МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

WEB-дизайн и мультимедиа
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка приложения для архивации данных фармацевтической компании «ОЗОН»

Обучающийся Л.А. Перешивайлов
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., В.С. Климов
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.ф.н., М.М. Бажутина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

Аннотация

Тема бакалаврской работы – «Разработка приложения для архивации данных фармацевтической компании «ОЗОН»»

Цель бакалаврской работы – разработка программного обеспечения для архивации данных, предназначенного для ООО «Управляющая компания» («Озон Фарм»), с учётом предъявленных заказчиком требований.

Объектом исследования является процесс сжатия и архивации данных.

Предметом исследования является автоматизация процесса архивации неиспользуемых файлов.

Структура работы представлена введением, 3 главами, заключением, списком литературы и приложением.

В первой главе представлен анализ алгоритмов сжатия и методов архивации данных.

Во второй главе идет моделирование системы архивации, в том числе выбор средств для её реализации, а также проектирование основных функций.

В третьей главе представлен процесс реализации проектного решения с описанием используемых библиотек, разработки дизайна пользовательского интерфейса итоговой программы-архиватора, тестирования приложения.

Разработан программный модуль архивации данных с возможностью создания iso-образа и функцией автоматической архивации неиспользуемых старых файлов в фоновом режиме.

Результаты бакалаврской работы имеют практический интерес и могут быть рекомендованы к использованию работниками отдела технической поддержки ООО «Управляющая компания» («Озон Фарм») с целью повышения эффективности работы отдела.

Бакалаврская работа состоит из 44 страниц текста, 26 рисунков и списка из 25 источников.

Abstract

The title of the bachelor’s thesis is «Development of an application for archiving data of the pharmaceutical company «OZON»».

The work consists of a 44-page explanatory note including 26 figures and a list of 25 references.

The aim of the work is to develop software for data achieving intended for OOO «Management Company» (Ozon Pharm), taking into account the requirements imposed by the customer.

The object of the research is the process of data compression and archiving.

The subject of the research is the automation of the process of archiving unused files.

In the first part we analyze data compression algorithms and archiving methods.

In the second part we carry out modeling of the archiving system, including the choice of instruments for its implementation, as well as the design of the main functions.

In the next part we implement a design solution with a description of the libraries used, developing the design of the user interface of the final archiver program, testing the application.

A software module for data archiving has been developed with the ability to create an iso-image and the function of automatic archiving of unused old files in the background.

The shortcomings of the bachelor's thesis are of practical interest and can be recommended for employees of the technical support department of OOO "Management Company" (Ozon Pharm) in order to improve the efficiency of the department.

Оглавление

Введение

В стремительно развивающемся современном информационном обществе достаточно особенно остро стоит вопрос хранения и передачи информации. Несмотря на непрерывно возрастающий накопительный объем информационных носителей, порой требуется в течении долгого времени хранить большие объёмы данных, или же переместить их на хранилище небольшой ёмкости для транспортировки, таком как флеш-накопитель. Может возникнуть также потребность создания копии имеющихся данных с целью предотвратить их утрату в случае непредвиденной ситуации. В каждом из приведённых случаев возникает потребность уменьшить размер занимаемого данными пространства на информационном носителе.

Для удовлетворения вышеупомянутой потребности были разработаны особые алгоритмы – так называемые алгоритмы сжатия. Путём устранения избыточности, которую содержат исходные данные, сжатие сокращает объём пространства, требуемый для хранения файлов, и, как следствие, количество времени, необходимого для передачи информации.

Целью данной работы является разработка программного обеспечения для архивации данных, предназначенного для ООО «Управляющая компания» («Озон Фарм»), с учётом предъявленных требований.

Достичь поставленной в данной работе цели позволит решение следующих задач:

- провести изучение теоретических основ архивации, включающее изучение алгоритмов и типов архивации;
- сформулировать требования к программному обеспечению;
- спроектировать систему архивации и пользовательский интерфейс;
- разработать и протестировать приложение.

Структура работы представлена тремя главами. Первая глава посвящена изучению теоретического материала, посвящённого алгоритмам сжатия и методам архивации.

Вторая глава посвящена моделированию системы архивации, в том числе составлению списка требований к разрабатываемой системе, основываясь на требованиях и пожеланиях заказчика, выбору средств для программной реализации приложения, а также проектированию основных функций.

Третья глава отражает процесс реализации проектного решения средствами выбранного языка программирования с описанием используемых библиотек, а таже разработку дизайна пользовательского интерфейса итоговой программы-архиватора. Тестирование итогового продукта и его результаты также освещены в третьей главе.

Глава 1 Теоретические основы архивации

1.1 Алгоритмы сжатия

1.1.1 Алгоритмы Шеннона - Фано и Хаффмана

Выделяют два больших класса, на которые можно разделить все разработанные на сегодняшний день алгоритмы сжатия данных – алгоритмы сжатия с потерями и алгоритмы сжатия без потерь. В данной работе будут рассматриваться исключительно алгоритмы сжатия без потерь. Это обусловлено тем, что алгоритмы сжатия с потерями применяются в основном для сжатия аналоговых данных: звука, статичных изображений, видеоряда; целью же данной работы является создание программного модуля, выполняющего сжатие различных типов данных в единый архив, с возможностью их последующего восстановление в исходном виде [12].

