |
| --- | --- |
| Именованный канал | Назначение |
| Scanner.CommandPipe | Канал для команд |
| ScannerService.Input | Канал для приёма путей к сканируемым файлам |
| ScannerService.Signatures | Канал для приёма сигнатур вирусов от модуля базы сигнатур |

2.2.7 Выходные данные

Выходные данные в модулях передаются так же, как и входные ­– с помощью именованных каналов

В таблице 2.2.7.1 приведено описание выходных именованных каналов.

Таблица 2.2.7.1 – Описание выходных именованных каналов

|  |  |  |
| --- | --- | --- |
| Наименование модуля | Наименование канала | Назначение |
| Filter | Filter.Output | Канал для передачи файлов для проверки |
| ScannerService | ScannerService.Output | Канал для передачи проверенных файлов |
| Cryptographer | Crypto.Data | Канал для передачи данных о процессе шифровки или расшифровки |
| API | API.User | Канал для передачи данных событий |
| API Monitor | API MON FILTER | Канал для отправки путей к файлам |
| VirusesDB | ScannerService.Signatures | Канал для передачи сигнатур в модуль сканнера |

В таблице 2.2.7.2 приведено описание данных, которые могут быть переданы по каналу API.User.

Таблица 2.2.7.2 – Описание выходных данных

|  |  |  |
| --- | --- | --- |
| Идентификатор | Назначение | Выходные данные |
| 0 | Путь к просканированному файлу | Строка пути к файлу |
| 1 | Путь к найденному вирусу | Путь к файлу, системный идентификатор, идентификатор вируса |
| 2 | Полная информация о найденном вирусе | Системный идентификатор, путь к файлу, идентификатор вируса, флаг нахождения в карантине, путь к файлу в карантине |
| 4 | Системная информация | Системная информация о файле |
| 5 | Информация о состоянии ядра | Состояния потоков ядра, подключенных модулей, компонентов |
| 6 | Статистика | Статистика по найденым вирусам |

Все данные, касаемо просканированных файлов и найденых вирусов передаются в клиент автоматически, но могут быть выданы по запросу клиента (например, если пользователь решил просмотреть информацию по вирусам, которые находятся в карантине).

Данные о состоянии и системной информации передаются только по запросу клиента. Если при передаче клиент не был подключен, то вся информация пропускается

2.2.8 Рекомендации по использованию

Программное средство запускается автоматически при загрузке системы. По умолчанию главная форма скрыта, чтобы её открыть необходимо сделать двойной щелчок по иконке в трее либо открыть контекстное меню и нажать на кнопку «Открыть» (рисунок 2.2.8.1)

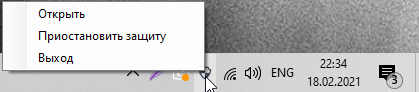


Рисунок 2.2.8.1 – Контекстное меню иконки

На рисунке 2.2.8.2 изображено главное окно программы

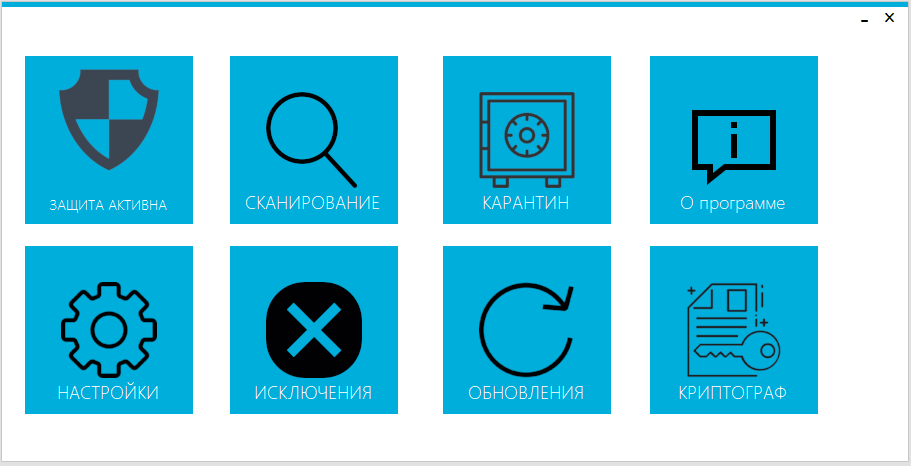


Рисунок 2.2.8.2 – Главное окно

При нажатии на кнопку «Сканирование» открывается окно настроек ручного сканирования.

При нажатии на кнопку «Настройки» открывается окно настроек программного средства.

При нажатии на кнопку «Карантин» открывается окно обозревателя карантина, в котором можно просмотреть файлы, находящиеся в карантине.

При нажатии на кнопку «Исключения» открывается окно настройки исключаемых из сканирования файлов и папок.

При нажатии на кнопку «Обновления» открывается окно, в котором можно просмотреть версию программного средства и вирусной базы сигнатур.

На рисунке 2.2.8.3 изображено окно настроек сканирования.

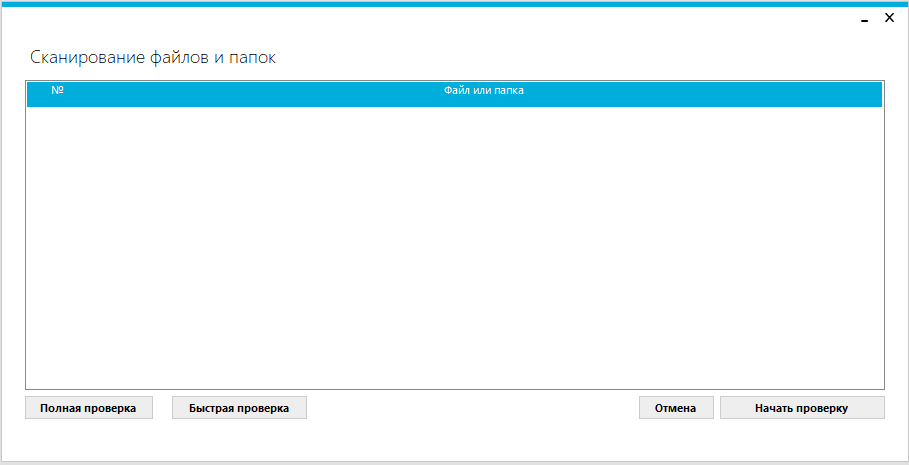


Рисунок 2.2.8.3 – Окно настроек сканирования

На форме расположены 4 кнопки:

* «Полная проверка», при нажатии на которую запускается процесс проверки всех файлов на всех разделах доступных жестких дисков;
* «Быстрая проверка», запускает проверку только исполняемых файлов на всех разделах доступных жестких дисков, за исключением системного раздела;
* «Отмена», возвращает на главную страницу;
* «Начать проверку», запускает проверку файлов.

Папки и файлы, которые необходимо проверить отображаются в списке. Проверяемые файлы и папки добавляются в список через контекстное меню (рисунок 2.2.8.4).

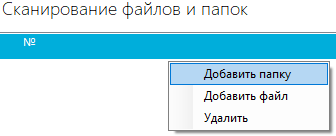


Рисунок 2.2.8.4 – Контекстное меню списка объектов сканирования

После старта сканирования открывается окно, на котором отображается ход сканирования (рисунок 2.2.8.5).

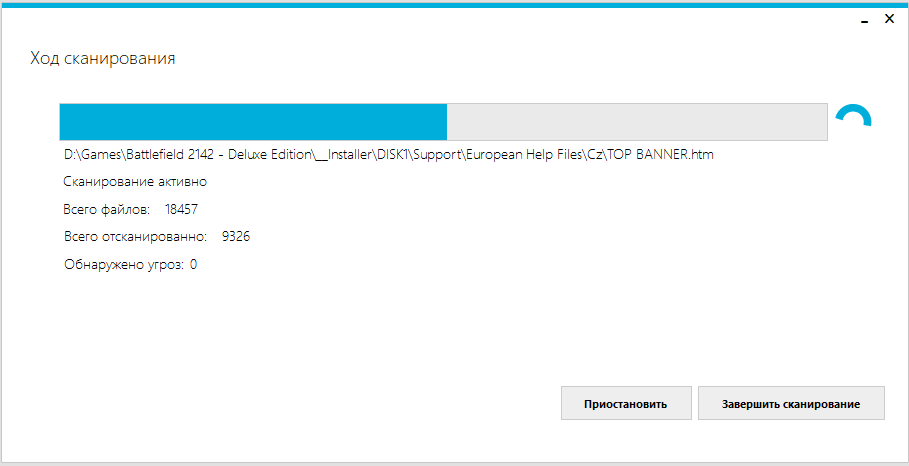


Рисунок 2.2.8.5 – Ход сканирования

На форме располагается 2 кнопки:

* «Приостановить», при нажатии на которую процесс сканирования будет приостановлен (но файлы, которые уже были поставлены в очередь на сканирование, будут просканированы вне зависимости от приостановки);
* «Завершить сканирование», при нажатии на которую сканирование будет завершено.

После того, как все файлы будут проверены, откроется окно с результатами сканирования (рисунок 2.2.8.6).

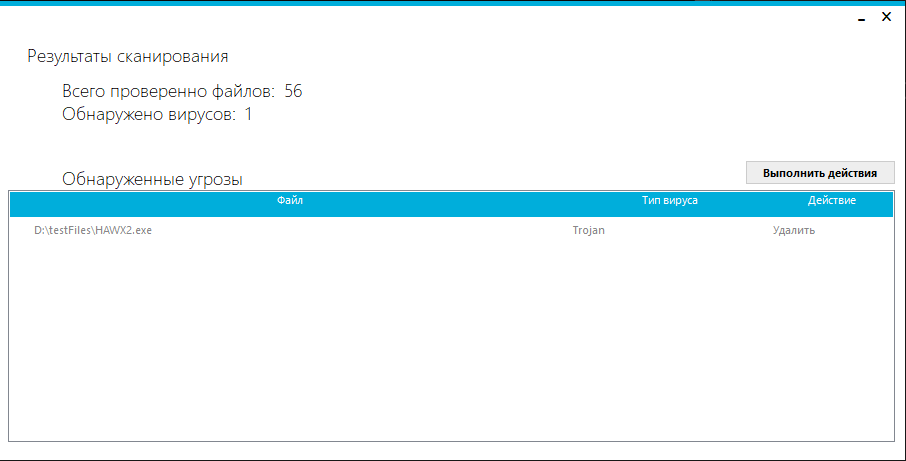


Рисунок 2.2.8.6 – Результат сканирования

В таблице расположена информация об обнаруженных вирусах (путь к файлу, тип вируса, осуществляемое действие). По умолчанию ко всем вирусам применяется удаление. Изменить действие можно через контекстное меню (рисунок 2.2.8.7).

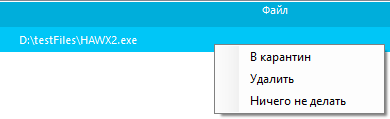


Рисунок 2.2.8.7 – Контекстное меню выбора действия

Все помещенные в карантин вирусы можно просмотреть в окне обозревателя карантина (рисунок 2.2.8.8).

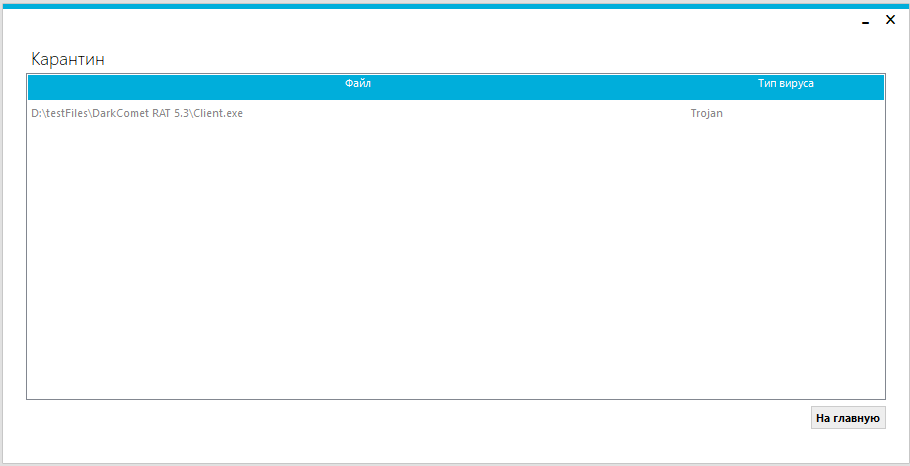


Рисунок 2.2.8.8 – Обозреватель карантина

Файлы, находящиеся в карантине, можно либо восстановить, либо удалить. Все действия осуществляются через контекстное меню (рисунок 2.2.8.9).

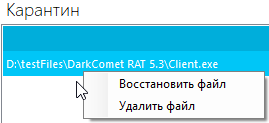


Рисунок 2.2.8.9 – Выбор действия над файлом в карантине

Находящийся в карантине файл можно либо восстановить, либо удалить.

На рисунке 2.2.8.10 изображена форма настроек. На форме настроек можно включить или выключить уведомления, автоматическую проверку съемных носителей (например флэш-накопителей) или установить действие, применяемое к обнаруженным вирусам по умолчанию.

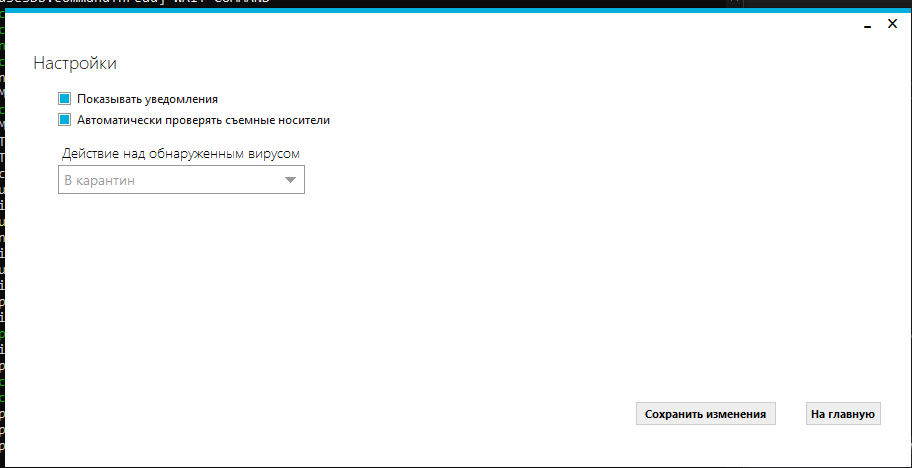


Рисунок 2.2.8.10 – Форма настроек

Для редактирования базы сигнатур существует утилита DBEditor. На рисунке 2.2.8.11 представлено главное окно редактора базы сигнатур.

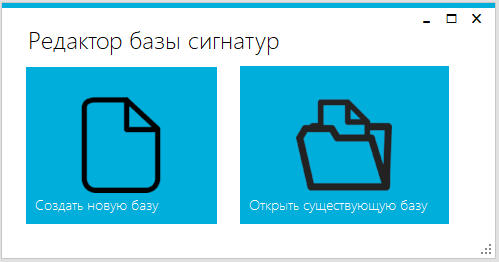


Рисунок 2.2.8.11 – Главное окно редактора сигнатур

После создания или открытия базы сигнатур появляется окно редактора. В таблице редактора отображается таблица, в которой есть 3 колонки, в которых отображается наименование вируса, его сигнатура и тип. На рисунке 2.2.8.12 изображено окно редактора.

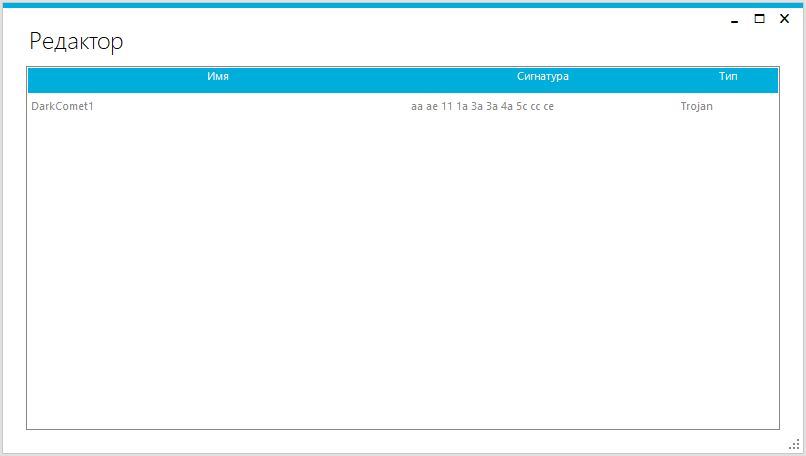


Рисунок 2.2.8.12 – Окно редактора

Добавление удаление и редактирование информации о вирусах в базе сигнатур производится через контекстное меню (рисунок 2.2.8.13).

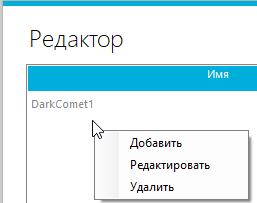


Рисунок 2.2.8.13 – Контекстное меню таблицы редактора

На рисунке 2.2.8.14 изображено окно добавления и редактирования информации о вирусе. Информацию о вирусе представляет собой:

* тип вируса;
* наименование вируса;
* сигнатура вируса, представляющая собой последовательность байт (на форме байты вводятся в виде шестнадцатеричной системы счисления).

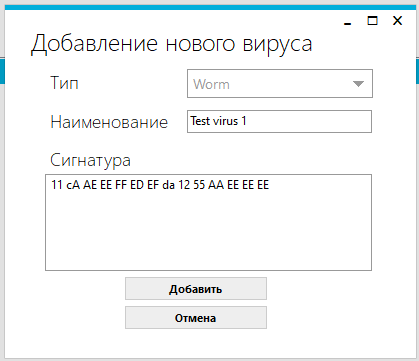


Рисунок 2.2.8.14 – Форма добавления и редактирования информации о вирусе

Шестнадцатеричная система включает в себя цифры от 0 до 10, а также буквы от A до F. Если не заполнить какое либо поле, либо ввести некорректную сигнатуру, то будет выдано соответствующее сообщение об ошибке (рисунок 2.2.8.15).

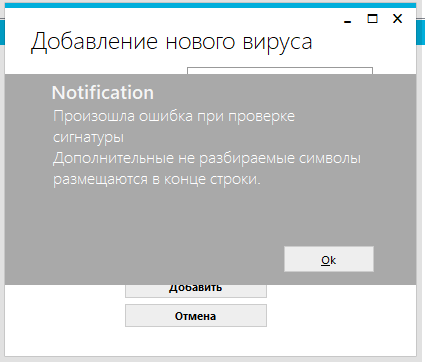


Рисунок 2.2.8.15 – Сообщение с ошибкой

Для сохранения всех изменений в базе, необходимо нажать на кнопку «Применить все изменения», которая появляется слева вверху.

3 Ожидаемые технико-экономические характеристики

3.1 Экономический раздел

Особое внимание в дипломном проекте должно быть уделено экономическим показателям. Целью экономического обоснования разработки является количественное и качественное доказательство экономической целесообразности разработанного программного средства, а также определение организационно-экономических условий его эффективного функционирования.

План разработки программного продукта представлен в таблице 3.1.1.

Таблица 3.1.1 – План разработки программного продукта

|  |  |
| --- | --- |
| Наименование этапов и видов работ | Трудоемкость, человеко-дни |
| Подготовительный этап – разработка и утверждение технического задания:  а) составление календарного графика работ | 2  1 |
| б) подбор и изучение литературы по теме | 1 |
| Изучение предметной области: | 3 |
| а) изучение аналогов | 1 |
| б) изучение основных понятий и объектов предметной области | 2 |
| Проектирование программного средства: | 3 |
| а) определение требований | 1 |
| б) определение логической структуры | 2 |
| Разработка программного средства | 7 |
| Тестирование и испытания | 1 |
| Внесение корректировок | 2 |
| Всего | 18 |

Месячная тарифная ставка каждого исполнителя определяется путем умножения действующей месячной тарифной ставки 1-го разряда на тарифный коэффициент, соответствующий установленному тарифному разряду (3.1.1).

МТСnр = МТС1р \* Кnр, (3.1.1)

где МТСnр – тарифная ставка за месяц n-го разряда, руб.;

МТС1р – тарифная ставка 1-го разряда за месяц, руб. (в 2021 г.);

Кnр – тарифный коэффициент n-го разряда.

МТС8р= 41 \* 2,17 = 88,97 руб.

Дневная тарифная ставка n-го разряда определяется делением месячной тарифной ставки n-го разряда на месячный норматив рабочего времени в часах и умножением на продолжительность рабочего дня в часах (3.1.2).

ДТСnр= \* 8, (3.1.2)

где МТСnр – тарифная ставка за месяц n-го разряда, руб.;

169.2 – месячный норматив рабочего времени, часы, (2021 г.).

ДТС8р= \* 8 = 4,20 руб.

Основная заработная плата исполнителей на конкретное ПС рассчитывается по формуле (3.1.3).

ЗПо = ДТСnр \* То \* Кпр, (3.1.3)

где ЗПо – основная заработная плата, руб.;

ДТСnр – тарифная ставка за день, руб.;

То – общая трудоемкость ПС, человеко-дней;

Кпр – коэффициент премирования (1,1 – 1,3).

ЗПо = 4,20 \* 18 \* 1,3 = 98,28 руб.

Дополнительная заработная плата на конкретное ПС включает выплаты, предусмотренные законодательством о труде (выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате (3.1.4).

Ндзп ≈ от 10 до 25%

ЗПд = , (3.1.4)

где ЗПд – дополнительная заработная плата на конкретное ПС, руб.;

Ндзп – норматив дополнительной заработной платы, %.

ЗПд = = 14,74 руб.

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей (3.1.5).

Офсзн = , (3.1.5)

где Офсзн – отчисления в Фонд социальной защиты населения, руб.;

Нфсзн – норматив отчислений в Фонд социальной защиты населения (34%).

Офсзн = = 38,42 руб.

Отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами в зависимости от уровня риска отрасли, к которой относится организация-разработчик (3.1.6).

, (3.1.6)

где Обгс – отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, руб.;

Нбгс – норматив отчислений по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний (0,6%).

Обгс = = 0,67 руб.

Расходы на материалы определяются с учетом действующих нормативов. По данной статье отражаются расходы на покупные полуфабрикаты и комплектующие изделия, необходимые для разработки ПС. Расчет целесообразно представить в табличной форме (таблица 3.1.2).

Таблица 3.1.2 – Расчет затрат на материалы, покупные полуфабрикаты и комплектующие изделия

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование материалов покупных полуфабрикатов и комплектующих изделий | Единица измерения | Количество | Цена приобретения, руб. | Сумма, руб. |
| Брошюрирование | шт. | 1 | 3,46 | 3,46 |
| Лист | блок | 0,5 | 8,00 | 4,00 |
| Диск | шт. | 1 | 2,75 | 2,75 |
| Конверт | шт. | 1 | 1,5 | 1,5 |
| Всего расходов | | | | 11,71 |

При создании ПП компьютер является неотъемлемым атрибутом. Такая техника имеет свои отличия в комплектации и мощности. От этих параметров и зависит общий расход электроэнергии. Как правило, чем мощнее ПК, тем больше киловатт он потребляет

В основном компьютер со средней мощностью потребляет от 250 до 350 ватт в час (0,25 – 0,35 кВт/ч). Более мощный процессор может потреблять до 450 ватт в час (0,45 кВт/ч). Расчет можно произвести по формуле 3.1.7

, (3.1.7)

где – установочная мощность i-го объекта основных производственных фондов, используемых для выполнения работ по данной теме, кВт;

– время фактического использования i-го объекта, ч;

– тариф за 1 кВт/ч энергии, руб., тариф уточняется в бухгалтерии предприятия (таблица 3.1.3).

руб.

Таблица 3.1.3 – Расчет затрат на топливно-энергетические ресурсы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование оборудования, используемого для научно-экспериментальных и технологических целей | Установочная мощность, кВт | Время использования, ч | Тариф за 1 кВт/ч | Сумма затрат, руб. |
| Персональный компьютер | 0,40 | 176 | 0,3777 | 26,59 |
| Всего затрат | | | | 26,59 |

Определяются прочие прямые расходы, связанные с амортизационными отчислениями на полное восстановление основных производственных фондов, арендная плата и лизинговые платежи, компенсация за износ (амортизацию) использованного в процессе создания научно-технической продукции оборудования по договоренности (4.8).

Рпр = ЗПо , (3.1.8)

где Нпр – норматив прямых расходов, Нпр ≈ от 10 до 20%.

Рпр = руб.

В процессе использования ОС изнашиваются и переносят свою стоимость по частям на изготавливаемую продукцию (3.1.9).

Ао = , (3.1.9)

где Ао – амортизационные отчисления, руб.;

– первоначальная стоимость объекта, руб.;

t – срок полезного использования.

Ао(год) = руб.

Ао(мес)  руб.

Данные затраты, связанные с необходимостью содержания аппарата управления, а также с расходами на общехозяйственные нужды, относятся на конкретное ПС по нормативу в процентном отношении к основной заработной плате исполнителей (3.1.10).

Рнр = , (3.1.10)

где Рнр – расчет накладных расходов, руб.;

Ннр – норматив накладных расходов по организации (≈ от 25 до 50%).

Рнр = руб.

Общая сумма расходов на ПС рассчитывается по формуле 3.1.11

Cп = ЗПо + ЗПд + Офсзн + Обгс + См + Рэл + Рпр + Ао + Рнр (3.1.11)

где Сп – сумма расходов на разработку программного средства, руб.;

ЗПо – основная заработная плата, руб.;

ЗПд – дополнительная заработная плата на программное средство, руб.;

Офсзн – отчисления в Фонд социальной защиты населения, руб.;

Обгс – отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, руб.;

См – сумма расходов на материалы, руб.;

Рэл – расчет стоимости топливно-энергетических ресурсов, руб.;

Рпр – расчет прочих прямых затрат, руб.;

Ао – амортизационные отчисления, руб.;

Рнр – расчет накладных расходов, руб.

Cп = 98,28 +14,74 +38,42 + 0,67 + 11,71+ 26,59 + 14,74 + 29,16 +24,57 = 258,88 руб.

По среднему уровню рентабельности в процентах от полной себестоимости определяется плановая прибыль ПС (3.1.12).

П = Сп, (3.1.12)

где Ур - средний уровень рентабельности, Ур от 10 до 30%.

П 77,66

Определяется приближенная (ориентировочная) отпускная цена ПС по формуле 3.1.13.1.

Цотп = Сп + П, (3.1.13)

Цотп =

По темам, выполняемым за счет внебюджетных средств (собственных средств, средств других предприятий и организаций, предпринимателей), определяется налог на добавленную стоимость по формуле 3.1.14.

НДС = Цотп , (3.1.14)

где Нндс – ставка налога (НДС), Нндс = 20%.

НДС

Отпускная цена определяется на основании цены разработчика, которая формируется на основе показателя рентабельности продукции. Рентабельность и прибыль по создаваемому ПС определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей налог на добавленную стоимость (3.1.15).

Цотп = Цотп + НДС, (3.1.15)

Цотп =

Все приведенные выше расчеты представлены в сводной таблице 3.1.4.

Таблица 3.1.4 – Расчет ориентировочной цены ПС

|  |  |  |
| --- | --- | --- |
| Статьи затрат | Условные обозначения | Сумма, руб. |
| 1 | 2 | 3 |
| 1. Основная заработная плата | ЗПо | 98,28 |
| 2. Дополнительная заработная плата | ЗПд | 14,74 |
| 3. Отчисления в ФСЗН | Офсзн | 38,42 |
| 4.Отчисления по обязательному страхованию от несчастных случаев на производстве | Обгс | 0,67 |
| 5. Материалы, покупные полуфабрикаты и комплектующие изделия | См | 14,75 |
| 6. Топливно-энергетические ресурсы для научно-экспериментальных целей | Рэл | 26,59 |
| 7. Прочие прямые расходы | Рпр | 14,74 |
| 8. Амортизационные отчисления | Ао | 29,16 |
| 9. Накладные расходы | Рнр | 24,57 |

Окончание таблицы 3.1.4

|  |  |  |
| --- | --- | --- |
| 10. Полная себестоимость | Cп | 258,88 |
| 11. Плановые накопления (прибыль) | П | 77,66 |
| 12. Отпускная цена (без НДС) | Цотп | 336,54 |
| 13.1. Налог на добавленную стоимость | НДС | 67,30 |
| 14. Отпускная цена с НДС | Цотп | 403,84 |

Вывод: Отпускная стоимость ПС с НДС составила 403,84 руб.

**3.2** Охрана труда

3.2.1 Характеристика трудовой деятельности и факторов среды при работе оператора

В процессе работы на оператора могут воздействовать следующие опасные и вредные производственные факторы:

– движущиеся машины и механизмы;

– подвижные части производственного оборудования;

‒ повышенная запыленность воздуха рабочей зоны;

‒ повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;

‒ повышенная или пониженная температура воздуха рабочей зоны;

‒ пониженная или повышенная влажность воздуха рабочей зоны;

‒ повышенный уровень шума на рабочем месте;

‒ повышенная или пониженная ионизация воздуха;

‒ повышенный уровень ионизирующих излучений;

‒ повышенный уровень статического электричества;

‒ повышенный уровень электромагнитного излучения;

‒ повышенная напряженность электромагнитного поля;

‒ повышенная напряженность электростатического поля;

‒ повышенная яркость света;

‒ прямая и отраженная блескость;

‒ статические перегрузки костно-мышечного аппарата и динамические локальные перегрузки мышц кистей рук;

‒ перенапряжение зрительного анализатора;

‒ умственное перенапряжение;

‒ монотонность труда;

‒ эмоциональные перегрузки.

Производственная (рабочая) среда включает в себя все, что окружает человека в процессе трудовой деятельности: техническое оснащение организации, особенности технологических процессов и производства, состояние зданий, строений, сооружений и инженерных коммуникаций, санитарно-гигиеническую и эстетическую обстановку, взаимоотношения в трудовом коллективе, уровень профессионального риска исходя из идентифицированных опасных и вредных производственных факторов и пр.

Производственные факторы делят на 2 типа: вредные и опасные. Вредный производственный фактор – это производственный фактор, воздействие которого на работника может привести его к заболеванию. Опасный производственный фактор – это производственный фактор, воздействие которого на работника может привести к его травме.

Оператор должен быть обеспечен средствами индивидуальной защиты в соответствии с типовыми отраслевыми нормами бесплатной выдачи средств индивидуальной защиты работникам связи, утвержденными Постановлением Министерства труда и социальной защиты Республики Беларусь от 4 мая 2012 г. № 55 (Национальный реестр правовых актов Республики Беларусь, 2012 г., № 76, 8/12561).

3.2.2 Организация общественного контроля условий и безопасности труда

Целью общественного контроля условий и безопасности труда является усиление профилактической работы по предупреждению травматизма, нарушений правил, норм и инструкций по охране труда, соблюдение требований трудового законодательства и законных интересов работников, а также повышение персональной ответственности руководителей подразделений и непосредственных исполнителей за безопасное выполнение работ.

**Основными видами контроля за состоянием охраны труда в организациях являются:**

**– оперативный контроль руководителя работ и других должностных лиц;**

– административно–общественный контроль;

– контроль, осуществляемый службой охраны труда.

Про­из­вод­ст­вен­ный кон­троль – кон­троль, обес­пе­чи­вае­мый производите­лем про­дук­ции (то­ва­ров), ра­бот и ус­луг, за со­блю­де­ни­ем требова­ний са­ни­тар­но-эпи­де­мио­ло­ги­че­ско­го за­ко­но­да­тель­ст­ва и выполнени­ем са­ни­тар­но-про­ти­во­эпи­де­ми­че­ских и про­фи­лак­ти­че­ских мероприятий, на­прав­лен­ный на со­хра­не­ние жиз­ни и здо­ро­вья лю­дей, сре­ды оби­та­ния при про­из­вод­ст­ве про­дук­ции (то­ва­ров), вы­пол­не­нии ра­бот и услуг.

Вход­ной кон­троль – кон­троль сы­рья и (или) про­дук­ции, по­сту­пив­ших к по­тре­би­те­лю или за­каз­чи­ку, пред­на­зна­чен­ных для даль­ней­ше­го использова­ния в про­из­вод­ст­ве.

Тех­но­ло­ги­че­ский кон­троль – кон­троль тех­но­ло­ги­че­ских па­ра­мет­ров про­из­вод­ст­ва про­дук­ции (то­ва­ров) в про­цес­се из­го­тов­ле­ния и ока­за­ния услуг.

Инспекционный контроль – контроль, осуществляемый уполномоченными органами с целью надзо­ра за со­блю­де­ни­ем тре­бо­ва­ний са­ни­тар­ных норм, пра­вил и ги­гие­ни­че­ских нор­ма­ти­вов.

Объ­ек­та­ми то­го или ино­го ви­да кон­тро­ля яв­ля­ют­ся производственные, об­ще­ст­вен­ные по­ме­ще­ния, зда­ния, со­ору­же­ния, санитар­но-за­щит­ные зо­ны, зо­ны са­ни­тар­ной ох­ра­ны, ат­мо­сфер­ный воз­дух, водо­снаб­же­ние, транс­порт, тех­но­ло­ги­че­ские про­цес­сы и обо­ру­до­ва­ние, сырье, про­из­во­ди­мая про­дук­ция (то­ва­ры), ус­ло­вия хра­не­ния, транспортиров­ки и реа­ли­за­ции, ра­бо­чие мес­та, вклю­чая ра­бо­таю­щих и оказы­вае­мые ус­лу­ги на­се­ле­нию.

Про­из­вод­ст­вен­ный кон­троль вклю­ча­ет:

* на­ли­чие офи­ци­аль­но из­дан­ных са­ни­тар­ных пра­вил и норм, аттестован­ных ме­то­дов и ме­то­дик кон­тро­ля сы­рья, про­дук­ции, про­из­вод­ст­ва, фак­то­ров сре­ды оби­та­ния в со­от­вет­ст­вии с осу­ще­ст­в­ляе­мой дея­тель­но­стью и дру­гой нор­ма­тив­ной до­ку­мен­та­ции;
* осу­ще­ст­в­ле­ние (ор­га­ни­за­цию) ла­бо­ра­тор­ных (тех­но­ло­ги­че­ских) иссле­до­ва­ний и ис­пы­та­ний:
* на гра­ни­це са­ни­тар­но-за­щит­ной зо­ны и в зо­не влия­ния предприятия, на тер­ри­то­рии (про­из­вод­ст­вен­ной пло­щад­ке), на ра­бо­чих местах с целью оцен­ки влия­ния про­из­вод­ст­ва на сре­ду оби­та­ния че­ло­ве­ка и его здоровье;
* сы­рья, по­лу­фаб­ри­ка­тов, го­то­вой про­дук­ции и тех­но­ло­гий их производ­ст­ва, хра­не­ния, транс­пор­ти­ров­ки, реа­ли­за­ции и ути­ли­за­ции;
* тех­но­ло­ги­че­ско­го обо­ру­до­ва­ния, про­из­вод­ст­вен­но­го ин­вен­та­ря, тары;
* про­цес­са са­ни­тар­ной об­ра­бот­ки на эта­пах про­из­вод­ст­ва;
* пе­ре­чень долж­но­стей ра­бот­ни­ков, под­ле­жа­щих ме­ди­цин­ским осмот­рам, про­фес­сио­наль­ной ги­гие­ни­че­ской под­го­тов­ке и ат­те­ста­ции;
* пе­ре­чень осу­ще­ст­в­ляе­мых юри­ди­че­ским ли­цом, ин­ди­ви­ду­аль­ным пред­при­ни­ма­те­лем ра­бот и ус­луг, вы­пус­кае­мой про­дук­ции (то­ва­ров);
* ме­ро­прия­тия, пре­ду­смат­ри­ваю­щие обос­но­ва­ние безо­пас­но­сти для здо­ро­вья и жиз­ни че­ло­ве­ка и сре­ды его оби­та­ния, вы­пус­кае­мой про­дук­ции и тех­но­ло­гии ее про­из­вод­ст­ва, кри­те­ри­ев безо­пас­но­сти и (или) без­вред­но­сти фак­то­ров про­из­вод­ст­ва, и раз­ра­бот­ку ме­то­дов кон­тро­ля, в том числе при поступлении сы­рья, хра­не­нии, транс­пор­ти­ров­ке, реа­ли­за­ции и ути­ли­за­ции про­дук­ции и от­хо­дов про­из­вод­ст­ва, а так­же про­цес­сов вы­пол­не­ния ра­бот, ока­за­ния ус­луг;
* пе­ре­чень форм уче­та и от­чет­но­сти, ус­та­нов­лен­ной дей­ст­вую­щим за­ко­но­да­тель­ст­вом по во­про­сам осу­ще­ст­в­ле­ния про­из­вод­ст­вен­но­го контроля;
* пе­ре­чень воз­мож­ных ава­рий­ных си­туа­ций, свя­зан­ных с ос­та­нов­кой про­из­вод­ст­ва, на­ру­ше­ния­ми тех­но­ло­ги­че­ских про­цес­сов, иных соз­даю­щих уг­ро­зу са­ни­тар­но-эпи­де­ми­че­ско­му бла­го­по­лу­чию на­се­ле­ния си­туа­ций, при воз­ник­но­ве­нии ко­то­рых осу­ще­ст­в­ля­ет­ся ин­фор­ми­ро­ва­ние ме­ст­ных исполни­тель­ных и рас­по­ря­ди­тель­ных ор­га­нов, ор­га­нов и уч­ре­ж­де­ний государ­ст­вен­но­го са­ни­тар­но­го над­зо­ра, на­се­ле­ния;
* дру­гие ме­ро­прия­тия, про­ве­де­ние ко­то­рых не­об­хо­ди­мо для осуществ­ле­ния эф­фек­тив­но­го кон­тро­ля за со­блю­де­ни­ем тре­бо­ва­ний санитар­ных норм, пра­вил и ги­гие­ни­че­ских нор­ма­ти­вов, вы­пол­не­ни­ем санитар­но-про­ти­во­эпи­де­ми­че­ских и про­фи­лак­ти­че­ских ме­ро­прия­тий.

3.2.3 Общественный контроль за состоянием охраны труда на объекте при проектируемой деятельности

Согласно пункта 3 Указа Президента Республики Беларусь от 16 октября 2009 г. № 510 «О совершенствовании контрольной (надзорной) деятельности в Республике Беларусь» (далее – Указ № 510) общественный контроль в форме проведения проверок вправе осуществлять только профсоюзы, их организационные структуры, объединения таких союзов и их организационные структуры в случаях и порядке, установленных законодательными актами.

Осуществление общественного контроля в форме проведения проверок другими организациями, а также физическими лицами не предусмотрено.

В развитие Указа № 510 принят Указ Президента Республики Беларусь от 6 мая 2010 г. № 240 «Об осуществлении общественного контроля профессиональными союзами» (далее – Указ № 240).