Основной принцип алгоритмов сжатия опирается на то, что любой файл имеет повторяющиеся блоки данных. Это и есть так называемая избыточность информации. От этой избыточности можно избавиться, например, отследив повторяющиеся блоки (фразы) и обозначив для них коды. Самые частые повторения, обозначаются самыми короткими кодами. В итоге целая строка может быть заменена несколькими битами [14].

После появления первых мейнфреймов в 1949 году американскими учёными Клодом Шенноном и Робертом Фано был придуман один из первых алгоритмов сжатия, который получил название «Алгоритм Шеннона - Фано». Алгоритм присваивает коды символам в блоке данных, исходя из вероятности их появления в данном блоке. Коды Шеннона - Фано являются префиксными, что в свою очередь означает, что можно однозначно декодировать абсолютно любую последовательность кодов, так как ни одно кодовое слово не может являться префиксом другого.

В 1952 году аспирант Массачусетского технологического института Дэвид Хаффман, будучи студентом в классе у Роберта Фано, в качестве учебной работы выбрал поиск улучшенного метода бинарного кодирования данных и в результате разработал улучшенный алгоритм Шеннона - Фано,

названный алгоритмом Хаффмана. В отличие от алгоритма Шеннона - Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов m_2 с более чем двумя символами.

Алгоритм Хаффмана в основном заключается в построении дерева кодирования Хаффмана (Н-дерево) с использованием таблицы с вероятностями или частотой появления символов в сжимаемом потоке данных [1].

Процесс построения бинарного древа начинается с составления списка свободных узлов из символов таблицы входного алфавита. Вероятность появления символа принимается за вес узла. Затем выбираются два свободных узла с наименьшими весами и создаётся родительский узел с весом, равным сумме их весов. Как правило, левой дуге следования в соответствие ставится бит «0», а правой – бит «1». Пара узлов выбывает из списка свободных узлов, а созданный родительский узел пополняет его, после чего процесс повторяется снова до тех пор, пока не останется только один свободный узел, который является корнем дерева Хаффмана [17].

Пример нахождения оптимальных префиксных кодов по алгоритму Хаффмана продемонстрирован на рисунке 1.

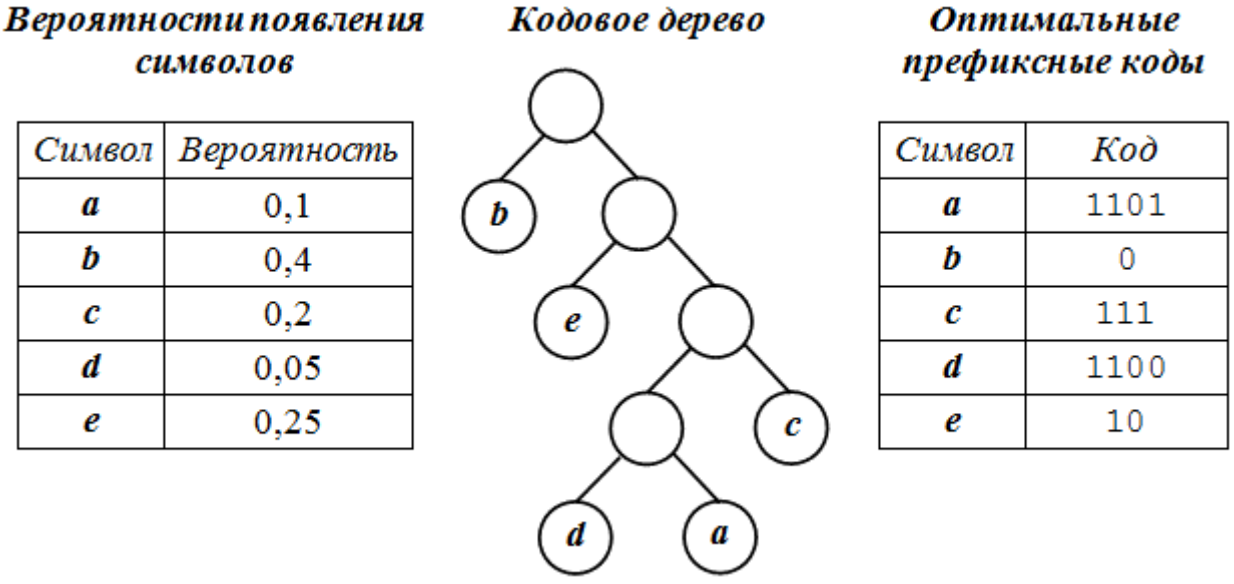


Рисунок 1 – Нахождение оптимальных префиксных кодов алгоритма Хаффмана

В ранних версиях алгоритмов Шеннона – Фано и Хаффмана применялись таблицы кодов, определённых заранее. Однако в последствии стали применяться коды, динамически сформированные на основе входных данных, предназначаемых для сжатия [18].

1.1.2 Алгоритмы семейства LZ

«LZ – семейство алгоритмов словарного сжатия данных, получившее своё название по инициалам двух исследователей – израильских математиков Абрахама Лемпэла (Abraham Lempel) и Якоба Зива (Jacob Ziv), разработавших в 1970-х годах алгоритмы LZ77 и LZ78. На базе LZ77 и LZ78, изменяя и комбинируя их с другими методами, исследователями был создан ряд других алгоритмов семейства LZ — LZO, LZS, LZW, LZC, LZMA»[9].

Первый алгоритм данного семейства – LZ77 был опубликован в 1977 году. Алгоритм основывается на применении динамически создаваемого словаря, так называемого «скользящего окна».