Виды общественного контроля, право на осуществление которого предоставлено профсоюзам, установлены Законом Республики Беларусь от 22 апреля 1992 года № 1605-ХП «О профессиональных союзах» (в редакции от 14 января 2011 года № 371-3).

Данным Законом, в частности, установлено, что профсоюзы вправе осуществлять общественный контроль за:

* соблюдением законодательства Республики Беларусь об охране труда в порядке, установленном законодательством;
* выполнением коллективного договора (соглашения);
* деятельностью организаций здравоохранения в области охраны здоровья;
* соблюдением нанимателем, собственником или уполномоченным им органом управления законодательства Республики Беларусь о труде;
* соблюдением нанимателем, собственником или уполномоченным им органом управления законодательства Республики Беларусь о профсоюзах.

Указом № 240 профсоюзам предоставлено право на осуществление указанных выше и иных видов общественного контроля в форме мероприятий по наблюдению, анализу, оценке соблюдения требований законодательства по обеспечению трудовых и социально-экономических прав граждан (мониторинг), участия в работе коллегиальных органов, комиссий и иных предусмотренных законодательством, коллективными договорами (соглашениями) формах.

Па основе типовой инструкции в организациях с учетом характера их деятельности разрабатываются локальные нормативные правовые акты, регламентирующие проведение контроля за соблюдением законодательства об охране труда. Полномочия и ответственность руководителей и специалистов организации по осуществлению контроля за соблюдением законодательства об охране труда определяются их должностными инструкциями. Руководители и специалисты организаций осуществляют контроль за соблюдением актов законодательства о труде и охране труда, локальных нормативных правовых актов по охране труда: выполнением работниками функциональных обязанностей по охране труда: выполнением    постановлений, приказов, предписаний органов государственного надзора и контроля, вышестоящей организации, представлений технической инспекции труда профсоюза: мероприятий по улучшению условий и охраны труда, предусмотренных программами, планами, коллективными договорами, планами мероприятий по охране труда и т. д.

Периодический контроль за соблюдением законодательства об охране труда осуществляется руководителями структурных подразделений (производственных участков, цехов и т. д.) с участием общественных инспекторов профсоюза по охране труда. При отсутствии в организации профсоюза, с согласия руководителя организации для участия в проведении периодического контроля, могут привлекаться уполномоченные. Периодический контроль, осуществляемый руководителями структурных подразделений, проводится:

– ежедневно на участке в смене, лаборатории и иных аналогичных структурных подразделениях организации;

– ежемесячно в цехе, отделе, иных аналогичных структурных подразделениях организации: ежеквартально – в организации в целом (ежедневный контроль за состоянием охраны труда осуществляется руководителем структурного подразделения с участием общественного инспектора профсоюза по охране труда (уполномоченного лица но охране труда работников) в начале рабочего дня (смены).

Ежедневно проверяются:

– состояние территории, проходов, проездов;

– состояние и правильность организации рабочих мест;

– безопасность технологического оборудования, оснастки и инструмента, грузоподъемных и транспортных средств;

– наличие и правильное использование работающими средств индивидуальной защиты;

– соблюдение работниками требований безопасности при выполнении работ и т. д.

Результаты проверки с конкретными мероприятиями по устранению выявленных недостатков и нарушений записываются в журнал ежедневного контроля за состоянием охраны труда, установленной формы.

Ежемесячный контроль проводится начальником цеха с участием общественного инспектора профсоюза по охране труда (уполномоченного лица по охране труда работников), руководителей служб цеха и представителей службы охраны труда (инженера по охране труда).

3.2.4 Ответственность за необеспечение трудоохранных требований

За нарушения работниками законодательства о труде и правил охраны труда установлены следующие виды ответственности: дисциплинарная, материальная, административная и уголовная.

Дисциплинарная ответственность работников предусмотрена статьей 198 ТК. За совершение дисциплинарного проступка наниматель может применить к работнику следующие меры дисциплинарного взыскания: замечание, выговор, увольнение с работы.

За однократное грубое нарушение требований по охране труда, повлекшее увечье или смерть других работников, работник может быть уволен по инициативе нанимателя в соответствии с пунктом 9 статьи 42 ТК.

Работники отдельных отраслей экономической деятельности за нарушения требований охраны труда несут ответственность в соответствии с уставами (положениями) о дисциплине.

Полное или частичное лишение нарушителя премий, перенос отпуска на другое время и другие меры воздействия не являются мерами дисциплинарной ответственности (они не предусмотрены в статье 198 ТК) и могут применяться одновременно с привлечением его к той или иной ответственности.

Если же в нарушениях законодательства о труде и правил по охране труда имеются признаки состава преступления, виновный независимо от того, наложено ли на него дисциплинарное взыскание, может быть привлечен и к уголовной ответственности.

За причинение в результате нарушений требований по охране труда имущественного ущерба нанимателю работник независимо от привлечения его к дисциплинарной или уголовной ответственности может быть привлечен к материальной ответственности в соответствии со статьями 400-409 ТК.

Административная ответственность физических и юридических лиц установлена Кодексом Республики Беларусь об административных правонарушениях от 21 апреля 2009 года № 194-3, а порядок привлечения к административной ответственности – Процессуально-исполнительным кодексом Республики Беларусь об административных правонарушениях от 20 декабря 2009 года № 194-3.

В специальную главу КоАП (глава 18) выделены правонарушения против безопасности движения и эксплуатации транспорта.

В зависимости от характера правонарушений КоАП предусматриваются различные меры ответственности как для физических, так и для юридических лиц. При этом, в частности, установлено, что минимальный размер штрафа, налагаемого на физическое лицо, не может быть менее одной десятой базовой величины. Минимальный размер штрафа, налагаемого на индивидуального предпринимателя, не может быть менее двух базовых величин, а на юридическое лицо – менее десяти базовых величин.

Максимальный размер штрафа, налагаемого на физическое лицо, не может превышать пятидесяти базовых величин. Максимальный размер штрафа, налагаемого на индивидуального предпринимателя, не может превышать двухсот базовых величин, а на юридическое лицо – тысячи базовых величин при исчислении штрафа в базовых величинах.

Административные взыскания налагаются уполномоченными государственными органами и должностными лицами, которым предоставлено такое право. Основанием для наложения взысканий являются протоколы об административных правонарушениях, которые составляются уполномоченными должностными лицами, указанными в законодательных актах.

В ПИКоАП подробно регламентированы вопросы, связанные со всеми процессуальными действиями при привлечении к административной ответственности, а также установлен порядок исполнения и обжалования принятых решений.

Уголовная ответственность установлена, в частности, следующими статьями Уголовного кодекса Республики Беларусь от 9 июля 1999 года № 275-3:

– статья 199. Нарушение законодательства о труде;

– статья 233. Незаконная предпринимательская деятельность.

В результате расследования несчастных случаев на производстве может возникнуть вопрос об ответственности виновных лиц.

В соответствии с п.85 правил расследования и учета несчастных случаев на производстве и профессиональных заболеваний, утвержденных Постановлением Совета Министров Республики Беларусь от 15.01.2015 № 30, наниматели, страхователи и должностные лица организаций, допустившие нарушение данных [правил](http://bii.by/tx.dll?d=67995&a=19#a19), привлекаются к ответственности в соответствии с законодательством.

За нарушение правил охраны труда, повлекшее несчастный случай на производстве, для нанимателя, страхователя, должностных лиц организаций могут наступать перечисленные ниже виды ответственности.

Уголовная ответственность за нарушение правил охраны труда возможна при возникновении профессионального заболевания, причинении телесного повреждения либо смерти человека. Уголовная ответственность установлена, в частности, следующими статьями:

– [часть первая](http://bii.by/tx.dll?d=33384&a=2849#a2849) ст.306 Уголовного кодекса Республики Беларусь. Нарушение правил техники безопасности, промышленной санитарии или иных правил охраны труда должностным лицом, ответственным за их соблюдение (нарушение правил охраны труда), повлекшее по неосторожности профессиональное заболевание либо причинение тяжкого или менее тяжкого телесного повреждения. Наказывается штрафом, или исправительными работами на срок до 2 лет, или ограничением свободы на срок до 3 лет, или лишением свободы на тот же срок с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения;

– [часть вторая](http://bii.by/tx.dll?d=33384&a=2850#a2850) ст.306 УК. Нарушение правил охраны труда, повлекшее по неосторожности смерть человека либо причинение тяжкого телесного повреждения двум или более лицам. Наказывается ограничением свободы на срок до 5 лет или лишением свободы на тот же срок с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения;

– [часть третья](http://bii.by/tx.dll?d=33384&a=3891#a3891) ст.306 УК. Нарушение правил охраны труда, повлекшее по неосторожности смерть двух или более лиц. Наказывается лишением свободы на срок от 3 до 7 лет с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения.

Кроме уголовной ответственности за нарушение правил охраны труда УК предусмотрена ответственность за иные преступления, связанные с обеспечением безопасности работников:

– нарушение правил производственно-технической дисциплины или правил безопасности на объектах использования атомной энергии ([ст.301](http://bii.by/tx.dll?d=33384&a=1024#a1024));

– нарушение правил производственно-технической дисциплины, правил безопасности на взрывоопасных предприятиях или во взрывоопасных цехах либо правил безопасности взрывоопасных работ ([ст.302](http://bii.by/tx.dll?d=33384&a=1025#a1025));

– нарушение правил безопасности горных или строительных работ ([ст.303](http://bii.by/tx.dll?d=33384&a=1026#a1026));

– нарушение правил пожарной безопасности ([ст.304](http://bii.by/tx.dll?d=33384&a=1027#a1027));

– нарушение проектов, обязательных для соблюдения требований технических нормативных правовых актов при производстве строительно-монтажных работ ([ст.305](http://bii.by/tx.dll?d=33384&a=1028#a1028));

– непринятие мер по спасению людей ([ст.307](http://bii.by/tx.dll?d=33384&a=1030#a1030));

– несообщение информации об опасности для жизни людей ([ст.308](http://bii.by/tx.dll?d=33384&a=1031#a1031));

– нарушение санитарных норм, правил и гигиенических нормативов ([ст.336](http://bii.by/tx.dll?d=33384&a=4333#a4333));

– служебную халатность ([ст.428](http://bii.by/tx.dll?d=33384&a=2704#a2704)).

Ответственность по [ст. 306](http://bii.by/tx.dll?d=33384&a=1029#a1029) УК несут только должностные лица, на которых в силу их служебных полномочий или по специальному распоряжению непосредственно возложена обязанность обеспечивать соблюдение правил охраны труда на определенном участке работы.

Руководители предприятий, учреждений и организаций, их заместители, главные инженеры, главные специалисты могут быть привлечены к ответственности по данной статье, если они не приняли мер к устранению заведомо известного нарушения правил охраны труда, либо дали указания, противоречащие этим правилам, или, взяв на себя непосредственное руководство отдельными видами работ, не обеспечили соблюдение тех же правил. В иных случаях должностные лица, виновные в ненадлежащем исполнении своих служебных обязанностей по обеспечению безопасных условий труда (например, в непринятии мер по разработке соответствующих инструкций, по созданию условий для выполнения правил охраны труда, в неосуществлении надлежащего контроля за их соблюдением), могут нести ответственность за преступления против интересов службы.

За нарушение правил охраны труда наступает при наличии факта нарушения норм соответствующих нормативных правовых актов, в том числе технических и локальных нормативных правовых актов. Административная ответственность установлена, в частности, следующими статьями:

– [часть 1](http://bii.by/tx.dll?d=61999&a=3959#a3959) ст.9.17 Кодекса Республики Беларусь об административных правонарушениях (далее - КоАП). Нарушение должностным или иным уполномоченным лицом работодателя или индивидуальным предпринимателем требований по охране труда, содержащихся в нормативных правовых актах, в том числе технических и локальных нормативных правовых актах. Штраф в размере от 5 до 40 базовых величин (далее - БВ);

– [часть 2](http://bii.by/tx.dll?d=61999&a=4070#a4070) ст.9.17 КоАП. То же деяние, совершенное повторно в течение 1 года после наложения административного взыскания за такое же нарушение. Штраф в размере от 15 до 50 БВ;

– [часть 4](http://bii.by/tx.dll?d=61999&a=2975#a2975) ст.9.19 КоАП. Иные нарушения законодательства о труде, кроме нарушений, предусмотренных ст.[9.16-9.18](http://bii.by/tx.dll?d=61999&a=105#a105) КоАП и частями [1](http://bii.by/tx.dll?d=61999&a=1333#a1333) и [3](http://bii.by/tx.dll?d=61999&a=1244#a1244) настоящей статьи, причинившие вред работнику. Штраф в размере от 2 до 20 БВ;

– [статья 9.20](http://bii.by/tx.dll?d=61999&a=109#a109) КоАП. Сокрытие страхователем наступления страхового случая при обязательном социальном страховании от несчастных случаев на производстве и профессиональных заболеваний. Штраф в размере от 2 до 10 БВ, на индивидуального предпринимателя - от 10 до 50 БВ, а на юридическое лицо - от 50 до 100 БВ.

С 31 января 2018 г. несообщение или несвоевременное сообщение страхователем о несчастном случае на производстве или профессиональном заболевании в случаях, предусмотренных законодательством, а равно нарушение установленного порядка расследования такого несчастного случая влекут наложение штрафа в размере от 10 до 50 БВ ([ст.9.20](http://bii.by/tx.dll?d=61999&a=4334#a4334) КоАП в редакции Закона Республики Беларусь от 08.01.2018 № 95-З).

Статьи [198-204](http://bii.by/tx.dll?d=33380&a=1748#a1748) Трудового кодекса Республики Беларусь. Противоправное, виновное неисполнение или ненадлежащее исполнение работником своих трудовых обязанностей (дисциплинарный проступок) может повлечь за собой:

– замечание;

– выговор;

– увольнение (пп.[4](http://bii.by/tx.dll?d=33380&a=151#a151), 5, [7-9](http://bii.by/tx.dll?d=33380&a=8500#a8500) ст.42, [п.1](http://bii.by/tx.dll?d=33380&a=2385#a2385) ст.47 ТК).

– лишение полностью или частично дополнительных выплат стимулирующего характера на срок до 12 месяцев ([подп.3.3](http://bii.by/tx.dll?d=292768&a=12#a12) п.3 Декрета Президента Республики Беларусь от 15.12.2014 № 5 «Об усилении требований к руководящим кадрам и работникам организаций», далее - Декрет № 5).

Дисциплинарная ответственность за нарушение правил охраны труда, в том числе повлекшее тяжелые последствия, предусматривается в коллективных договорах, правилах внутреннего трудового распорядка, иных локальных нормативных правовых актах, трудовых договорах (контрактах) работника.

Трудовой договор, заключенный на неопределенный срок, а также срочный трудовой договор до истечения срока его действия может быть расторгнут нанимателем в случае однократного грубого нарушения правил охраны труда, повлекшего увечье или смерть других работников ([п.9](http://bii.by/tx.dll?d=33380&a=8501#a8501) ст.42 ТК).

В целях укрепления общественной безопасности и дисциплины [подп.1.4](http://bii.by/tx.dll?d=69084&a=20#a20) п.1 Директивы Президента Республики Беларусь от 11.03.2014 № 1 «О мерах по укреплению общественной безопасности и дисциплины» требует от руководителей государственных органов, иных организаций независимо от форм собственности обеспечить безусловное привлечение работников организаций к дисциплинарной ответственности вплоть до увольнения за нарушение требований по охране труда, повлекшее увечье или смерть других работников.

Одним из основных критериев оценки выполнения [Директивы](http://bii.by/tx.dll?d=69084&a=15#a15) № 1 следует считать обеспечение здоровых и безопасных условий труда, промышленной, пожарной, ядерной и радиационной безопасности, безопасности движения и эксплуатации транспорта, формирование правопослушного поведения, здорового образа жизни, навыков по обеспечению личной и имущественной безопасности граждан, в том числе в подчиненных (расположенных на подведомственной территории) органах и организациях.

Если в результате несчастного случая на производстве застрахованному лицу был причинен материальный или моральный вред, то виновные в его причинении лица могут нести ответственность действующего законодательства.

Если в результате нарушения требований охраны труда, несчастного случая на производстве нанимателю, страхователю, организации был причинен материальный ущерб, то работник независимо от привлечения его к уголовной, административной или дисциплинарной ответственности может быть привлечен к материальной ответственности согласно ст.[400-409](http://bii.by/tx.dll?d=33380&a=3236#a3236) ТК.

[Декрет](http://bii.by/tx.dll?d=292768&a=8#a8) № 5 изменяет правила удержания из заработной платы, производимые по инициативе нанимателя ([ст.408](http://bii.by/tx.dll?d=33380&a=3555#a3555) ТК).

Так, руководители организаций вправе удерживать из заработной платы работника по распоряжению нанимателя ущерб, причиненный нанимателю по вине работника, в размере до трех его среднемесячных заработных плат. При этом при каждой выплате заработной платы размер такого удержания (при взыскании сумм, в том числе по исполнительным документам, общий размер всех удержаний) не может превышать 50 % заработной платы, причитающейся к выплате работнику, если возможность большего размера удержания (общего размера всех удержаний) не установлена законодательными актами ([подп.3.6](http://bii.by/tx.dll?d=292768&a=15#a15)п.3 Декрета № 5).

Подпункт 3.5 п.3 Декрета № 5 предоставляет право руководителям организаций расторгать трудовой договор (контракт) с работником, допустившим нарушение производственно-технологической, исполнительской или трудовой дисциплины, повлекшее причинение организации ущерба в размере, превышающем три начисленные среднемесячные заработные платы работников Республики Беларусь. При этом указанное действие (бездействие) работника признается грубым нарушением трудовых обязанностей, а увольнение по данному основанию производится с одновременным уведомлением (в день увольнения) соответствующего профсоюза.

# **ЗАКЛЮЧЕНИЕ**

В наше время информационные технологии проникают во все сферы деятельности человека. В связи с этим возросло число преступлений, направленных против информационной безопасности. На сегодняшний день массовое применение компьютеров оказалось под угрозой самовоспроизводящихся программ-вирусов, нарушающих нормальную работу компьютера, портящих файловую структуру дисков и наносящих ущерб хранимой информации на компьютере. Это требует от пользователя персонального компьютера знаний о природе вирусов, способах заражения вирусами и защиты от них. Из раза в раз вирусы становятся все более коварными, что приводит к постоянному изменению профилю угроз. Но и разработчики антивирусных программ не стоят на месте.

Антивирусные программы постоянно совершенствуются. Это означает, что пользователь должен находится в поиске более эффективной системы защиты персонального компьютера.

Цель дипломного проекта достигнута – изучены особенности разработки программных средств защиты от файловых вирусов, разработано программное средство. Все задачи выполнены в полном объеме, определен объект и предмет исследования, актуальность, цели и задачи дипломного проекта; изучена актуальность разработки и использования антивирусных средств; их виды, структура и требования к разработке; описаны основные сведения, входные и выходные данные, разработан графический интерфейс, его структура и диаграммы «Диаграмма вариантов использования», «Диаграмма последовательности действий» и «Диаграмма компонентов»; описаны рекомендации по использованию программного средства; произведен расчет себестоимости и отпускной цены программного средства; изучена тема по охране труда «Организация общественного контроля за состоянием охраны труда» с вопросами: xарактеристика трудовой деятельности и факторов среды при работе оператора; организация общественного контроля условий и безопасности труда; общественный контроль за состоянием охраны труда на объекте при проектируемой деятельности; ответственность за необеспечение трудоохранных требований.