Скользящее окно представляет собой динамическую структуру данных, например буфер, организованную с целью запоминания считанной информации и предоставления доступа к ней. Таким образом, сам процесс алгоритма сжатия LZ77 опирается на замену повторяющейся последовательности ссылкой на предыдущее вхождение данной последовательности, сохранённое в «скользящем окне». Размер самого окна может составлять 2, 4 или 32 килобайта и изменяется динамически [11].

Пример кодирования по алгоритму LZ77 с использование скользящего окна продемонстрирован на рисунке 2.

СЛОВАРЬ(8)	БУФЕР(5)	КОД
"........"	"КРАСН"	<0,0,'К'>
".......К"	"РАСНА"	<0,0,'Р'>
"......КР"	"АСНАЯ"	<0,0,'А'>
".....КРА"	"СНАЯ "	<0,0,'С'>
"....КРАС"	"НАЯ К"	<0,0,'Н'>
"...КРАСН"	"АЯ КР"	<5,1,'Я'>
".КРАСНАЯ"	"КРАС"	<0,0,' '>
"КРАСНАЯ "	"КРАСК"	<0,4,'К'>
"АЯ КРАСК"	"А...."	<0,0,'А'>

Рисунок 2 – Пример кодирования по алгоритму LZ77

Опубликованный 1978 году алгоритм LZ78, в отличие от своего предшественника уже не использует «скользящее окно». Вместо этого он хранит словарь из уже просмотренных фраз, иными словами, вместо того, чтобы создавать словарь динамически, LZ78 парсит данные перед процессом замены повторов.

Приведённые выше алгоритмы обрели большую популярность и стали буквально прародителями большого семейства LZ. Среди алгоритмов этого семейства, большинство основаны на LZ77, и у этого имеется довольно весомое обоснование. Вся суть в том, что LZW, производный алгоритм LZ78, в 1984 году был запатентован компанией Unisys. После этого они начали «преследовать» всех и каждого, включая даже случаи использования изображений в формате GIF. Под раздачу попали даже разработчики UNIX, так как в ней использовался алгоритм LZC, являющийся вариацией алгоритма LZW. Из-за проблем с правами пришлось отказаться от его использования. Предпочтение отдали алгоритму DEFLATE (gzip) и преобразованию Барроуза – Уилера, BWT (bzip2).

1.1.3 Алгоритм DEFLATE

Алгоритмы сжатия применялись большими корпорациями с целью рационального использования пространства хранилищ данных, однако широкое распространение алгоритмов произошло в конце 80-х, после зарождения интернета. Ввиду чрезвычайно малой пропускной способности каналов передачи данных по сети, для сжатия этих самых данных были придуманы такие форматы, как PNG, GIF и ZIP.

Первый коммерчески успешный архиватор был разработан Томом Хендерсеном и выпущен в 1985 году компанией System Enhancement Associates под названием ARC. Он был одним из первых архиваторов, способных сжимать несколько файлов в архив, а в его основе лежал модифицированный алгоритм LZW. К тому же исходный код ARC был открытым.

Американский программист, выпускник Висконсинского университета в Милуоки Филлип Уолтер Кац, вдохновившись популярностью ARC и успехом Хендерсона, выпустил PKARC, улучшив алгоритмы сжатия, переписав их на Ассемблере, что ускорило процесс сжатия. Однако, он был засужен Хендерсоном и признан виновным.

В 1989 году Фил Кац поработал над архиватором, сильно его изменив, и выпустил PKZIP. После этого, его снова атаковали, в этот раз уже в связи с патентом на алгоритм LZW. В итоге Кац заменил базовый алгоритм сжатия, разработав собственный под названием IMPLODE. В 1993 году с выходом PKZIP 2.0, алгоритм был заменён снова, и заменой стал DEFLATE.

DEFLATE использует комбинацию алгоритмов LZ77 и Хаффмана [5].

Схема работы алгоритма продемонстрирована на рисунке 3.