Программное средство разработано на языке C# с использованием языка C++ для разработки драйверов операционной системы. Тестирование разрабатываемого программного средства показало работоспособность, целостность и структурированность. Программное средство имеет модульную структуру, что обеспечивает возможность быстрого освоения использования, сопровождения, а также доработку при необходимости.

В результате работы было получено программное средство защиты от файловых вирусов.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1 Бусел И.П., Малихтарович П.И. Экономика сельского хозяйства. Учебное пособие. –Минск: РИПО 2014.

2 Бусел И.П., Малихтарович П.И. Экономика сельскохозяйственного предприятия с основами менеджмента. –Минск: Лiтаратура i Мастацтва, 2009.

3 Боров, С.А. .NET: Учебный курс / С.А. Боров. – СПб.: Питер, 2017. – 228 с.

4 Воробьев, Е.С. Многопоточное программирование в примерах / Е.С.  Воробьев. – СПб.: Питер, 2016. – 123 с.

5 Бьянкуцци, Ф.А. Пионеры программирования. Диалоги с создателями наиболее популярных языков программирования / Ф.А. Бьянкуцци, Ш.  Уорден. – СПб.: Питер, 2018. – 608 c.

6 Гофман, В.Э. MySql. Быстрый старт / В.Э. Гофман. – СПб.: Петербург, 2018. – 288 с.

7 Гармаза, А.К. Охрана труда: учеб. пособие для студентов высших учебных заведений по специальностям лесного профиля / А.К. Гармаза, И.Т.  Ермак, Б.Р. Ладик. – Минск: БГТУ, 2010. – 366 с.

8 Кэнту, В.М. TransactSQL: Для профессионалов / В.М. Кэнту. – СПб.: Питер, 2016. – 1101 с.

9 Культин, Н.Б. Основы фильтрации низкоуровневых запросов операционной системы / Н.Б. Культин – СПб.: Питер, 2019. – 464 c.

10 Керниган, Б.Д. Создание вирусных баз для начинающих / Б.Д.  Керниган, Д.М. Ритчи. – СПб.: Питер, 2016. – 288 c.

11 Маркин, А.В. Сокеты для профессионалов / А.В. Маркин. – СПб.: Питер, 2019. – 144 c.

12 Мансуров, К.Т. Основы работы с Windows API / К.Т.  Мансуров. – СПб.: Питер, 2015. – 773 с.

13 Осипов, Д.Е. Профессиональное программирование / Д.Е. Осипов – СПб.: Питер, 2017. – 322 c.

14 Окулов, С.М. Практическое пособие разработчика / С.М. Окулов. – СПб.: Питер, 2018. – 120 c.

15 Павловская, Т.А. Программирование на языке высокого уровня / Т.А.  Павловская. – СПб.: Питер, 2018. – 393 с.

16 Ушаков, Д.П. Справочник для мастеров / Д.П. Ушаков. – СПб.: Питер, 2017. – 320 с.

17 Харитонов, С.С. C++: Исследуем низкоуровневое программирование / С.С.  Харитонов.  – СПб.: Лидер, 2019. – 640 с.

18 Челноков, А.А. Охрана труда: учебное пособие для студентов технологических специальностей вузов / А.А. Челноков, И.Н. Жмыхов, В.Н. Цап; под общ. ред. А.А. Челнокова. – Минск: Лидер, 2011. – 671 с.

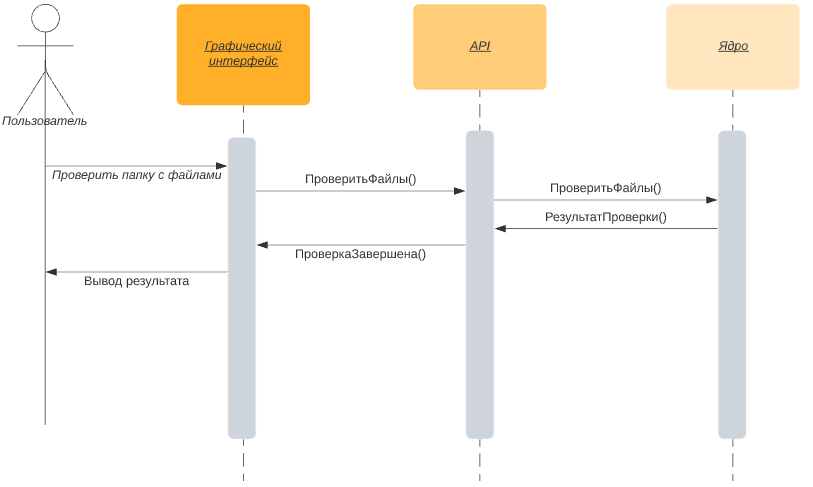
19 Шохирев, М.В. Лучшие решения для бизнеса на C# / М.В.  Шохирев.  –  СПб.: Питер, 2015. – 279 c.

**ПРИЛОЖЕНИЯ**



**ПРИЛОЖЕНИЕ А**

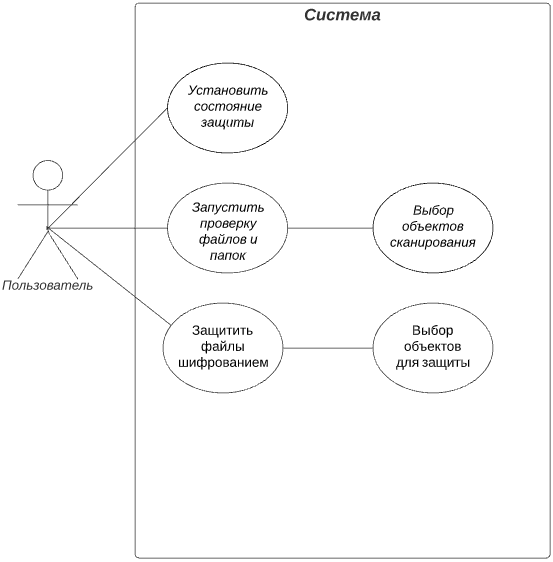
**ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ ДЕЙСТВИЙ**





**ПРИЛОЖЕНИЕ Б**

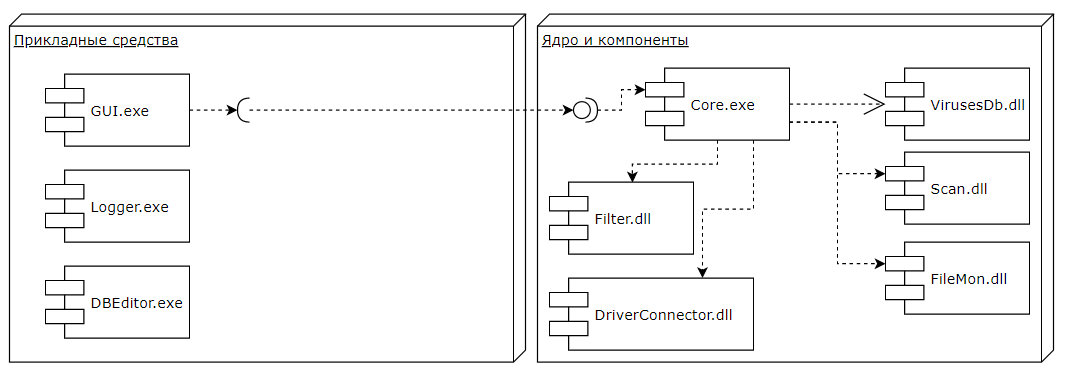
**ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ**





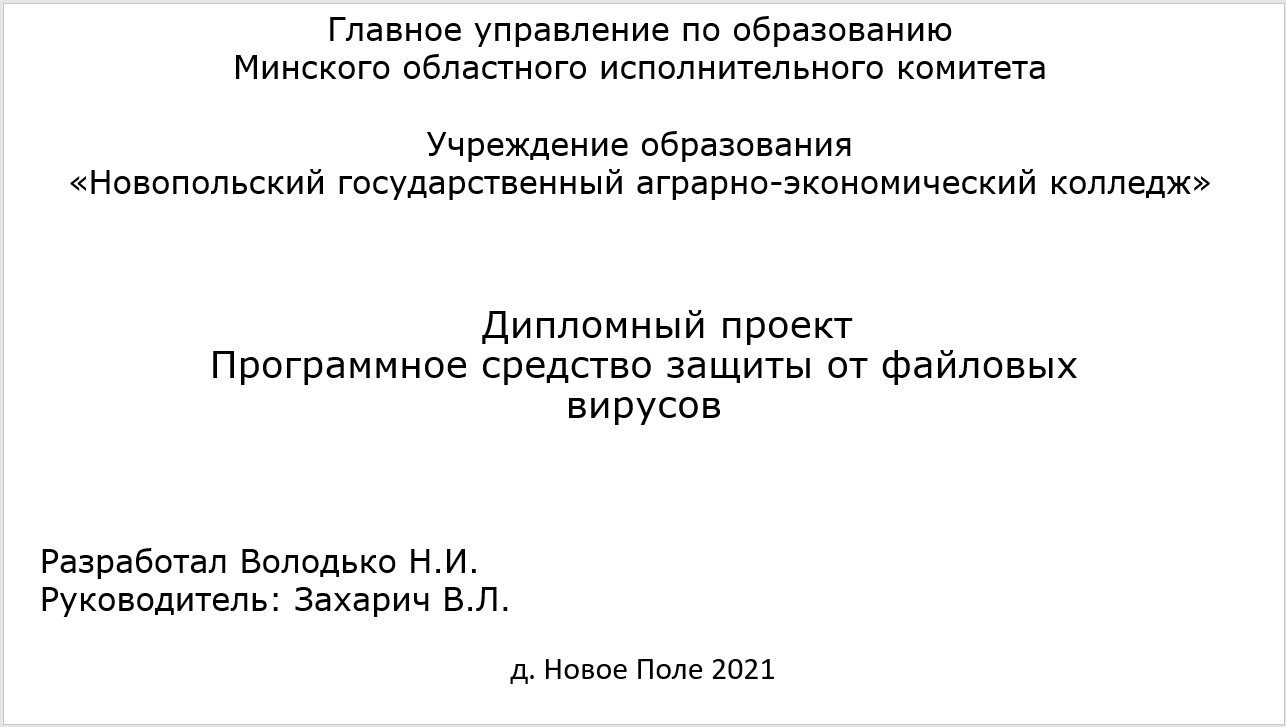
**ПРИЛОЖЕНИЕ В**

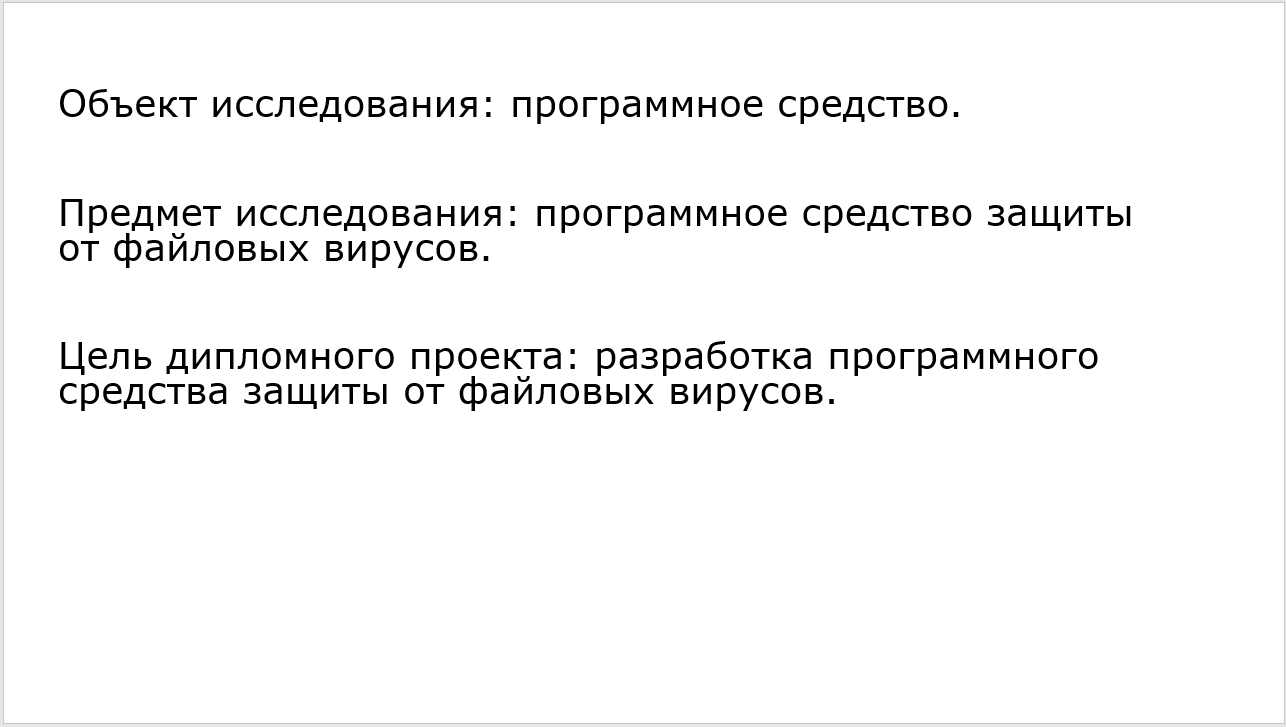
**ДИАГРАММА КОМПОНЕНТОВ**





**ПРИЛОЖЕНИЕ Г**





**ПРИЛОЖЕНИЕ Д**

**ЛИСТИНГ**

using System;

using System.IO;

using System.Text;

using System.Threading;

using System.Diagnostics;

using System.Threading.Tasks;

using Core.Kernel.Configurations;

using Core.Kernel.VirusesManager;

using Core.Kernel.Cryptographer;

using Core.Kernel.ModuleLoader;

using Core.Kernel.ScanModule;

using Core.Kernel.Quarantine;

using Core.Kernel.Connectors;

using Core.Kernel.API;

namespace Core

{

static class KernelInitializator

{

#if DEBUG

private static Process LoggerProc;

#endif

private static Process GUI\_Proc;

public static Configuration Config;

/// <summary>

/// Инициализация конфигурации ядра

/// </summary>

static void InitKernelConfiguration()

{

Config = new Configuration("SystemConf.xml", "UserConf.xml");

}

/// <summary>

/// Инициализация внутренних компонентов ядра

/// ВАЖНО! Менеджер обнаруженных вирусов должен быть запущен раньше чем менеджер задач сканирования

/// </summary>

static void InitKernelComponents()

{

API.Init();

ScannerResponseHandler.Init();

FoundVirusesManager.Init();

ScanTasks.Init();

Quarantine.InitStorage();

}

/// <summary>

/// Инициализация подключаемых(DLL) модулей

/// </summary>

static void InitModules()

{

foreach (string FileName in Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Modules\\", "\*.dll"))

{

string File = FileName.Substring(FileName.LastIndexOf('\\') + 1, FileName.Length - FileName.LastIndexOf('\\') - 1);

ModuleManager.Loader.LoadModule(File);

}

#if DEBUG

KernelConnectors.Logger.WriteLine("[Kernel.initModules] Проверка таблицы сервисов");

foreach (ModuleManager.Module m in ModuleManager.Modules)

{

KernelConnectors.Logger.WriteLine($"[Kernel.initModules] Модуль {m.ModuleName}, статус модуля {m.IsRunning}");

}

#endif

}

/// <summary>

/// Применить базовые настройки модулей

/// </summary>

private static void ApplyingBasicSettings()

{

//Partition monitor

new Task(() =>

{

var drives = DriveInfo.GetDrives();

var SystemDrive = Path.GetPathRoot(Environment.GetFolderPath(Environment.SpecialFolder.System));

for (byte index = 0; index < drives.Length; index++)

{

KernelConnectors.Logger.WriteLine($"[Kernel.ApplyingBasicSettings] Found drive {drives[index].Name}");

if (drives[index].DriveType == DriveType.Removable ||

drives[index].DriveType == DriveType.CDRom ||

drives[index].DriveType == DriveType.Network ||

drives[index].DriveType == DriveType.Unknown ||

drives[index].Name == SystemDrive)

{

//Если диск не подходит

continue;

}

KernelConnectors.Logger.WriteLine($"[Kernel.ApplyingBasicSettings] Create api mon for {drives[index].Name}");

KernelConnectors.PartitionMon\_CommandWriter.Write($"0\*{drives[index].Name}&\*.\*");

KernelConnectors.PartitionMon\_CommandWriter.Flush();

 KernelConnectors.Logger.WriteLine($"[Kernel.ApplyingBasicSettings] Create api mon WRITED");

}

}).Start();

//VirusesDb

new Task(() =>

{

Thread.Sleep(200);

//Выгрузить все сигнатуры в менеджер сканера

KernelConnectors.VirusesDb\_CommandWriter.Write("/upload\_to\_scanner");

KernelConnectors.VirusesDb\_CommandWriter.Flush();

}).Start();

}

/// <summary>

/// Точка входа в ядро

/// </summary>

/// <param name="args"></param>

static void Main(string[] args)

{

Console.CancelKeyPress += Console\_CancelKeyPress;

AppDomain.CurrentDomain.ProcessExit += OnCloseProcess;

#if DEBUG

LoggerProc = Process.Start("Loggers\\Logger.exe");

KernelConnectors.Logger.Init();

#endif

//Инициализация конфигурации ядра

InitKernelConfiguration();

//Инициализация DLL модулей (ввод их в состояние готовности подключится к ядру)

InitModules();

//Инициализация входящих подключений

KernelConnectors.InitInputConnections();

//Инициализация исходящих подключений

KernelConnectors.InitOutputConnections();

//Инициализация компонентов ядра

InitKernelComponents();

//Базовая настройка модулей

ApplyingBasicSettings();

if (Config.GUI\_Autostart)

{

if (File.Exists("GUI\\GUI.exe"))

{

KernelConnectors.Logger.WriteLine("[Kernel] Запуск GUI");

GUI\_Proc = Process.Start("GUI\\GUI.exe");

}

else

{

KernelConnectors.Logger.WriteLine("[Kernel] Файл GUI не найден");

}

}

//await Task.Delay(-1);

}

private static void OnCloseProcess(object sender, EventArgs e)

{

#if DEBUG

if (!LoggerProc.HasExited)

{

LoggerProc.Kill();

}

#endif

}

private static void Console\_CancelKeyPress(object sender, ConsoleCancelEventArgs e)

{

#if DEBUG

if (!LoggerProc.HasExited)

{

LoggerProc.Kill();

}

#endif

if (!GUI\_Proc.HasExited)

{

GUI\_Proc.Kill();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.IO.Pipes;

using System.Threading;

using System.Threading.Tasks;

using Core.Kernel.Connectors;

using Core.Kernel.ScanModule;

using Core.Kernel.Quarantine;

using Core.Kernel.VirusesManager;

using Core.Kernel.ErrorTasks;

using Core.Kernel.ModuleLoader;

using System.Diagnostics;

namespace Core.Kernel.API

{

static class API

{

private static Thread RequestHandler = new Thread(Handler);

private static NamedPipeClientStream UserOutputConnector;

private static NamedPipeServerStream UserInputConnector;

private static BinaryWriter Out\_writer;

private static void Handler()

{

var binaryReader = new BinaryReader(UserInputConnector);

Out\_writer = new BinaryWriter(UserOutputConnector);

#if DEBUG

Console.WriteLine("[API] Init success");

#endif

byte code = 255;

while (true)

{

KernelConnectors.Api\_In\_Sync.WaitOne();

{

try

{

code = binaryReader.ReadByte();

}

catch

{

#if DEBUG

Console.WriteLine("[API] Чтение завершилось ошибкой");