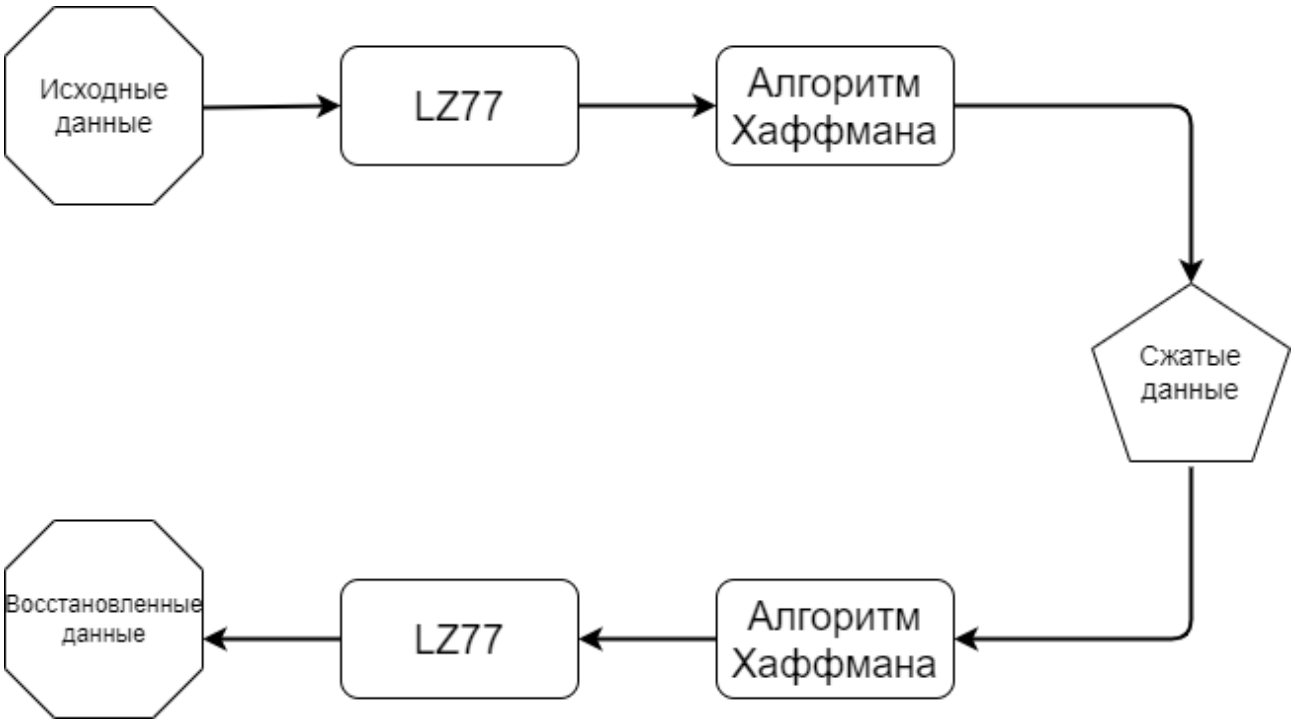


Рисунок 3 – Схема работы алгоритма DEFLATE

Deflate-поток содержит серии блоков. Перед каждым блоком находится трёхбитовый заголовок.

Один бит: флаг последнего блока:

– 1: блок последний;

– 0: блок не последний.

Два бита: метод, с помощью которого были закодированы данные:

– 00: данные не закодированы (в блоке находятся непосредственно выходные данные);

– 01: данные закодированы по методу статического Хаффмана;

– 10: данные закодированы по методу динамического Хаффмана;

– 11: зарезервированное значение (ошибка).

Большая часть блоков кодируется с помощью метода 10 (динамический Хаффман), который предоставляет оптимизированное дерево кодов Хаффмана для каждого нового блока. Инструкции для создания дерева кодов Хаффмана следуют непосредственно за заголовком блока [15].

Само сжатие выполняется в два этапа:

1. замена повторяющихся строк указателями (алгоритм LZ77);

2. замена символов новыми символами, основываясь на частоте их использования (алгоритм Хаффмана).

1.2 Типы архивации

Выделяют три типа архивации: полная (Full), добавочная (Incremental) и разностная (Differential).

Для лучшего понимания на рисунках 4, 5 и 6 продемонстрированы визуальные представления названных типов архивации соответственно.

Полную архивацию можно также считать стандартным типом архивации. При ней в архив записываются все выбранные файлы.

Рисунок 4 – Визуальное представление полной архивации

Добавочная и разностная архивации используются в основном для создания резервных копий файлов.

При добавочной архивации в архив из выбранных файлов записываются только те, что были добавлены или изменены с момента предыдущей полной или добавочной архивации. В сравнении с предыдущим, этот способ архивации сокращает используемое для хранения копий дисковое пространство и позволяет создавать резервные копии файлов с максимальной скоростью, однако при этом продолжительность восстановления увеличивается пропорционально числу произведённых добавочных архиваций. По сравнению с полной, где необходимо восстановить только последнюю копию, при добавочной нужно последовательно восстановить полную резервную копию, а затем каждую добавочную.

Рисунок 5 – Визуальное представление добавочной архивации

Разностная архивация занимает промежуточное положение между полной и добавочной. При ней в архив записываются файлы, которые были изменены или добавлены с момента последней полной архивации.

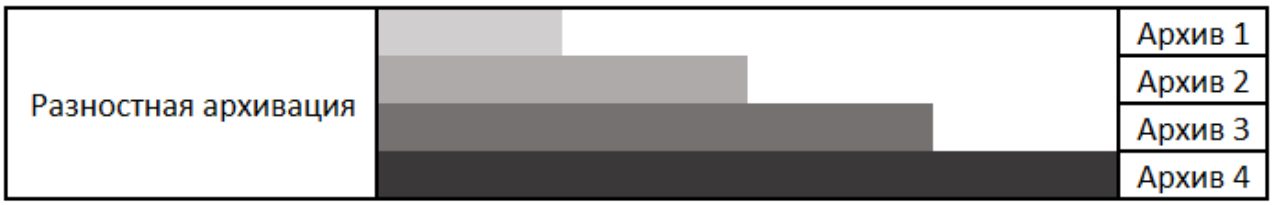


Рисунок 6 – Визуальное представление разностной архивации

В данном случае при восстановлении достаточно будет восстановить полную резервную копию и последнюю разностную.

Выводы к главе 1

В данной главе была рассмотрена теоретическая информация об основных типах архивации и некоторых, наиболее известных алгоритмах.

Последний рассмотренный алгоритм – DEFLATE, несмотря на свой возраст, остаётся актуальным и является основным алгоритмом архивации для формата ZIP, а потому хорошо подходит для достижения цели данной работы.

Глава 2 Проектирование системы архивации

2.1 Техническое задание на разработку

Перед нами стоит задача создания практичной, быстродействующей системы архивации данных с дружественным, интуитивно понятным интерфейсом и минимальными требованиями к знаниям пользователя.

Опираясь на требования, предъявленные ООО "Озон Фарм", был составлен список функций, которые должен выполнять разрабатываемый программный продукт:

- архивация в режимах Full, Incremental, Differential;
- возможность архивации NTFS прав;
- возможность создания iso-образа;
- система долгих архивов, то есть автоматическая архивация в фоновом режиме файлов, не использующихся долгое время.

Язык разработки программного модуля и интерфейса, а также формат архивации оставлены на выбор разработчика.

2.2 Выбор средств разработки

2.2.1 Формат архива

Заказчик оставил выбор формата архива на усмотрение разработчика, а потому стоит рассмотреть доступные варианты и выбрать наиболее подходящий для данной работы.

На данный момент самыми распространёнными форматами архивов являются ZIP, 7Z и RAR.

Первым был рассмотрен 7Z. Формат 7z изначально был выпущен как архиватор 7-Zip, исходный код, которого является открытым.

Формат 7z обеспечивает следующие основные функции:

- Открытая модульная архитектура, которая позволяет использовать любой метод сжатия, преобразования или шифрования.
- AES 256 битное шифрование.
- Имена файлов Unicode.

– Поддержка сплошного сжатия, при котором несколько файлов одинакового типа сжимаются в одном потоке, чтобы использовать объединённую избыточность, присущую аналогичным файлам.
– Сжатие и шифрование заголовков архива.
– Поддержка архивов, состоящих из нескольких частей
– Поддержка пользовательских библиотек плагинов кодеков.

Также открытая архитектура формата позволяет добавлять в стандарт дополнительные будущие методы сжатия.

Следом рассмотрены особенности RAR, проприетарного формата сжатых данных:

– Практически полное отсутствие ограничений на количество файлов в архиве и их объём.
– Дополнительно добавляемые в архив служебные данные для восстановления, позволяющие восстанавливать архив при его физическом повреждении.
– Шифрования AES, включая шифрование оглавления и служебной информации, так что без указания правильного пароля невозможно даже просмотреть оглавление архива.
– Комментарии к архиву, возможность хранения и восстановления прав доступа NTFS и др.
– Добавление в архивы (RAR5) хеш-сумм BLAKE2 для практически полной гарантии точной идентификации файлов (в отличие от контрольных сумм CRC32, которые могут быть легко подделаны).
– Добавление в архивы (RAR5) служебной информации, ускоряющей их открытие (вывод оглавления), что особенно заметно для крупных архивов с большим количеством файлов.

Стоит заметить, что данный формат защищён авторскими правами, а потому идёт рука об руку только с условно-бесплатным архиватором WinRAR.

Напоследок, был рассмотрен формат ZIP. Архив ZIP может содержать один или несколько файлов и каталогов, которые могут быть сжаты разными

алгоритмами. Наиболее часто в ZIP используется алгоритм сжатия DEFLATE, рассмотренный в главе 1.

В настоящее время формат ZIP считается общепризнанным форматом для многих приложений, включающих функции сжатия, резервного копирования и обмена данными, а также имеет встроенную поддержку в таких операционных системах, как Microsoft Windows и Apple Mac OS X. По этим причинам данный формат подходит для разрабатываемой системы архивации как нельзя лучше.

2.2.2 Язык программирования

В качестве языка программирования для написания кода программного модуля данной работы было принято решение использовать Python ввиду его лёгкости, понятности и удобства работы с данными. Простота кода облегчает не только разработку приложений, но и дальнейшее их обслуживание [13].

Python собрал довольно внушительное сообщество, которое негласно считается одним из самых мощных в сфере программирования.

Нельзя не отметить наличие огромного количества библиотек, написанных под всевозможные нужды. Некоторые из этих самых библиотек лягут в основу при написании кода приложения данной бакалаврской работы. Ниже приведён перечень некоторых используемых библиотек:

- pathlib (библиотека, предоставляющая набор классов для обработки путей файловой системы)[19];
- zipfile (библиотека для работы с архивами формата ZIP)[25];
- PySimpleGUI (библиотека для реализации пользовательского интерфейса)[21];
- APScheduler (библиотека, позволяющая реализовать выполнение кода по расписанию либо единожды, либо периодически)[16];
- PyCdlib (библиотека для работы с форматом iso)[20].

2.2.3 Интегрированная среда разработки

Писать код на Python можно, используя IDLE или Python Shell. Они вполне подходят для работы над небольшими, простыми проектами, однако, когда речь заходит о чём-то серьёзном и крупном, то данные варианты являются отнюдь не лучшим выбором. При работе над крупными проектами стоит прибегнуть к использованию интегрированной среды разработки, либо же редактора кода.

«Интегрированная среда разработки, ИСР (англ. IDE, Integrated Development Environment или Integrated Debugging Environment) — система программных средств, используемая программистами для разработки программного обеспечения (ПО)»[6].

IDE включает в себя ряд инструментов, специально предназначенных для разработки. Среди этик инструментов имеется редактор, предназначенный для работы с кодом, который обладает функциями по типу подсветки синтаксиса и автозаполнения; инструменты сборки, выполнения и отладки; также определённую форму системы управления версиями.

Большинство IDE обеспечивают поддержку большого набора языков программирования и располагают внушительным перечнем функций, из-за чего зачастую могут иметь довольно большой вес, вплоть до нескольких десятков гигабайт, а также занимать много времени для загрузки. Ко всему прочему, для правильного пользования IDE и предоставляемым ей инструментарием может потребоваться определённый уровень знаний.