#endif

if (!UserInputConnector.IsConnected)

{

#if DEBUG

Console.WriteLine("[API] Ожидание переподключения пользователя");

#endif

try

{

UserInputConnector.WaitForConnection();

}

catch(Exception ex)

{

#if DEBUG

Console.WriteLine($"[API] {ex.Message}");

#endif

break;

}



if (!UserOutputConnector.IsConnected)

{

#if DEBUG

Console.WriteLine("[API] Переподключение к пользователю");

#endif

UserOutputConnector.Connect();

}

KernelConnectors.Api\_In\_Sync.ReleaseMutex();

continue;

}

}

#if DEBUG

Console.WriteLine($"[API] Запрос, код {code}");

#endif

switch (code)

{

//Запрос на

case 0:

{

break;

}

//Запрос на перемещение файла в карантин

case 1:

{

Task.Run(() =>

{

var id = binaryReader.ReadInt32();

ToQuarantine(id);

});

break;

}

//Запрос на восстановление файла из карантина

case 2:

{

Task.Run(() =>

{

var id = binaryReader.ReadInt32();

Restore(id);

});

break;

}

//Запрос на удаление файла

case 3:

{

Task.Run(() =>

{

var id = binaryReader.ReadInt32();

Delete(id);

});

break;

}

//Запрос информации о вирусе

case 4:

{

Task.Run(() =>

{

var id = binaryReader.ReadInt32();

getVirusInfo(id);

});

break;

}

//Запрос информации о всех вирусах

case 5:

{

Task.Run(() =>

{

getAllVirusesInfo();

});

break;

}

//Просканировать файл

case 6:

 {

string file = binaryReader.ReadString();

#if DEBUG

Console.WriteLine("[API] Добавление задачи " + file);

#endif

Task.Run(() =>

{

if (ScanTasks.Add(file) is null)

{

#if DEBUG

Console.WriteLine("[API] Ошибка добавления задачи, проверка завершена");

#endif

API\_ScanCompleted(0, false, 0, file);

}

}

);

break;

}

//Очистить очередь сканирования

case 7:

{

KernelConnectors.ScannerService\_Command\_Sync.WaitOne();

{

ScanTasks.ClearQueue();

}

KernelConnectors.ScannerService\_Command\_Sync.ReleaseMutex();

break;

}

//Включение/выключение автоматической проверки съемных носителей

case 8:

{

var flag = binaryReader.ReadBoolean();

if (flag)

{

KernelConnectors.PartitionMon\_CommandWriter.Write("2\*");

}

else

{

KernelConnectors.PartitionMon\_CommandWriter.Write("3\*");

}

KernelConnectors.PartitionMon\_CommandWriter.Flush();

break;

}

//Очистить информацию о подключенных устройствах

case 9:

{

KernelConnectors.PartitionMon\_CommandWriter.Write("4\*");

KernelConnectors.PartitionMon\_CommandWriter.Flush();

break;

}

//Добавить простое правило фильтрации

case 10:

{

string rule = binaryReader.ReadString();

KernelConnectors.Filter\_CommandWriter.Write((byte)3);

KernelConnectors.Filter\_CommandWriter.Write(rule);

KernelConnectors.Filter\_CommandWriter.Flush();

 break;

}

//Удалить простое правило фильтрации

case 11:

{

string rule = binaryReader.ReadString();

KernelConnectors.Filter\_CommandWriter.Write((byte)4);

KernelConnectors.Filter\_CommandWriter.Write(rule);

KernelConnectors.Filter\_CommandWriter.Flush();

break;

}

//Удалить все простые правила фильтрации

case 12:

{

KernelConnectors.Filter\_CommandWriter.Write((byte)5);

KernelConnectors.Filter\_CommandWriter.Flush();

break;

}

//Отключить всё

case 13:

{

#if DEBUG

Console.WriteLine("[API] Выключение ядра");

#endif

ScanTasks.Stop();

{

#if DEBUG

Console.WriteLine("[API] Выключение фильтра ");

#endif

KernelConnectors.Filter\_CommandWriter.Write((byte)6);

#if DEBUG

Console.WriteLine("[API] Выключение монитора разделов ");

#endif

KernelConnectors.PartitionMon\_CommandWriter.Write("7\*");

#if DEBUG

Console.WriteLine("[API] Выключение сканнера ");

#endif

KernelConnectors.ScannerService\_CommandWriter.Write((byte)1);

#if DEBUG

Console.WriteLine("[API] Выключение вирусной БД");

#endif

KernelConnectors.VirusesDb\_CommandWriter.Write("/shutdown");

}

#if DEBUG

Console.WriteLine("[API] Закрытие подключений");

#endif

KernelConnectors.Stop();

Task.Run(() => {

Thread.Sleep(1000);

#if DEBUG

Console.WriteLine("[API] Отключение обработчика API");

#endif

RequestHandler.Abort();

#if DEBUG

Console.WriteLine("[API] Закрытие процесса ядра");

#endif

Process.GetCurrentProcess().Kill();

} );

break;

}

case 14:

{

Defender(binaryReader.ReadBoolean());

break;

}

default:

{

#if DEBUG

Console.WriteLine("[API] Unknown request");

#endif

break;

}

}

}

KernelConnectors.Api\_In\_Sync.ReleaseMutex();

}

}

private static void API\_ScanCompleted(int id, bool found, int virusId, string file)

{

if (!UserOutputConnector.IsConnected)

{

return;

}

KernelConnectors.Api\_Out\_Sync.WaitOne();

{

if (found)

{

//Идентификатор

Out\_writer.Write((byte)1);

Out\_writer.Write(id);

Out\_writer.Write(file);

Out\_writer.Write(virusId);

}

else

{

Out\_writer.Write((byte)0);

Out\_writer.Write(file);

}

Out\_writer.Flush();

}

KernelConnectors.Api\_Out\_Sync.ReleaseMutex();

}

/// <summary>

/// Переместить файл в карантин

/// </summary>

/// <param name="id"></param>

private static void ToQuarantine(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

Quarantine.Quarantine.MoveVirusToQuarantine(id);

}

/// <summary>

/// Восстановить файл

/// </summary>

/// <param name="id"></param>

private static void Restore(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

ScanTasks.RestoredFile = virusInfo.file;

KernelConnectors.Logger.WriteLine("[API.Restore] Восстановление файла " + virusInfo.file, LoggerLib.LogLevel.OK);

KernelConnectors.Logger.WriteLine("[API .Restore] Удаление информации о вирусе из менеджера", LoggerLib.LogLevel.OK);

FoundVirusesManager.Delete(id);

KernelConnectors.Logger.WriteLine("[API.Restore] Вызов восстановления файла у менеджера карантина", LoggerLib.LogLevel.OK);

Quarantine.Quarantine.Restore(virusInfo.fileInQuarantine, virusInfo.file);

}

/// <summary>

/// Удалить файл, где бы он не находился(в карантине/на диске)

/// </summary>

/// <param name="id"></param>

private static void Delete(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

if (virusInfo.inQuarantine)

{

KernelConnectors.Logger.WriteLine($"[API] Удаление файла из карантина {virusInfo.fileInQuarantine}", LoggerLib.LogLevel.OK);

Quarantine.Quarantine.DeleteFromStorage(id);

}

else

{

KernelConnectors.Logger.WriteLine($"[API] Удаление файла на жестком диске {virusInfo.file}", LoggerLib.LogLevel.OK);

File.Delete(virusInfo.file);

}

KernelConnectors.Logger.WriteLine($"[API] Удаление из менеджера вирусов", LoggerLib.LogLevel.OK);

FoundVirusesManager.Delete(id);

}



private static void getVirusInfo(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

if(virusInfo == null)

{

return;

}

KernelConnectors.Api\_Out\_Sync.WaitOne();

{

Out\_writer.Write((byte)2);

Out\_writer.Write(virusInfo.id);

Out\_writer.Write(virusInfo.file);

Out\_writer.Write(virusInfo.VirusId);

Out\_writer.Write(virusInfo.inQuarantine);

Out\_writer.Write(virusInfo.fileInQuarantine is null ? " " : virusInfo.fileInQuarantine);

Out\_writer.Flush();

}

KernelConnectors.Api\_Out\_Sync.ReleaseMutex();

}

private static void getAllVirusesInfo()

{

KernelConnectors.Api\_Out\_Sync.WaitOne();

{

foreach(VirusInfo virusInfo in FoundVirusesManager.getAllViruses())

{

Out\_writer.Write((byte)2);

Out\_writer.Write(virusInfo.id);

Out\_writer.Write(virusInfo.file);

Out\_writer.Write(virusInfo.VirusId);

Out\_writer.Write(virusInfo.inQuarantine);

Out\_writer.Write(virusInfo.fileInQuarantine is null ? " " : virusInfo.fileInQuarantine);

Out\_writer.Flush();

}

}

KernelConnectors.Api\_Out\_Sync.ReleaseMutex();

}

private static void Defender(bool flag)

{

KernelConnectors.PartitionMon\_CommandPipe\_Sync.WaitOne();

if (flag)

{

KernelConnectors.PartitionMon\_CommandWriter.Write("5\*");

}

else

{

KernelConnectors.PartitionMon\_CommandWriter.Write("6\*");

}

KernelConnectors.PartitionMon\_CommandPipe\_Sync.ReleaseMutex();

}

public static void Init()

{

UserOutputConnector = KernelConnectors.Api\_Out;

UserInputConnector = KernelConnectors.Api\_In;

ScannerResponseHandler.onScanCompleted += API\_ScanCompleted;

RequestHandler.Start();

}

}

} using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Core.Kernel.Connectors;

using System.IO;

using System.Xml;

using System.Xml.Serialization;

namespace Core.Kernel.Configurations

{

[Serializable]

public class Configuration

{

/// <summary>

/// Кодировка именованных каналов(труб)

/// </summary>

public Encoding NamedPipeEncoding { get; private set; }

/// <summary>

/// Автозапуск GUI

/// </summary>

public bool GUI\_Autostart { get; private set; }

private FileStream SystemConftFile;

private XmlDocument SystemXmlFile = new XmlDocument();

public Configuration()

{

}

public Configuration(string pathToSystemConf, string pathToUserConf)

{

/\*Check conf files\*/

{

var defaultConf = GetDefaultConfiguration();

if (!File.Exists(pathToSystemConf))

{

KernelConnectors.Logger.WriteLine("[Configuration] CREATE SYSTEM CONF", LoggerLib.LogLevel.WARN);



var file = new StreamWriter(File.Create(pathToSystemConf));

file.WriteLine(

"<?xml version=\"1.0\"?>"

+ "<sysConf>\n" +

$"<PipeEncode>{defaultConf.NamedPipeEncoding}</PipeEncode>\n" +

$"<guiautostart>true</guiautostart>\n" +

"</sysConf>\n"

);

file.Close();

}

}

KernelConnectors.Logger.WriteLine("[Configuration] LOAD SYSTEM CONF", LoggerLib.LogLevel.WARN);

{

this.SystemConftFile = new FileStream(pathToSystemConf, FileMode.OpenOrCreate);

SystemXmlFile.Load(this.SystemConftFile);

var root = SystemXmlFile.DocumentElement;

/\* Named pipe encoding \*/

{

var XmlEncode = getElementValueByName("PipeEncode", root);

{

if (XmlEncode != null)

{

switch (XmlEncode.InnerText)

{

case "System.Text.UTF8Encoding":

{

NamedPipeEncoding = Encoding.UTF8;

break;

}

case "System.Text.UnicodeEncoding":

{

NamedPipeEncoding = Encoding.Unicode;

break;

}

case "System.Text.UTF32Encoding":

{

NamedPipeEncoding = Encoding.UTF32;

break;

}

}

}

}

}

/\*\*/

{

var XmlEncode = getElementValueByName("guiautostart", root);

if(XmlEncode != null)

{

this.GUI\_Autostart = bool.Parse(XmlEncode.InnerText);

}

}

}

KernelConnectors.Logger.WriteLine("[Configuration] Конфигурация загружена", LoggerLib.LogLevel.OK);

}

/// <summary>

/// Найти XML элемент по имени среди потомков

/// </summary>

private static XmlElement getElementValueByName(string name, XmlElement element)

{

foreach(XmlElement child in element.ChildNodes)

{

if(child.Name == name)

{

return child;

}

}

return null;

}

/// <summary>

/// Загрузить стандартную конфигурацию

/// </summary>

/// <returns></returns>

public static Configuration GetDefaultConfiguration()

{

return new Configuration

{

NamedPipeEncoding = Encoding.Unicode

};

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.IO;

using System.IO.Pipes;

using Core.Kernel;

using Core.Kernel.Configurations;

using LoggerLib;

namespace Core.Kernel.Connectors

{

static class KernelConnectors

{

/\* OUTPUT CONNECTORS \*/

public static NamedPipeClientStream PartitionMon\_CommandPipe = new NamedPipeClientStream("PartitionMon.Command");

public static BinaryWriter PartitionMon\_CommandWriter;

public static Mutex PartitionMon\_CommandPipe\_Sync = new Mutex();

public static NamedPipeClientStream VirusesDb\_CommandPipe = new NamedPipeClientStream("VirusesDb.CommandPipe");

public static BinaryWriter VirusesDb\_CommandWriter;

public static Mutex VirusesDb\_CommandPipe\_Sync = new Mutex();

public static NamedPipeClientStream ScannerService\_Output = new NamedPipeClientStream("ScannerService.Input");

public static BinaryWriter ScannerService\_Writer;

public static Mutex ScannerService\_Output\_Sync = new Mutex();

public static NamedPipeClientStream ScannerService\_Command = new NamedPipeClientStream("Scanner.CommandPipe");

public static BinaryWriter ScannerService\_CommandWriter;

public static Mutex ScannerService\_Command\_Sync = new Mutex();

public static NamedPipeClientStream Filter\_Command = new NamedPipeClientStream("Filter.CommandPipe");

public static BinaryWriter Filter\_CommandWriter;

public static Mutex Filter\_Command\_Sync = new Mutex();

/\* INPUT CONNECTORS \*/

public static NamedPipeServerStream Filter\_Input = new NamedPipeServerStream("Filter.Output");

 public static BinaryReader Filter\_Reader;

public static Mutex Filter\_Input\_Sync = new Mutex();

public static NamedPipeServerStream ScannerService\_Input = new NamedPipeServerStream("ScannerService.Output");

public static BinaryReader ScannerService\_Reader;

public static Mutex ScannerService\_Input\_Sync = new Mutex();

/\*For API\*/

public static NamedPipeServerStream Api\_In = new NamedPipeServerStream("API.Core");

public static NamedPipeClientStream Api\_Out = new NamedPipeClientStream("API.User");

public static Mutex Api\_In\_Sync = new Mutex();

public static Mutex Api\_Out\_Sync = new Mutex();

/\*For logs\*/

public static LoggerClient Logger = new LoggerClient("Logger.Kernel", "Kernel log");

public static void InitInputConnections()

{

Logger.WriteLine($"[Kernel.Connectors] Wait input connections", LogLevel.WARN);

Task.Run(() =>

{

Filter\_Input\_Sync.WaitOne();

{

Filter\_Input.WaitForConnection();

Filter\_Reader = new BinaryReader(Filter\_Input, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] Filter connected", LogLevel.OK);

}

Filter\_Input\_Sync.ReleaseMutex();

});

Task.Run(() =>

{

ScannerService\_Input\_Sync.WaitOne();

{

ScannerService\_Input.WaitForConnection();

ScannerService\_Reader = new BinaryReader(ScannerService\_Input, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] Scanner INPUT connected", LogLevel.OK);

}

ScannerService\_Input\_Sync.ReleaseMutex();

});

Task.Run(() =>

{

Api\_In\_Sync.WaitOne();

{

Api\_In.WaitForConnection();

Logger.WriteLine($"[Kernel.Connectors] API INPUT connected", LogLevel.OK);

}

Api\_In\_Sync.ReleaseMutex();

});

Task.Run(() =>

{

Filter\_Command\_Sync.WaitOne();

{

Filter\_Command.Connect();

Filter\_CommandWriter = new BinaryWriter(Filter\_Command, KernelInitializator.Config.NamedPipeEncoding);

}

 Filter\_Command\_Sync.ReleaseMutex();

});

}

public static void InitOutputConnections()

{

Logger.WriteLine($"[Kernel.Connectors.InitOutputConnections] Wait output connections");

Task.Run(() =>

{

PartitionMon\_CommandPipe\_Sync.WaitOne();

{

PartitionMon\_CommandPipe.Connect();

PartitionMon\_CommandWriter = new BinaryWriter(PartitionMon\_CommandPipe, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] Монитор разделов подключен", LogLevel.OK);

}

PartitionMon\_CommandPipe\_Sync.ReleaseMutex();

});

Task.Run(() =>

{

ScannerService\_Output\_Sync.WaitOne();

{

ScannerService\_Output.Connect();

ScannerService\_Writer = new BinaryWriter(ScannerService\_Output, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] Сканнер подключен(на вход сканнера)", LogLevel.OK);

}

ScannerService\_Output\_Sync.ReleaseMutex();

ScannerService\_Command\_Sync.WaitOne();

{

ScannerService\_Command.Connect();

ScannerService\_CommandWriter = new BinaryWriter(ScannerService\_Command, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] Канал команд сканера подключен", LogLevel.OK);

}

ScannerService\_Command\_Sync.ReleaseMutex();

});

Task.Run(() =>

{

VirusesDb\_CommandPipe\_Sync.WaitOne();

{

VirusesDb\_CommandPipe.Connect();

VirusesDb\_CommandWriter = new BinaryWriter(VirusesDb\_CommandPipe, KernelInitializator.Config.NamedPipeEncoding);

Logger.WriteLine($"[Kernel.Connectors] База вирусов подключена", LogLevel.OK);

}

VirusesDb\_CommandPipe\_Sync.ReleaseMutex();

Thread.Sleep(100);

});



Task.Run(() =>

{

Api\_Out\_Sync.WaitOne();

{

Api\_Out.Connect();

Logger.WriteLine($"[Kernel.Connectors] API OUT connected", LogLevel.OK);

}

Api\_Out\_Sync.ReleaseMutex();

});

}

public static void Stop()

{

VirusesDb\_CommandPipe.Close();

ScannerService\_Command.Close();

ScannerService\_Output.Close();

ScannerService\_Input.Close();

Filter\_Command.Close();

Filter\_Input.Close();

PartitionMon\_CommandPipe.Close();

}

}

} using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.IO;

using System.IO.IsolatedStorage;

using System.Xml;

using System.Xml.Serialization;

using Core.Kernel.Connectors;

namespace Core.Kernel.VirusesManager

{

/// <summary>

/// Класс отвечающий за найденные вирусы

/// </summary>

public static class FoundVirusesManager

{

private static List<VirusInfo> VirusesTable = new List<VirusInfo>();

public static Mutex VirusesTable\_sync = new Mutex();

private static IsolatedStorageFile Storage;

private static XmlSerializer Serializer;

/// <summary>

/// Добавить новый вирус в таблицу

/// </summary>

public static void AddNewVirus(VirusInfo info)

{

VirusesTable\_sync.WaitOne();

{

VirusesTable.Add(info);

}

VirusesTable\_sync.ReleaseMutex();

}

public static VirusInfo getInfo(int id)

{

VirusInfo result = null;

VirusesTable\_sync.WaitOne();

{

for (int index = 0; index < VirusesTable.Count; index++)

{

if (VirusesTable[index].id == id)

{

result = VirusesTable[index];

break;

}

}

}

VirusesTable\_sync.ReleaseMutex();

return result;