По сравнению с IDE, редакторы кода, представляющие собой, грубо говоря, текстовый редактор с возможностью форматирования кода, куда более лёгкие и быстрые, однако менее функциональные.

Как уже было отмечено ранее, практически все IDE – тяжеловесные, и обладают избыточным для данной работы функционалом. По этой причине были рассмотрены в основном редакторы кода с единственным исключением в лице PyCharm – IDE, буквально разработанной именно для Python.

Первый рассмотренный редактор кода – Sublime Text. Sublime Text является весьма популярным редактором кода. Доступный на всех платформах, Sublime Text имеет встроенную поддержку редактирования Python-кода, а также богатый набор расширений, называемых пакетами, которые расширяют возможности синтаксиса и редактирования.

Установить дополнительный Python-пакет может быть непросто – все пакеты Sublime Text написаны на Python, поэтому для установки пакетов сообщества зачастую может потребоваться выполнить Python-скрипт непосредственно в редакторе. Данный момент в купе с тем фактом, что в редакторе в чистом виде отсутствует поддержка отладки и запуска кода, можно считать довольно серьёзным недостатком в сравнении с конкурентами.

На рисунке 7 продемонстрирован пользовательский интерфейс Sublime Text.

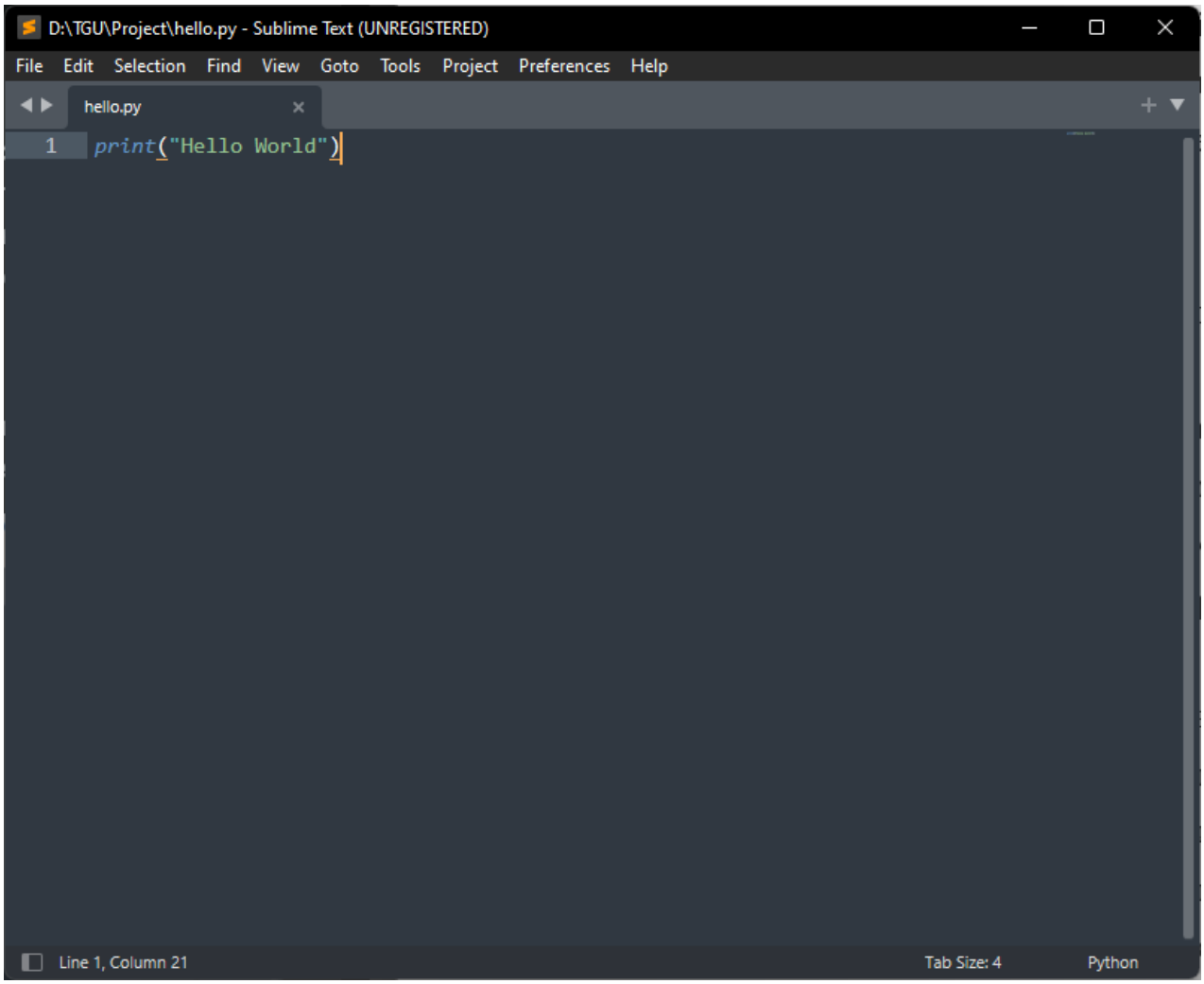


Рисунок 7 – Экран редактора кода Sublime Text

Первым среди рассмотренных конкурентов является Atom. Он доступен на всех платформах и написан с использованием Electron – фреймворка для создания кроссплатформенных desktop-приложений средствами JavaScript, HTML и CSS. Atom имеет множество расширений, которые можно установить непосредственно в Atom, среди которых и расширение для поддержки Python.

Стоит отметить, что, поскольку Atom написан с помощью Electron, он работает не как нативное приложение, а как JavaScript-процесс.

На рисунке 8 продемонстрирован пользовательский интерфейс Atom.

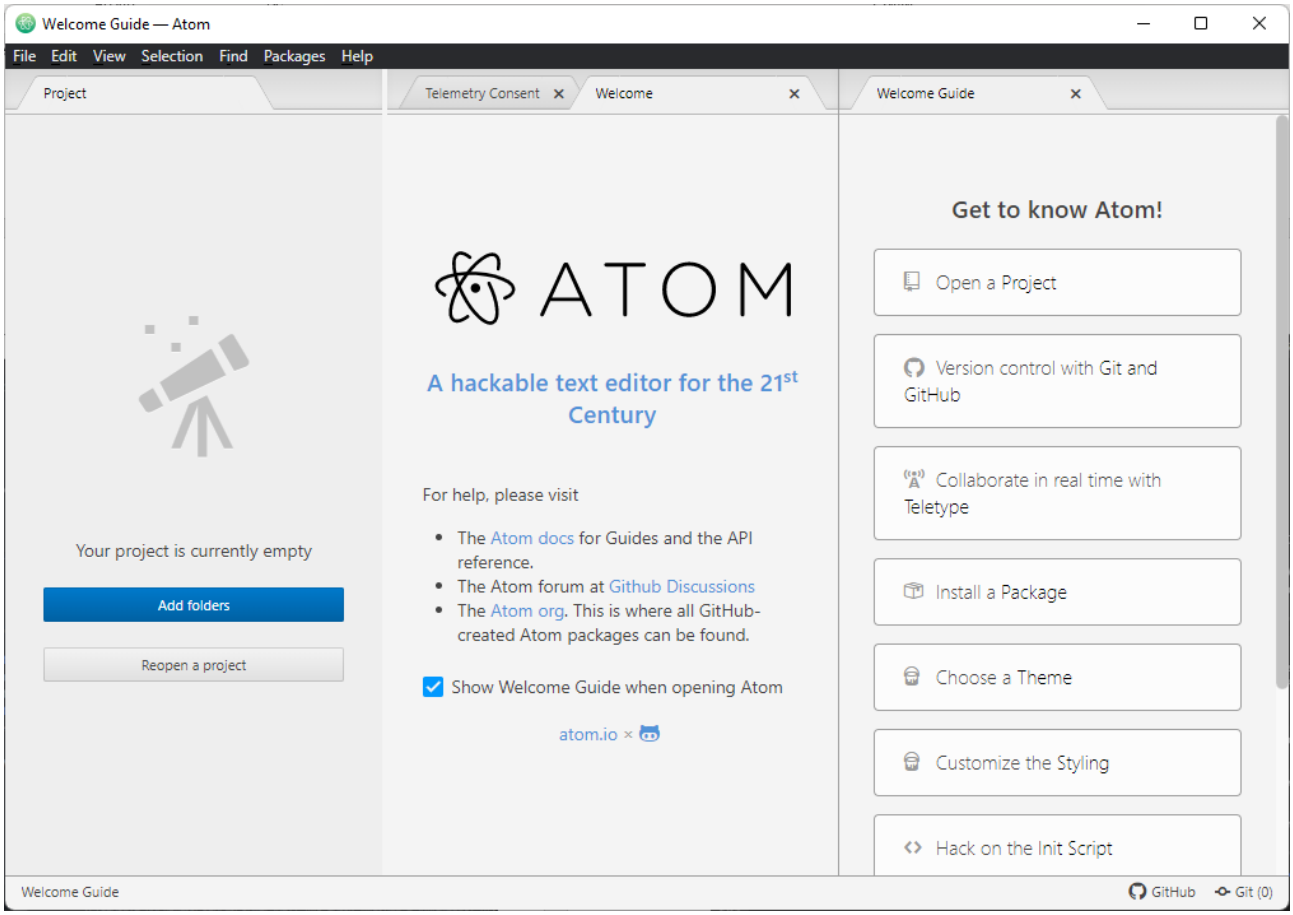


Рисунок 8 – Экран приветствия редактора кода Atom

Последним рассмотренным редактором кода является Microsoft Visual Studio Code (далее VS Code). Он представляет собой полнофункциональный редактор кода, доступный на всех платформах.

VS Code является расширяемым open-source редактором, который можно настроить под любую задачу. Поддержка Python обеспечивается установкой

расширения во вкладке Marketplace (Расширения) в самом VS Code. Как и Atom, VS Code построен на Electron, поэтому у него есть те же преимущества и недостатки.

На рисунке 9 продемонстрирован пользовательский интерфейс VS Code.