}

/// <summary>

/// Проверка существования такого файла в таблице обнаруженных вирусов

/// </summary>

 public static bool Exists(string file)

{

bool result = false;

VirusesTable\_sync.WaitOne();

{

for (int index = 0; index < VirusesTable.Count; index++)

{

if (VirusesTable[index].file == file)

{

result = true;

break;

}

}

}

VirusesTable\_sync.ReleaseMutex();

return result;

}

public static VirusInfo[] getAllViruses()

{

return VirusesTable.ToArray();

}

public static int getLastId()

{

return VirusesTable.Count > 0 ? VirusesTable[VirusesTable.Count - 1].id : 0;

}

/// <summary>

/// Удалить запись о вирусе

/// </summary>

/// <returns></returns>

public static bool Delete(int id)

{

VirusesTable\_sync.WaitOne();

{

for (int index = 0; index < VirusesTable.Count; index++)

{

if (VirusesTable[index].id == id)

{

VirusesTable.RemoveAt(index);

VirusesTable\_sync.ReleaseMutex();

return true;

}

}

}

VirusesTable\_sync.ReleaseMutex();

return false;

}

/// <summary>

/// Инициализация компонента

/// </summary>

public static void Init()

{

KernelConnectors.Logger.WriteLine("[FoundVirusesManager] Загрузка данных из изолированного хранилища");

Storage = IsolatedStorageFile.GetUserStoreForDomain();

var file = Storage.OpenFile("viruses.xml", FileMode.OpenOrCreate);

if(file.Length < 5)

{

KernelConnectors.Logger.WriteLine($"[FoundVirusesManager] Файл данных о вирусах пуст");

file.Close();

return;

}

Serializer = new XmlSerializer(typeof(VirusInfo[]));

VirusInfo[] viruses = (VirusInfo[])Serializer.Deserialize(file);

foreach(VirusInfo info in viruses)

{

VirusesTable.Add(info);

}

file.Close();

KernelConnectors.Logger.WriteLine($"[FoundVirusesManager] Загружено {viruses.Length} записей");

}

 }

[Serializable]

public class VirusInfo

{

public int id;

public bool inQuarantine; // Находится ли файл в карантине

public string fileInQuarantine; // Путь к файлу в карантине

public string file; // Путь к файлу на жестком диске

public int VirusId;

public VirusInfo()

{

}

public VirusInfo(int id, string file, int VirusId)

{

this.id = id;

this.file = file;

this.VirusId = VirusId;

this.inQuarantine = false;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using Core.Kernel.Connectors;

using Core;

namespace Core.Kernel.ModuleLoader

{

public static class ModuleManager

{

public static List<Module> Modules = new List<Module>();

public static class Loader

{

public static void LoadModule(string ModuleFileName)

{

Modules.Add(new Module(ModuleFileName));

}

}

/// <summary>

/// Представляет собой модуль, а так же загрузчик

/// </summary>

public class Module

{

public readonly string ModuleName;

public readonly Assembly ModuleAssembly;

public bool IsRunning { private set; get; }

public Module(string ModuleFileName)

{

this.IsRunning = false;

this.ModuleName = ModuleFileName;

try

{

this.ModuleAssembly = Assembly.LoadFrom("Modules\\" + ModuleFileName);

}

catch

{

KernelConnectors.Logger.WriteLine($"[ModuleManager.LoadModule] Ошибка загрузки сборки модуля {ModuleFileName}", LoggerLib.LogLevel.ERROR);

return;

}

{

//Проверка существования класса инициализатора

bool found = false;

foreach (Type type in this.ModuleAssembly.GetTypes())

{

if (type.Name == "Initializator")

 {

found = true;

break;

}

}

if (!found)

{

KernelConnectors.Logger.WriteLine($"[Kernel.ModuleLoader] {ModuleFileName} Инициализатор не найден", LoggerLib.LogLevel.ERROR);

//Не найден класс инициализатора

return;

}

}

//Вход в модуль

{

Type InitializatorType = this.ModuleAssembly.GetType(ModuleFileName.Substring(0, ModuleFileName.Length - 3) + "Initializator", true, true);

MethodInfo EntryPoint = InitializatorType.GetMethod("EntryPoint");

if (EntryPoint == null)

{

KernelConnectors.Logger.WriteLine($"[Kernel.ModuleLoader] {ModuleFileName} точка входа в модуль не найдена", LoggerLib.LogLevel.ERROR);

return;

}

else

{

object result = EntryPoint.Invoke(null, new object[] { });

if ((byte)result == 0)

{

this.IsRunning = true;

}

}

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.IO.IsolatedStorage;

using System.IO.Compression;

using System.IO.MemoryMappedFiles;

using Core.Kernel.VirusesManager;

using Vlingo.UUID;

using Core.Kernel.Connectors;

namespace Core.Kernel.Quarantine

{

static class Utils

{

/// <summary>

/// Не найденные файлы

/// </summary>

public static List<string> NotExistsFiles = new List<string>();

}

static class Quarantine

{

private static IsolatedStorageFile VirusStorage;

/// <summary>

/// Поместить файл в карантин(защищенное хранилище)

/// </summary>

/// <param name="pathToFile"></param>

/// <returns></returns>

static public AddToStorageResult AddFileToStorage(string pathToFile)

{

if (!File.Exists(pathToFile))

{

Utils.NotExistsFiles.Add(pathToFile);

return new AddToStorageResult(false, "Target file not exists");

}



string FileName = pathToFile.Substring(pathToFile.LastIndexOf('\\')+1);

KernelConnectors.Logger.WriteLine($"[Quarantine] Create file({FileName}) in isolated storage ", LoggerLib.LogLevel.WARN);

IsolatedStorageFileStream storageFile;

FileStream targetFile;

try

{

storageFile = VirusStorage.CreateFile($"VirusFiles\\{FileName}");

targetFile = File.Open(pathToFile, FileMode.Open);

}

catch (Exception ex)

{

return new AddToStorageResult(false, ex.Message);

}

byte[] buffer = new byte[2048];

while (targetFile.Read(buffer, 0, buffer.Length) > 0)

{

storageFile.Write(buffer, 0, buffer.Length);

}

storageFile.Close();

targetFile.Close();

KernelConnectors.Logger.WriteLine("[Quarantine] Delete original file >" + pathToFile, LoggerLib.LogLevel.WARN);

File.Delete(pathToFile);

return new AddToStorageResult(true, $"VirusFiles\\{FileName}");

}

/// <summary>

/// Восстановить файл из карантина

/// </summary>

/// <param name="id"></param>

static public void Restore(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

KernelConnectors.Logger.WriteLine("[Quarantine] Восстановление файла из " + virusInfo.fileInQuarantine + " в " + virusInfo.file);

var CreatedFileStream = File.Create(virusInfo.file);

var targetFileStream = VirusStorage.OpenFile(virusInfo.fileInQuarantine, FileMode.Open);

byte[] buffer = new byte[2048];

while (targetFileStream.Read(buffer, 0, buffer.Length) > 0)

{

CreatedFileStream.Write(buffer, 0, buffer.Length);

}

CreatedFileStream.Close();

targetFileStream.Close();

}

static public void Restore(string from, string to)

{

KernelConnectors.Logger.WriteLine("[Quarantine] Восстановление файла из " + from + " в " + to);

var CreatedFileStream = File.Create(to);

var targetFileStream = VirusStorage.OpenFile(from, FileMode.Open);

byte[] buffer = new byte[2048];

while (targetFileStream.Read(buffer, 0, buffer.Length) > 0)

{

 CreatedFileStream.Write(buffer, 0, buffer.Length);

}

CreatedFileStream.Close();

targetFileStream.Close();

}

/// <summary>

/// Удалить файл в защищенном хранилище

/// </summary>

/// <param name="path">Путь к файлу в хранилище</param>

static public void Delete(string path)

{

if (VirusStorage.FileExists(path))

{

VirusStorage.DeleteFile(path);

}

}

/// <summary>

/// Получить пути ко всем файлам в карантине

/// </summary>

static public string[] GetAllFiles()

{

return VirusStorage.GetFileNames("VirusFiles\\");

}

/// <summary>

/// Переместить вирус в карантин по его id

/// </summary>

static public void MoveVirusToQuarantine(int id)

{

var virusInfo = FoundVirusesManager.getInfo(id);

if (virusInfo != null && virusInfo.inQuarantine == false) {

var result = AddFileToStorage(virusInfo.file);

if (result.is\_success)

{

virusInfo.inQuarantine = true;

virusInfo.fileInQuarantine = result.fileName;

KernelConnectors.Logger.WriteLine("=== [Quarantine] ===");

KernelConnectors.Logger.WriteLine("ПЕРЕМЕЩЕНИЕ В КАРАНТИН УСПЕШНО", LoggerLib.LogLevel.OK);

KernelConnectors.Logger.WriteLine(" Идентификатор задачи" + virusInfo.id);

KernelConnectors.Logger.WriteLine(" Идентификатор вируса" + virusInfo.VirusId);

KernelConnectors.Logger.WriteLine(" Файл " + virusInfo.file);

KernelConnectors.Logger.WriteLine(" Путь к файлу в карантине " + virusInfo.fileInQuarantine);

KernelConnectors.Logger.WriteLine("=== [Quarantine] ===");

}

else

{

KernelConnectors.Logger.WriteLine("[Quarantine] Ошибка добавления файла в карантин");

}

}

}

static public bool DeleteFromStorage(int id)

{

var info = FoundVirusesManager.getInfo(id);

if (VirusStorage.FileExists(info.fileInQuarantine))

{

VirusStorage.DeleteFile(info.fileInQuarantine);

return true;

}

 return false;

}

/// <summary>

/// Инициализация хранилища

/// </summary>

static public bool InitStorage()

{

try

{

VirusStorage = IsolatedStorageFile.GetUserStoreForDomain();

if (!VirusStorage.DirectoryExists("VirusFiles"))

{

KernelConnectors.Logger.WriteLine("[Quarantine] Создана директория в изолированном хранилище");

VirusStorage.CreateDirectory("VirusFiles");

}

return true;

}

catch

{

return false;

}

}

/// <summary>

/// Результат добавления в изолированное хранилище

/// </summary>

public class AddToStorageResult

{

public bool is\_success;

public string fileName;

public AddToStorageResult(bool success, string FileName)

{

this.is\_success = success;

this.fileName = FileName;

}

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.IO.Pipes;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Diagnostics;

using Core.Kernel.Connectors;

using Core.Kernel.Configurations;

using Core.Kernel.VirusesManager;

using Core.Kernel.ErrorTasks;

namespace Core.Kernel.ScanModule

{

/// <summary>

/// Обработчик ответов с результатами сканирования от сервиса скана

/// </summary>

static class ScannerResponseHandler

{

static readonly TimeSpan ErrorScanProbeSleep = TimeSpan.FromSeconds(1);

public delegate void ScanCompletedEvent(int id, bool found, int virusId, string file);

public static event ScanCompletedEvent onScanCompleted;

private static NamedPipeServerStream Connector = Connectors.KernelConnectors.ScannerService\_Input;

public static Thread InputThreadHandler = new Thread(InputHandler);

/// <summary>

/// Максимальное количество попыток для скана

/// </summary>

public const int MAX\_PROBES = 3;

private static void InputHandler()

{

 KernelConnectors.Logger.WriteLine("[ScannerResponseHandler.InputHandler] Started");

while (true)

{

KernelConnectors.ScannerService\_Input\_Sync.WaitOne();

{

int id = KernelConnectors.ScannerService\_Reader.ReadInt32();

byte result = KernelConnectors.ScannerService\_Reader.ReadByte();

int virusId = KernelConnectors.ScannerService\_Reader.ReadInt32();

var task = ScanTasks.getTaskById(id);

if (task is null)

{

KernelConnectors.Logger.WriteLine($"[ScannerResponseHandler.InputHandler] Задача сканирования {id} не найдена", LoggerLib.LogLevel.ERROR);

continue;

}

if (result == 2)

{

KernelConnectors.Logger.WriteLine($"[ScannerResponseHandler.InputHandler] Задача сканирования {id} не была выполнена, новая попытка {task.ProbesCount}", LoggerLib.LogLevel.ERROR);

if (task.ProbesCount == MAX\_PROBES)

{

KernelConnectors.Logger.WriteLine($"[ScannerResponseHandler.InputHandler] Задача сканирования {id} не была выполнена спустя несколько попыток", LoggerLib.LogLevel.ERROR);

ErrorScanTasksManager.Add(1, "Message", task);

onScanCompleted.Invoke(id, false, 0, task.File);

}

else

{

task.ProbesCount++;

new Task(() =>

{

Thread.Sleep(ErrorScanProbeSleep);

ScanTasks.RestartScan(id);

}).Start();

}

continue;

}

onScanCompleted.Invoke(id, result == 1, virusId, task.File);

}

KernelConnectors.ScannerService\_Input\_Sync.ReleaseMutex();

}

}

public static void Init()

{

InputThreadHandler.Start();

}

public static void Stop()

{

InputThreadHandler.Abort();

}

}

/// <summary>

/// Обработчик подключаемого фильтра

/// </summary>

static class FilterHandler

{

/// <summary>

/// Обработчик обнаруженных файлов от фильтра

/// </summary>

private static readonly Thread FilterMonitor = new Thread(Handler);



public static void Handler()

{

KernelConnectors.Logger.WriteLine("[FileQueue] [Thr.Monitor] Запуск, ожидание мьютекса... ");

KernelConnectors.Filter\_Input\_Sync.WaitOne();

{

if (KernelConnectors.Filter\_Input.IsConnected)

{

KernelConnectors.Logger.WriteLine("[FileQueue] [Thr.Monitor] Подключен ");

}

}

KernelConnectors.Filter\_Input\_Sync.ReleaseMutex();

while (true)

{

string commandBuffer = string.Empty;

try

{

commandBuffer = KernelConnectors.Filter\_Reader.ReadString();

}

catch

{

KernelConnectors.Logger.WriteLine("[FileQueue] [Thr.Monitor] Отключаю поток ", LoggerLib.LogLevel.WARN);

break;

}

if (FoundVirusesManager.Exists(commandBuffer.Substring(1)))

{

//Если файл уже числится у нас как вирус

continue;

}

switch (commandBuffer[0])

{

case '1':

{

KernelConnectors.Logger.WriteLine("[FileQueue] [Thr.Monitor] Создание файла -> " + commandBuffer);

ScanTasks.Add(commandBuffer.Substring(1));

break;

}

case '4':

{

KernelConnectors.Logger.WriteLine("[FileQueue] [Thr.Monitor] Изменение файла -> " + commandBuffer);

ScanTasks.Add(commandBuffer.Substring(1));

break;

}

}

}

}

public static void Run()

{

FilterMonitor.Start();

}

public static void Stop()

{

FilterMonitor.Abort();

}

}

/// <summary>

/// Менеджер задач сканирования

/// </summary>

public static class ScanTasks

{

public static int id = 0;



public static NamedPipeClientStream Scanner\_Output;

public static BinaryWriter ScannerBinaryWriter;

private static List<ScanTask> tasks = new List<ScanTask>();

public static Mutex tasks\_sync = new Mutex();

/// <summary>

/// Восстановленный файлы после карантина

/// </summary>

public static string RestoredFile = "\*ab12\*s.ts.bak.dek.dec.ts";

public static ScanTask Add(string file)

{

if (file.Contains(RestoredFile))

{

KernelConnectors.Logger.WriteLine($"[ScanQueue] Пропускаю добавление задачи скана файла восстановленного из карантина!", LoggerLib.LogLevel.OK);

RestoredFile = "\*ab12\*s.ts.bak.dek.dec.ts";

return null;

}

ScanTask task = null;

tasks\_sync.WaitOne();

{

task = new ScanTask(file, id);

tasks.Add(task);

ScannerBinaryWriter.Write(id);

ScannerBinaryWriter.Write(file);

ScannerBinaryWriter.Flush();

id++;

}

tasks\_sync.ReleaseMutex();

return task;

}

/// <summary>

/// Повторить сканирование

/// </summary>

/// <param name="taskId"></param>

public static void RestartScan(int taskId)

{

tasks\_sync.WaitOne();

{

var task = getTaskById(taskId);

if(task is null)

{

tasks\_sync.ReleaseMutex();

return;

}

ScannerBinaryWriter.Write(taskId);

ScannerBinaryWriter.Write(task.File);

ScannerBinaryWriter.Flush();

}

tasks\_sync.ReleaseMutex();

}

/// <summary>

/// Удалить задачу по айди

/// </summary>

public static void RemoveById(int id)

{

tasks\_sync.WaitOne();

{

for (int taskIndex = 0; taskIndex < tasks.Count; taskIndex++)

{

if (tasks[taskIndex].TaskId == id)

{

tasks.RemoveAt(taskIndex);

break;

}

}

}

tasks\_sync.ReleaseMutex();

}

/// <summary>

/// Удалить по полному пути к файлу

/// </summary>

public static void RemoveByFileName(string file)

{

 tasks\_sync.WaitOne();

for(int index = 0; index < tasks.Count; index++)

{

if(tasks[index].File == file)

{

tasks.RemoveAt(index);

break;

}

}

tasks\_sync.ReleaseMutex();

}

/// <summary>

/// Получить экземпляр задачи

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

public static ScanTask getTaskById(int id)

{

tasks\_sync.WaitOne();

for (int index = 0; index < tasks.Count; index++)

{

if (tasks[index].TaskId == id)

{

var task = tasks[index];

tasks\_sync.ReleaseMutex();

return task;

}

}

tasks\_sync.ReleaseMutex();

return null;

}

/// <summary>

/// Извлечь задачу и удалить из листа задач

/// </summary>

public static ScanTask getTaskAndRemove(int id)

{

tasks\_sync.WaitOne();

{

for (int index = 0; index < tasks.Count; index++)

{

if (tasks[index].TaskId == id)

{

var task = tasks[index];

tasks.RemoveAt(index);

tasks\_sync.ReleaseMutex();

return task;

}

}

}

tasks\_sync.ReleaseMutex();

return null;

}

/// <summary>

/// Очистить локальные задачи и задачи в сервисе сканирования

/// </summary>

public static void ClearQueue()

{

tasks\_sync.WaitOne();

{

tasks.Clear();

{

KernelConnectors.ScannerService\_CommandWriter.Write((byte)0);

KernelConnectors.ScannerService\_CommandWriter.Flush();

}

}

tasks\_sync.ReleaseMutex();

}

public static void ScanCompleted(int id, bool found, int virusId, string file)

{

if (found)

{

var task = getTaskAndRemove(id);

if (task != null)

{

FoundVirusesManager.AddNewVirus(

new VirusInfo(

id,

 task.File,

virusId

)

);

KernelConnectors.Logger.WriteLine($"[ScanQueue] Вирус найден {id}!", LoggerLib.LogLevel.OK);

KernelConnectors.Logger.WriteLine($"[ScanQueue] Restored = {RestoredFile}!", LoggerLib.LogLevel.OK);

}

else

{

KernelConnectors.Logger.WriteLine("[ScanQueue] КРИТИЧЕСКАЯ ОШИБКА, НЕ НАЙДЕНА ЗАДАЧА ОБНАРУЖЕННОГО ВИРУСА!!!", LoggerLib.LogLevel.ERROR);

}

}

else

{

KernelConnectors.Logger.WriteLine($"[ScanQueue] Не вирус {id}!");

}

RemoveById(id);

}

/// <summary>

/// Инициализация менеджера задач сканирования

/// </summary>

public static void Init()

{

id = FoundVirusesManager.getLastId() + 1;

FilterHandler.Run();

ScannerResponseHandler.onScanCompleted += ScanCompleted;

Scanner\_Output = KernelConnectors.ScannerService\_Output;

ScannerBinaryWriter = new BinaryWriter(Scanner\_Output, KernelInitializator.Config.NamedPipeEncoding);

}

public static void Stop()

{

KernelConnectors.Logger.WriteLine("[ScanQueue.Stop] Остановка обработчика фильтра");

FilterHandler.Stop();

KernelConnectors.Logger.WriteLine("[ScanQueue.Stop] Остановка обработчика ответов от сканнера");

ScannerResponseHandler.Stop();

KernelConnectors.Logger.WriteLine("[ScanQueue.Stop] Очистка очереди сканирования");

ClearQueue();

}

}

/// <summary>

/// Представляет собой задачу сканирования

/// </summary>

public class ScanTask

{

public string File;

public int TaskId;

/// <summary>

/// Количество попыток сканирования

/// </summary>

public byte ProbesCount;

public ScanTask(string file, int id)

{

this.File = file;

this.TaskId = id;

ProbesCount = 0;

 }

}

} using System;

using System.Collections.Generic;

using System.IO;

using System.IO.MemoryMappedFiles;

using System.IO.IsolatedStorage;

using System.IO.Compression;

using System.IO.Pipes;

using System.IO.Ports;

using System.Collections;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using LoggerLib;

using MODULE\_\_SCAN.Yara;

namespace MODULE\_\_SCAN

{

public static class Configuration

{

public static Encoding PipeEncoding = Encoding.Unicode;

public static bool UsingYara = true;

/// <summary>

/// Максимальный размер файла для быстрого сканирования (файл подходящий под такой критерий будет полностью считываться в память)

/// </summary>

public const int MAX\_FAST\_SCAN\_FILE = 1\_000\_000\_000;

public static int THREAD\_COUNT = 30;

public const int SCAN\_THREAD\_SLEEP = 100;

public const string YaraRulesDir = @"Modules\YARA\Rules";

public const SearchOption YaraRulesSearchOption = SearchOption.TopDirectoryOnly;

}

public static class Connector

{

public static NamedPipeServerStream commandPipe = new NamedPipeServerStream("Scanner.CommandPipe");

public static NamedPipeServerStream inputPipe = new NamedPipeServerStream("ScannerService.input");

public static NamedPipeServerStream signaturesPipe = new NamedPipeServerStream("ScannerService.signatures");

public static NamedPipeClientStream outputPipe = new NamedPipeClientStream("ScannerService.output");

private static Thread inputHandler = new Thread(inputThread) { Name = "inputHandler", Priority = ThreadPriority.Highest };

private static Thread signatureHandler = new Thread(signatureThread) { Name = "signatureHandler" };

private static Thread commandHandler = new Thread(commandThread) { Name = "commandHandler" };

private static BinaryWriter outputWriter;

private static BinaryReader commandReader;

public static LoggerClient Logger = new LoggerClient("Logger.Modules.Scanner", "Log");

/// <summary>

/// Записать данные о скане в выходящую трубу

/// </summary>

public static void ToOutput(int id, ScanResult result)

{

outputWriter.Write(id);

outputWriter.Write((byte)result.Result);

 outputWriter.Write(result.VirusID);

outputWriter.Flush();

}

/// <summary>

/// Обработчик трубы по которой приходят пути к файлам для сканирования

/// </summary>

public static void inputThread()

{

Logger.WriteLine("[Scanner.inputThread] ScannerService.input ожидание подключения");

inputPipe.WaitForConnection();

var binaryReader = new BinaryReader(inputPipe, Configuration.PipeEncoding);

Logger.WriteLine("[Scanner.inputThread] ScannerService.input подключен");

while (true)

{

int id = 0;

try

{

id = binaryReader.ReadInt32();

}

catch

{

Logger.WriteLine("[Scanner.inputThread] Отключаю поток", LogLevel.ERROR);

break;

}

string file = binaryReader.ReadString();

Logger.WriteLine($"[Scanner.inputThread] Добавляю задачу сканирования, айди {id}, путь -> {file}");

ScanTasks.Add(id, file);

}

}

public static void signatureThread()

{

Logger.WriteLine("[Scanner.signatureThread] Ожидание подключения");

signaturesPipe.WaitForConnection();

var binaryReader = new BinaryReader(signaturesPipe);

Logger.WriteLine("[Scanner.signatureThread] ScannerService.signatures подключен", LogLevel.OK);

while (true)

{

var ID = 0;

try

{

ID = binaryReader.ReadInt16();

}

catch

{

Logger.WriteLine("[Scanner.signatureThread] Отключаю поток", LogLevel.ERROR);

break;

}

var Signature = binaryReader.ReadBytes(binaryReader.ReadInt16());

if (ID >= Scanner.Signatures.Length)

{

Logger.WriteLine("[Scanner.signatureThread] Записываю сигнатуру в локальный буфер");

Array.Resize(ref Scanner.Signatures, Scanner.Signatures.Length + 1);

Scanner.Signatures[Scanner.Signatures.Length - 1] = new Signature(Signature);

}

else

 {

Logger.WriteLine("[Scanner.signatureThread] Обновление сигнатуры в локальном буффере");

Scanner.Signatures[ID] = new Signature(Signature);

}

}

}

public static void commandThread()

{

Logger.WriteLine("[Scanner.commandThread] Ожидание подключения", LogLevel.WARN);

commandPipe.WaitForConnection();

commandReader = new BinaryReader(commandPipe, Configuration.PipeEncoding);

Logger.WriteLine("[Scanner.commandThread] Подключено", LogLevel.OK);

while (true)

{

var code = commandReader.ReadByte();

switch (code)

{

case 0:

{

Logger.WriteLine("[Scanner.commandThread] Очистка буфера задач сканирования", LogLevel.WARN);

lock(ScanTasks.TaskQueue.SyncRoot)

{

ScanTasks.TaskQueue.Clear();

ScanTasks.ActiveScanTasks = 0;

}

break;

}

case 1:

{

Logger.WriteLine("[Scanner.commandThread] Очистка буфера задач сканирования", LogLevel.WARN);

lock (ScanTasks.TaskQueue.SyncRoot)

{

ScanTasks.TaskQueue.Clear();

ScanTasks.ActiveScanTasks = 0;

}

for (int index = 0; index < ScanTasks.ScanThreads.Length; index++)

{

Logger.WriteLine($"[Scanner.commandThread] Остановка потока сканирования {index}", LogLevel.WARN);

ScanTasks.ScanThreads[index].Abort();

}

Logger.WriteLine("[Scanner.commandThread] Остановка потоков", LogLevel.WARN);

inputHandler.Abort();

signatureHandler.Abort();

Thread.Sleep(100);

Logger.WriteLine("[Scanner.commandThread] Закрытие труб", LogLevel.WARN);

commandPipe.Close();

inputPipe.Close();

commandPipe.Close();

outputPipe.Close();

commandHandler.Abort();

break;

}

}

}

}

/// <summary>

 /// Инициализация коннектора

/// </summary>

public static void Init()

{

#if DEBUG

Logger.Init();

#endif

inputHandler.Start();

signatureHandler.Start();

Logger.WriteLine("[Scanner.Init] Ожидание подключения к outputPipe", LogLevel.WARN);

outputPipe.Connect();

outputWriter = new BinaryWriter(outputPipe);

Logger.WriteLine("[Scanner.Init] outputPipe подключен", LogLevel.OK);

commandHandler.Start();

}

}

public static class Scanner

{

public static Signature[] Signatures = new Signature[0];

public static ScanResult ScanFile(Stream FileStream)

{

if (FileStream.Length <= Configuration.MAX\_FAST\_SCAN\_FILE)

{

ScanResult Result = new ScanResult(0, result.NotVirus);

byte[] FileBuffer = new byte[FileStream.Length];

int readed = 0;

try

{

readed = FileStream.Read(FileBuffer, 0, FileBuffer.Length);

}

catch (Exception ex)

{

Connector.Logger.WriteLine($"[Scanner] Ошибка чтения файла {ex.Message}", LogLevel.ERROR);

return new ScanResult(0, result.Error);

}

if (readed == 0)

{

return Result;

}

Array.Resize(ref FileBuffer, readed);

try

{

Parallel.For(0, Signatures.Length, (int sigIndex, ParallelLoopState state) =>

{

int bufferSumm = 0;

int backOffset = Signatures[sigIndex].SignatureBytes.Length; //Задний оффсет, на каждой итерации делаем summ -= fileBuffer[bufPos - backOffset]

for (int initPos = 0; initPos < Signatures[sigIndex].SignatureBytes.Length; initPos++)

{

bufferSumm += FileBuffer[initPos];

}

for (int bufferPosition = backOffset; bufferPosition < FileBuffer.Length; bufferPosition++)

{

bufferSumm -= FileBuffer[bufferPosition - backOffset];

bufferSumm += FileBuffer[bufferPosition];

//Если сумма совпала, проверяем этот участок

if (bufferSumm == Signatures[sigIndex].Summ)

 {

bool found = true;

for (

int filePos = 1 + bufferPosition - backOffset, signaturePos = 0;

filePos < bufferPosition && signaturePos < Signatures[sigIndex].SignatureBytes.Length;

filePos++, signaturePos++)

{

if (FileBuffer[filePos] != Signatures[sigIndex].SignatureBytes[signaturePos])

{

found = false;

break;

}

}

if (found)

{

Result = new ScanResult(sigIndex, result.Virus);

state.Break();

}

}

}

});

}

catch (Exception ex)

{

Connector.Logger.WriteLine($"[Scanner] Ошибка во время проверки {ex.Message}", LogLevel.ERROR);

}

return Result;

}

else

{

return new ScanResult(0, result.NotVirus);

}

}

}

public static class ScanTasks

{

public static Thread[] ScanThreads;

/// <summary>

/// Очередь задач сканирования

/// </summary>

public static Queue TaskQueue = new Queue();

/// <summary>

/// Количество активных задач сканирования

/// </summary>

public static byte ActiveScanTasks = 0;

/// <summary>

/// Мьютекс для сервиса сканнера

/// </summary>

public static Mutex ScanMutex = new Mutex();

public delegate void ScanComplete(ScanTask task, ScanResult result);

public delegate void ScanStart();

/// <summary>

/// При старте сканирования

/// </summary>

private static void ScanStarted()

{

ActiveScanTasks++;

}

/// <summary>

/// Событие окончания сканирования файла

/// </summary>

private static void ScanCompleted(ScanTask task, ScanResult result)

{

Connector.Logger.WriteLine($"\n\n[SCAN COMPLETE EVENT] {task.file}, result {result.Result}");

{

Connector.ToOutput(task.id, result);

ActiveScanTasks--;

 }

}

/// <summary>

/// Добавить задачу сканирования

/// </summary>

/// <param name="pathToFile"></param>

public static void Add(int id, string pathToFile)

{

//lock (TaskQueue.SyncRoot)

{

TaskQueue.Enqueue(new ScanTask(id, pathToFile, ScanStarted, ScanCompleted));

}

}

public static void ScanThread()

{

while (true)

{

ScanTask task = null;

lock (TaskQueue.SyncRoot)

{

if (TaskQueue.Count > 0)

{

task = (ScanTask)TaskQueue.Dequeue();

}

}

if (task != null)

{

if (Configuration.UsingYara)

{

Connector.Logger.WriteLine($"[ScanThread {Thread.CurrentThread.Name}] Поток начал проверку правил Yara", LogLevel.WARN);

ScanResult result;

if (!File.Exists(task.file))

{

Connector.Logger.WriteLine($"[ScanThread {Thread.CurrentThread.Name}] Файл не найден", LogLevel.ERROR);

result = new ScanResult(0, MODULE\_\_SCAN.result.Error);

}

else

{

try

{

if (YaraIntegration.CheckFile(task.file))

{

result = new ScanResult(0, MODULE\_\_SCAN.result.Virus);

}

else

{

result = new ScanResult(0, MODULE\_\_SCAN.result.NotVirus);

}

}

catch(Exception ex)

{

Connector.Logger.WriteLine($"[ScanThread {Thread.CurrentThread.Name}] Ошибка при проверке файла {ex.Message}", LogLevel.ERROR);

result = new ScanResult(0, MODULE\_\_SCAN.result.Error);

}

}

ScanCompleted(

task,

new ScanResult(0, result.Result)

);

}

else

{

FileStream stream = null;

try

{

stream = File.Open(task.file, FileMode.Open, FileAccess.Read);

}

 catch (PathTooLongException)

{

Connector.Logger.WriteLine($"[SCANNER] PathTooLongException", LogLevel.ERROR);

ScanCompleted(task, new ScanResult(0, MODULE\_\_SCAN.result.NotVirus));

}

catch (UnauthorizedAccessException)

{

Connector.Logger.WriteLine($"[SCANNER] UnauthorizedAccessException", LogLevel.ERROR);

ScanCompleted(task, new ScanResult(0, MODULE\_\_SCAN.result.NotVirus));

}

catch (DirectoryNotFoundException)

{

Connector.Logger.WriteLine($"[SCANNER] DirectoryNotFoundException", LogLevel.ERROR);

ScanCompleted(task, new ScanResult(0, MODULE\_\_SCAN.result.NotVirus));

}

catch (FileNotFoundException)

{

Connector.Logger.WriteLine($"[SCANNER] FileNotFoundException", LogLevel.ERROR);

ScanCompleted(task, new ScanResult(0, MODULE\_\_SCAN.result.NotVirus));

}

catch (Exception ex)

{

Connector.Logger.WriteLine($"[SCANNER] ERROR OPEN FILE {ex.Message}", LogLevel.ERROR);

ScanCompleted(task, new ScanResult(0, MODULE\_\_SCAN.result.Error));

continue;

}

Connector.Logger.WriteLine($"[ScanThread {Thread.CurrentThread.Name}] Поток начал сканирование поиском сигнатуры", LogLevel.WARN);

var result = Scanner.ScanFile(stream);

stream.Close();

ScanCompleted(task, result);

}

}

Thread.Sleep(Configuration.SCAN\_THREAD\_SLEEP);

}

}

/// <summary>

/// Инициализация сервиса

/// </summary>

public static void Init()

{

Configuration.THREAD\_COUNT = Environment.ProcessorCount \* 4;

Connector.Logger.WriteLine($"[Scanner.Init] Запуск {Configuration.THREAD\_COUNT} потоков сканирования");

ScanThreads = new Thread[Configuration.THREAD\_COUNT];

for (int index = 0; index < ScanThreads.Length; index++)

{

ScanThreads[index] = new Thread(ScanThread) { Name = "Scan Thread " + index.ToString(), Priority = ThreadPriority.Lowest };

ScanThreads[index].Start();

}

}

}



public static class Initializator

{

public static byte EntryPoint()

{

new Thread(() => Connector.Init()).Start();

ScanTasks.Init();

YaraIntegration.Init();

return 0;

}

public static void Exit()

{

}

}

public class ScanTask

{

public string file;

public int id;

public ScanTask(int id, string file, ScanTasks.ScanStart onStart, ScanTasks.ScanComplete onCompleted, Task task = null)

{

this.id = id;

this.file = file;

}

}

/// <summary>

/// Содержит информацию о результате сканирования

/// </summary>

public class ScanResult

{

public readonly int VirusID;

public readonly result Result;

public ScanResult(int virusId, result res)

{

this.VirusID = virusId;

this.Result = res;

}

}

/// <summary>

/// Результат сканирования

/// </summary>

public enum result

{

NotVirus,

Virus,

Error

}

/// <summary>

/// Сигнатура

/// </summary>

public struct Signature

{

public readonly byte[] SignatureBytes;

public readonly int Summ;

public Signature(byte[] signature)

{

this.SignatureBytes = signature;

this.Summ = 0;

foreach (byte b in signature)

{

this.Summ += b;

}

}

}

} using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using MODULE\_\_SCAN;

using LoggerLib;

using YaraSharp;

using Alphaleonis;

namespace MODULE\_\_SCAN.Yara

{

static class YaraIntegration

{

public static YSInstance Instance;

public static YSContext Context;

public static YSCompiler Compiler;

public static YSRules Rules;

public static YSReport Errors;

 public static YSReport Warnings;

private static string[] skipExtentions = new string[] {/\* ".db", ".toc", ".sp", ".sb", ".dll"\*/ };

public static void Init()

{

Connector.Logger.WriteLine("[YaraIntegration.Init] Start");

Instance = new YSInstance();

Dictionary<string, object> externals = new Dictionary<string, object>()

{

{ "filename", string.Empty },

{ "filepath", string.Empty },

{ "extension", string.Empty }

};

Connector.Logger.WriteLine("[YaraIntegration.Init] Загрука YARA правил");

List<string> ruleFilenames = Directory.GetFiles(MODULE\_\_SCAN.Configuration.YaraRulesDir, "\*.yar", MODULE\_\_SCAN.Configuration.YaraRulesSearchOption).ToList();

Connector.Logger.WriteLine($"[YaraIntegration.Init] Загружено {ruleFilenames.Count} файлов ");

Context = new YSContext();

Compiler = Instance.CompileFromFiles(ruleFilenames, externals);

Rules = Compiler.GetRules();

Errors = Compiler.GetErrors();

Warnings = Compiler.GetWarnings();

var ErrDump = Errors.Dump();

var WrnDump = Warnings.Dump();

foreach (var key in ErrDump)

{

Connector.Logger.WriteLine($"[YaraIntegration.Init] Error!");

}

foreach (var key in WrnDump)

{

Connector.Logger.WriteLine($"[YaraIntegration.Init] Warning!");

}

Connector.Logger.WriteLine($"[YaraIntegration.Init] Загрузка завершена");

}

public static List<YSMatches> GetCheckResult(string path)

{

foreach (string ext in skipExtentions)

{

if (path.Contains(ext))

{

return new List<YSMatches>();

}

}

List<YSMatches> Matches = new List<YSMatches>();

try

{

Matches = Instance.ScanFile(path, Rules,

new Dictionary<string, object>()

{

{ "filename", Alphaleonis.Win32.Filesystem.Path.GetFileName(path) },

{ "filepath", Alphaleonis.Win32.Filesystem.Path.GetFullPath(path) },

{ "extension", Alphaleonis.Win32.Filesystem.Path.GetExtension(path) }

},

0);

}

catch

 {

}

var matches = new List<YSMatches>();

foreach (YSMatches Match in Matches)

{

if (Match.Rule.Identifier == "UPX")

{

continue;

}

Connector.Logger.WriteLine("[Yara.check] ВНИМАНИЕ! СОВПАДЕНИЕ! ->" + Match.Rule.Identifier, LogLevel.WARN);

matches.Add(Match);

}

return matches;

}

public static bool CheckFile(string path)

{

if (GetCheckResult(path).Count > 0)

{

return true;

}

return false;

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.IO.Pipes;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Management;

using System.Linq;

using LoggerLib;

namespace MODULE\_\_RESERVE\_NEW\_FILE\_DETECTOR

{

public static class Configuration

{

public static string API\_MON\_PIPE\_NAME = "API\_MON\_FILTER";

public static string COMMAND\_PIPE\_NAME = "PartitionMon.Command";

public static bool RemovableAutoScan = false;

//Флаг выключенной защиты

public static bool Disable = false;

public static Encoding NamedPipeEncoding = Encoding.Unicode;

/// <summary>

/// Какие файлы будут проверятся на флешке

/// </summary>

public static string[] RemovableDevicesFilter = new string[]{ "exe", "dll", "msi", "mp4"};

}

public static class Connector

{

/// <summary>

/// Для отправки путей к файлам в модуль фильтра, которые были обнаружены

/// </summary>

public static NamedPipeClientStream FilterInputPipe = new NamedPipeClientStream(Configuration.API\_MON\_PIPE\_NAME);

/// <summary>

/// Для приёма команд

/// </summary>

public static NamedPipeServerStream CommandPipe = new NamedPipeServerStream(Configuration.COMMAND\_PIPE\_NAME);

 /// <summary>

/// Для чтения команд

/// </summary>

public static BinaryReader CommandReader;

/// <summary>

/// Для записи путей обнаруженных файлов

/// </summary>

public static BinaryWriter FilterPipeWriter;

public static Mutex Writer\_Sync = new Mutex();

public static LoggerClient Logger = new LoggerClient("Logger.ApiMonitor", "Api monitor");

public static void Init()

{

#if DEBUG

Logger.Init();

#endif

}

}

/// <summary>

/// Таблица носителей

/// </summary>

static class HardDrives

{

static public List<Drive> DriveTable = new List<Drive>(); //Таблица подключенных устройств

static public List<string> WhiteSerialList = new List<string>(); //Белый лист USB накопителей, тут их сериальные номера, подгружаются с файла

static public byte countConnectedRemovableDevices { get; set; } //Будет помогать определять, отключение/подключение устройств

//Запись(строчка) в таблице

public struct Drive

{

public DriveInfo DriveInf;

public string VolumeLabel;

public string SerialNumber;

public long TotalSize;

public long TotalFreeSpace;

public string FileSystem;

public bool IsConnected;

public string RootDir;

public Drive(DriveInfo drive)

{

this.DriveInf = drive;

this.TotalSize = drive.TotalSize;

this.TotalFreeSpace = drive.TotalFreeSpace;

this.VolumeLabel = drive.VolumeLabel;

this.FileSystem = drive.DriveFormat;

this.RootDir = drive.RootDirectory.FullName;

this.IsConnected = true;

this.SerialNumber = null;

}

/// <summary>

/// Проверка подключены ли эти устройства

/// </summary>

public bool CheckConnect(string[] SerialNumbers)

{

//Определить подключение, через WMI

foreach (string serial in SerialNumbers)

if (serial == this.SerialNumber)

{

this.IsConnected = true;

return true;

}

this.IsConnected = false;

return false;

}

}

/// <summary>

/// Добавить в таблицу информацию о новом устройстве

/// </summary>

 /// <returns>Количество устройств в таблице</returns>

static public byte AddNewDrive(DriveInfo newDrive, string serialNumber)

{

Drive drive = new Drive(newDrive);

drive.SerialNumber = serialNumber;

DriveTable.Add(drive);

return (byte)DriveTable.Count;

}

/// <summary>

/// Есть ли устройство с таким серийником в таблице

/// </summary>

static public bool CheckSerial(string serialNumber)

{

foreach (Drive mDrive in DriveTable)

{

if (serialNumber == mDrive.SerialNumber) return true;

}

return false;

}

/// <summary>

/// Есть ли такое устройство в таблице

/// </summary>

static public bool CheckDrive(DriveInfo drive)

{

foreach (Drive drive1 in DriveTable)

{

if (drive1.VolumeLabel == drive.VolumeLabel &&

drive1.TotalSize == drive.TotalSize &&

drive1.FileSystem == drive.DriveFormat)

{

return true;

}

}

return false;

}

/// <summary>

/// Получить информацию по сериал номеру

/// </summary>

static public Drive getDriveBySerial(string serialNumber)

{

foreach (Drive drive in DriveTable)

{

if (drive.SerialNumber == serialNumber)

{

return drive;

}

}

return DriveTable[0];

}

/// <summary>

/// Проверить, принадлежит ли этому носителю такой сериал номер

/// </summary>

/// <returns>true, если сериал номер равен этому накопителю</returns>

static public bool checkDriveSerial(DriveInfo drive, string serial)

{

foreach (Drive myDrive in DriveTable)

{

if (myDrive.FileSystem == drive.DriveFormat &&

myDrive.TotalSize == drive.TotalSize)

{

return true;

}

}

return false;

}

/// <summary>

/// Обновление таблицы подключенных устройств

/// </summary>

static public void RefreshConnectedDevices(string[] serialNumbers)

 {

//Всем ставить false (все устройства помечены как отключенные)

for (int index = 0; index < DriveTable.Count; index++)

{

Drive newStruct = DriveTable[index];

newStruct.IsConnected = false;

DriveTable[index] = newStruct;

}

//А теперь проверка, какие из устройств в нашей таблице подключены

for (int index = 0; index < DriveTable.Count; index++)

{

foreach (string serial in serialNumbers)

{

//Проверка, есть ли устройство с таким серийником в нашей таблице

if (DriveTable[index].SerialNumber == serial)

{

//Если устройство с таким серийным номером есть в нашей таблице, то выставляем ему флаг CONNECTED

Drive newStruct = DriveTable[index];

newStruct.IsConnected = true;

DriveTable[index] = newStruct;

Connector.Logger.WriteLine($"[RemovableDevicesMon] Обновление таблицы, устройство SER:{serial} было подключено", LogLevel.OK);

break;

}

}

}

}

}

static public class RemovableDeviceMonitor

{

static public Thread ThreadMonitor;

static public void AddRemovableDeviceToScan(string pathToDrive)

{

Console.WriteLine(pathToDrive);

List<FileInfo> files = new List<FileInfo>();

foreach(FileInfo info in new DirectoryInfo(pathToDrive).GetFiles("\*.exe"))

{

files.Add(info);

}

foreach (FileInfo info in new DirectoryInfo(pathToDrive).GetFiles("\*.dll"))

{

files.Add(info);

}

foreach (FileInfo info in new DirectoryInfo(pathToDrive).GetFiles("\*.mp4"))

{

files.Add(info);

}

foreach (FileInfo file in files)

{

Connector.Writer\_Sync.WaitOne();

{

Connector.FilterPipeWriter.Write("1" + file.FullName);

Connector.FilterPipeWriter.Flush();

}

Connector.Writer\_Sync.ReleaseMutex();

}

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Файлы({files.Count}) добавлены в очередь сканирования", LogLevel.INFO);

 }

static private void Worker()

{

Connector.Logger.WriteLine("[FileSysApiMon.RemovableDeviceMonitor] Сервис мониторинга внешних носителей активен", LogLevel.OK);

while (true)

{

ManagementObjectCollection collection = new ManagementObjectSearcher("SELECT \* FROM Win32\_usbhub WHERE Caption=\"Запоминающее устройство для USB\"").Get();

string[] SerialNumbers = new string[collection.Count]; byte index = 0;

foreach (ManagementObject obj in collection)

{

//Выделение серийных номеров подключенных USB устройств

SerialNumbers[index] = obj["DeviceID"].ToString().Trim();

SerialNumbers[index] = SerialNumbers[index].Substring(SerialNumbers[index].LastIndexOf(@"\") + 1, (SerialNumbers[index].Length - SerialNumbers[index].LastIndexOf("\\")) - 1);

}

{

if (collection.Count != HardDrives.countConnectedRemovableDevices)

{

//Если в подключенных устройствах что то изменилось

if (collection.Count > HardDrives.countConnectedRemovableDevices)

{

//Новое ПОДКЛЮЧЕННОЕ устройство

Connector.Logger.WriteLine("[FileSysApiMon.RemovableDeviceMonitor] Обнаруженно подключение съемного устройства", LogLevel.WARN);

foreach (string serialNumber in SerialNumbers)

{

if (!HardDrives.CheckSerial(serialNumber))

{

//Если устройство с таким серийным номером отсутствует в таблице

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Устройство ранее не подключалось SER:{serialNumber}, ожидание 3000ms", LogLevel.WARN);

Thread.Sleep(3000); //Даем винде время подумать

DriveInfo[] ConnectedDrives = DriveInfo.GetDrives();

//Выделение этого носителя, среди массива DriveInfo

foreach (DriveInfo drive in ConnectedDrives)

{

if (!drive.IsReady || drive.DriveType != DriveType.Removable)

{

continue;

}

if (!HardDrives.CheckDrive(drive))

{

//Если устройство с такими данными(серийник не проверяется) не существует в таблице

byte countDevices = HardDrives.AddNewDrive(drive, serialNumber);

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Устройство добавлено в таблицу SER:{serialNumber} TOTAL\_SIZE:{drive.TotalSize} FILESYS:{drive.DriveFormat}, колво устройств в таблицe {countDevices}", LogLevel.WARN);

if (Configuration.RemovableAutoScan)

{

new Task(() =>

{

int probes = 0;

while (probes++ < 10)

{

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Проба сканировать({probes})", LogLevel.WARN);

if (drive.IsReady)

{

Thread.Sleep(200);

try

{

AddRemovableDeviceToScan(drive.Name);

}

catch

{

continue;

}

break;

}

Thread.Sleep(500);

}

}).Start();

}

}

else

{

 //Если устройство с таким серийником отсутствует в таблице, но существует с такими данными

Connector.Logger.WriteLine("[FileSysApiMon.RemovableDeviceMonitor] Устройство с таким серийником отсутствует в таблице, но существует с такими данными", LogLevel.ERROR);

}

}

}

else

{

//Если в таблице есть устройство с таким серийником, то сравниваем сейчас его новые данные с теми, которые есть у нас в таблице

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Устройство с таким серийным номером SER:{serialNumber}, ранее уже подключалось", LogLevel.INFO);

HardDrives.Drive DriveInfoFromTable = HardDrives.getDriveBySerial(serialNumber);

Thread.Sleep(2000); //Хз почему, но винде нужно дать время подумать

DriveInfo[] ConnectedDrives = DriveInfo.GetDrives();

foreach (DriveInfo drive in ConnectedDrives)

{

//Выделение этого носителя, среди массива DriveInfo

if (drive.IsReady && !HardDrives.checkDriveSerial(drive, serialNumber)) continue;

if (Configuration.RemovableAutoScan)

{

new Task(() =>

{

int probes = 0;

while (probes++ < 10)

 {

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] Проба сканировать({probes})", LogLevel.WARN);

if (drive.IsReady)

{

Thread.Sleep(200);

try

{

AddRemovableDeviceToScan(drive.Name);

}

catch

{

continue;

}

break;

}

Thread.Sleep(500);

}

}).Start();

}

if (drive.IsReady && DriveInfoFromTable.TotalFreeSpace != drive.TotalFreeSpace)

{

Connector.Logger.WriteLine($"[FileSysApiMon.RemovableDeviceMonitor] {drive.Name} Съемное устройство было изменено на другом устройстве, требуется перепроверка файлов", LogLevel.WARN);

try

{

if (Configuration.RemovableAutoScan) AddRemovableDeviceToScan(drive.Name);

}

catch (Exception ex)

{

Connector.Logger.WriteLine(ex.Message, LogLevel.ERROR);

}

//Если свободное место во время предыдущего подключения

// не совпадает с свободным местом при текущем подключении, значит там что то изменилось

}

else

{

Connector.Logger.WriteLine("[FileSysApiMon.RemovableDeviceMonitor] Эта флешка не изменялась на других устройствах", LogLevel.WARN);

}

}

}

}

}

if (collection.Count < HardDrives.countConnectedRemovableDevices)

{

Connector.Logger.WriteLine("[FileSysApiMon.RemovableDeviceMonitor] Обнаруженно отключение съемного устройства", LogLevel.WARN);

HardDrives.RefreshConnectedDevices(SerialNumbers);

}

HardDrives.countConnectedRemovableDevices = (byte)collection.Count;

}

}

Thread.Sleep(500);

}

}



static public void ClearAllDevices()

{

HardDrives.DriveTable.Clear();

}

static public void Init()

{

ThreadMonitor = new Thread(Worker)

{

Priority = ThreadPriority.Lowest,

Name = "RemovableDeviceMonitor"

};

ThreadMonitor.Start();

Thread.Sleep(100);

}

static public void StopService()

{

ThreadMonitor.Abort();

}

}

public static class PartitionMonitor

{

private static readonly Thread CommandExecuter = new Thread(CommandThread);

private static FileSystemWatcher[] FileSystemWatchers = new FileSystemWatcher[0];

private static readonly Mutex FileSystemWatchers\_sync = new Mutex();

public static List<string> CreatedFilesBuffer = new List<string>();

private static readonly Mutex CreatedFilesBuffer\_sync = new Mutex();

public static Queue<string> FileQueue = new Queue<string>();

public static Thread LoaderThread = new Thread(Loader) { Name = "MonitorLoader" };

public static Thread BufferCleaner = new Thread(Cleaner) { Name = "MonitorBufferCleaner" };

private static bool RemoveIfExists(string path)

{

for (int index = 0; index < CreatedFilesBuffer.Count; index++)

{

if (CreatedFilesBuffer[index] == path)

{

CreatedFilesBuffer.RemoveAt(index);

return true;

}

}

return false;

}

public static void Cleaner()

{

while (true)

{

Connector.Logger.WriteLine($"[FileSysApiMon.AutoCleaner] Очистка буффера файлов {CreatedFilesBuffer.Count}");

CreatedFilesBuffer.Clear();

Thread.Sleep(TimeSpan.FromSeconds(60));

}

}

public static void Loader()

{

while (true)

{

if(FileQueue.Count > 0)

{

var file = FileQueue.Dequeue();

if (Configuration.Disable)

{

continue;

}

if (RemoveIfExists(file))

{

continue;

}

else

 {

CreatedFilesBuffer.Add(file);

}

Connector.FilterPipeWriter.Write("1" + file);

}

Thread.Sleep(10);

}

}

public static void CommandThread()

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Wait connection...");

{

Connector.CommandPipe.WaitForConnection();

Connector.CommandReader = new BinaryReader(Connector.CommandPipe, Configuration.NamedPipeEncoding);

}

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Connected", LogLevel.OK);

while (true)

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Wait command...");

string buffer = Connector.CommandReader.ReadString();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Read command ->" + buffer);

if (buffer.Length > 0)

{

//Парсинг операндов

string[] args = buffer.Substring(buffer.IndexOf('\*')+1).Split('&');

switch (buffer[buffer.IndexOf('\*') - 1])

{

//command id 0 - Create Partition Monitor

case '0':

{

if (args[1].Length == 0)

{

args[1] = "\*.\*";

}

CreatePartitionMon(args[0], args[1]);

break;

}

//command id 1 - Disable partition monitor

case '1':

{

DisablePartitionMon(args[0]);

break;

}

//Включить авто проверку съемных носителей

case '2':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Включаю автоскан съемных носителей");

Configuration.RemovableAutoScan = true;

break;

}

//Выключить авто проверку съемных носителей

case '3':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Выключаю автоскан съемных носителей");

Configuration.RemovableAutoScan = false;

 break;

}

//Очистить информацию о подключенных устройствах

case '4':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Очищаю информацию о подключенных устройствах");

RemovableDeviceMonitor.ClearAllDevices();

break;

}

//Приостановить защиту

case '5':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Защита отключена!", LogLevel.WARN);

Configuration.Disable = true;

break;

}

//Активировать защиту

case '6':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Защита включена!", LogLevel.OK);

Configuration.Disable = false;

break;

}

//Выключить всё

case '7':

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю всё!", LogLevel.OK);

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю монитор съемных носителей!", LogLevel.OK);

RemovableDeviceMonitor.ThreadMonitor.Abort();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Поток-загрузчик!", LogLevel.OK);

LoaderThread.Abort();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Чищу все записи!", LogLevel.OK);

RemovableDeviceMonitor.ClearAllDevices();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю входную трубу!", LogLevel.OK);

Connector.FilterInputPipe.Close();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю командную трубу!", LogLevel.OK);

Connector.CommandPipe.Close();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Остановка потока загрузчика!", LogLevel.OK);

LoaderThread.Abort();

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Остановка потока очистки буффера!", LogLevel.OK);

BufferCleaner.Abort();



foreach (FileSystemWatcher watcher in FileSystemWatchers)

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю монитор раздела!", LogLevel.OK);

watcher.Dispose();

}

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Отключаю командный поток!", LogLevel.OK);

CommandExecuter.Abort();

break;

}

default:

{

Connector.Logger.WriteLine("[FileSysApiMon.CommandThread] Command not found", LogLevel.WARN);

break;

}

}

}

}

}

/// <summary>

/// Создать и включить FileSystemWatcher для раздела

/// </summary>

public static void CreatePartitionMon(string PartitionPath, string Filter)

{

FileSystemWatchers\_sync.WaitOne();

{

Array.Resize(ref FileSystemWatchers, FileSystemWatchers.Length + 1);

FileSystemWatchers[FileSystemWatchers.Length - 1] = new FileSystemWatcher(PartitionPath, Filter);

FileSystemWatchers[FileSystemWatchers.Length - 1].NotifyFilter = NotifyFilters.FileName | NotifyFilters.Size | NotifyFilters.CreationTime;

FileSystemWatchers[FileSystemWatchers.Length - 1].IncludeSubdirectories = true;

FileSystemWatchers[FileSystemWatchers.Length - 1].Created += CreateFileEvent;

FileSystemWatchers[FileSystemWatchers.Length - 1].Changed += ChangedFileEvent;

FileSystemWatchers[FileSystemWatchers.Length - 1].Error += Error;

FileSystemWatchers[FileSystemWatchers.Length - 1].EnableRaisingEvents = true;

}

FileSystemWatchers\_sync.ReleaseMutex();

Connector.Logger.WriteLine($"[FileSysApiMon.CreatePartition] Created api monitor for {PartitionPath}", LogLevel.OK);

}

/// <summary>

/// Disable FileSystemWatcher for partition

/// </summary>

/// <param name="PartitionPath"></param>

public static void DisablePartitionMon(string PartitionPath)

{

lock (FileSystemWatchers\_sync)

{

for (int index = 0; index < FileSystemWatchers.Length; index++)

{

 if (FileSystemWatchers[index].Path == PartitionPath)

{

{

FileSystemWatchers[index].Dispose();

FileSystemWatchers[index] = FileSystemWatchers[FileSystemWatchers.Length - 1];

Array.Resize(ref FileSystemWatchers, FileSystemWatchers.Length - 1);

}

}

}

}

}

static void CreateFileEvent(object sender, FileSystemEventArgs e)

{

FileQueue.Enqueue(e.FullPath);

}

static void ChangedFileEvent(object sender, FileSystemEventArgs e)

{

FileQueue.Enqueue(e.FullPath);

}

static void Error(object sender, ErrorEventArgs e)

{

Connector.Logger.WriteLine($"[FileSysApiMon.Error] =============", LogLevel.ERROR);

Connector.Logger.WriteLine($"[FileSysApiMon.Error] ERROR {e}", LogLevel.ERROR);

Connector.Logger.WriteLine($"[FileSysApiMon.Error] =============", LogLevel.ERROR);

}

public static void Init()

{

Connector.FilterInputPipe.Connect();

Connector.FilterPipeWriter = new BinaryWriter(Connector.FilterInputPipe, Configuration.NamedPipeEncoding);

CommandExecuter.Start();

}

/// <summary>

/// Disable all threads and FileSys watchers

/// </summary>

public static void StopAll()

{

CommandExecuter.Abort();

Connector.FilterInputPipe.Close();

Connector.CommandPipe.Close();

foreach (FileSystemWatcher watcher in FileSystemWatchers)

{

watcher.Dispose();

}

}

}

public static class Initializator

{

public static byte EntryPoint()

{

new Task(() =>

{

PartitionMonitor.LoaderThread.Start();

PartitionMonitor.BufferCleaner.Start();

Connector.Init();

PartitionMonitor.Init();

RemovableDeviceMonitor.Init();

}).Start();

return 0;

}

public static void Stop()

{

PartitionMonitor.StopAll();

}

}

}