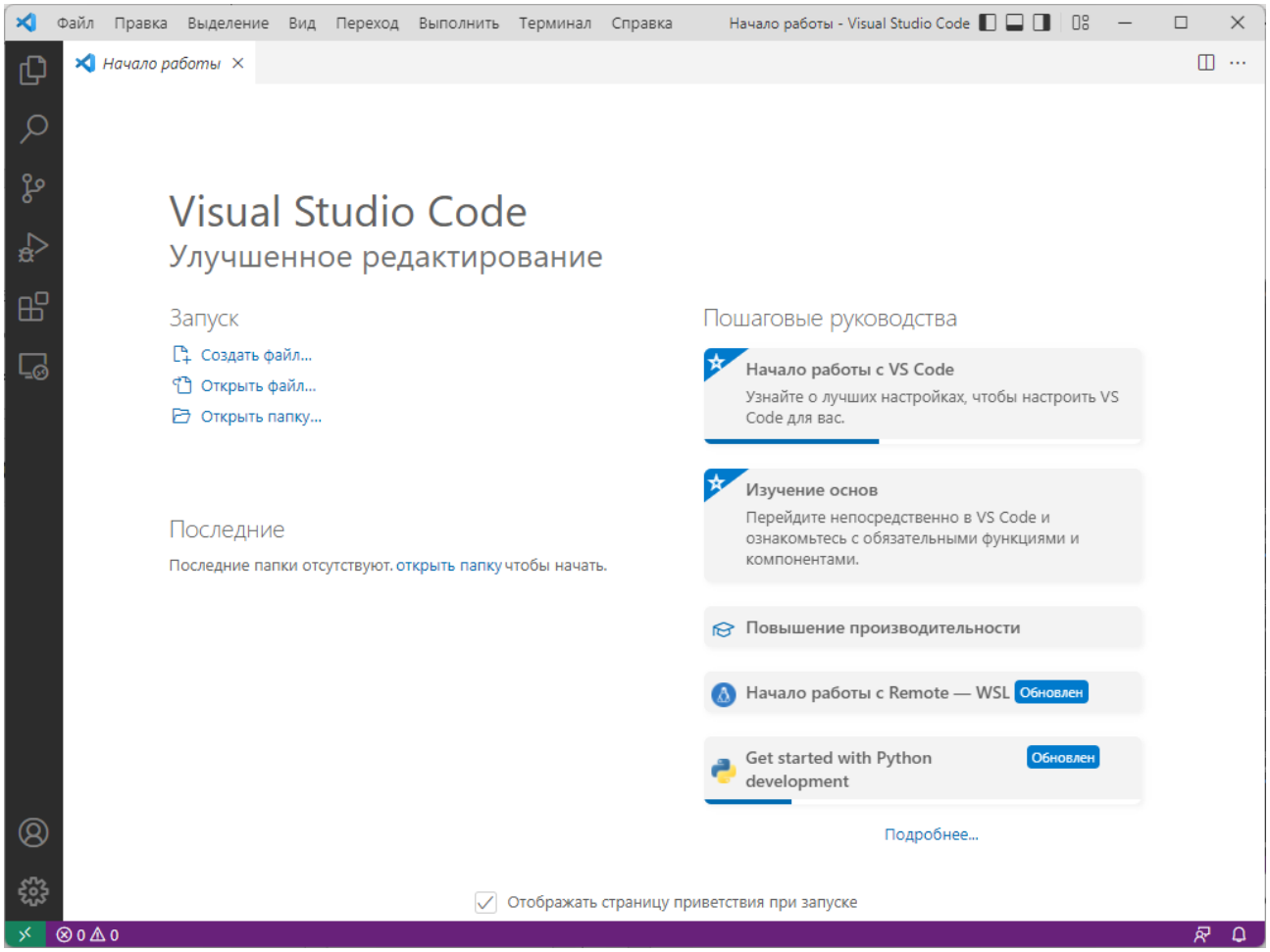


Рисунок 9 – Экран приветствия редактора кода VS Code

В завершение рассмотрена, упомянутая ранее, IDE – PyCharm. Она является одной из лучших IDE для разработки на Python, так как разрабатывалась для работы именно с ним. Существует как бесплатный open-source (Community), так и платный (Professional) варианты IDE. PyCharm доступен на всех платформах и поддерживает Python «из коробки». Стоит отметить, что PyCharm обладает довольно хорошим сообществом пользователей, однако может медленно загружаться, а также может возникнуть потребность корректировки настроек по умолчанию для существующих проектов [8].

Для данной работы в качестве среды разработки был выбран редактор кода VS Code. Данный выбор обусловлен субъективными предпочтениями дизайна и имеющимся опытом работы над проектами в данной среде.

2.3 Проектирование режимов архивации

Для начала стоит рассмотреть режимы архивации. По техническому заданию от заказчика необходимо реализовать архивацию в трёх режимах: полная (Full), добавочная (Incremental), разностная (Differential). Подробно они были рассмотрены в главе 1.

В случае с полной архивацией всё предельно просто – архивируются все входящие исходные данные. Проблема возникает в случае с добавочной и разностной архивацией, ведь необходимо как-либо определить является ли рассматриваемый файл новым или изменённым с момента последней полной (полной и добавочной) архивации.

Для решения данной проблемы было принято решение вести записи (records) архивируемых файлов. Так, например, при полной архивации данные каждого архивируемого файла будут заноситься в словарь, где ключ – абсолютный путь к исходному файлу, значение – хеш-код файла, вычисленное по алгоритму хеширования md5. После завершения процесса архивации заполненный словарь записывается в отдельный файл.

Ведение подобных записей позволит проводить сравнение файлов при добавочной и разностной архивациях, ведь при изменении файла – изменится и его хеш-код, а нового файла не будет в словаре вовсе.

Стоит отметить, что файла с записями должно быть 2 (full_records и incr_records) – для полной и добавочной архивации, т.к. при добавочной архивации необходимо проводить сравнение не только с последней полной, но и последней добавочной архивацией.

Записи будут сохраняться в формате TOML (Tom’s Obvious, Minimal Language / пер. Очевидный язык Тома) ввиду удобства обработки python-словаря с помощью одноимённой библиотеки [23].

На рисунке 10 приведены диаграммы последовательности для каждого из описанных ранее режимов.

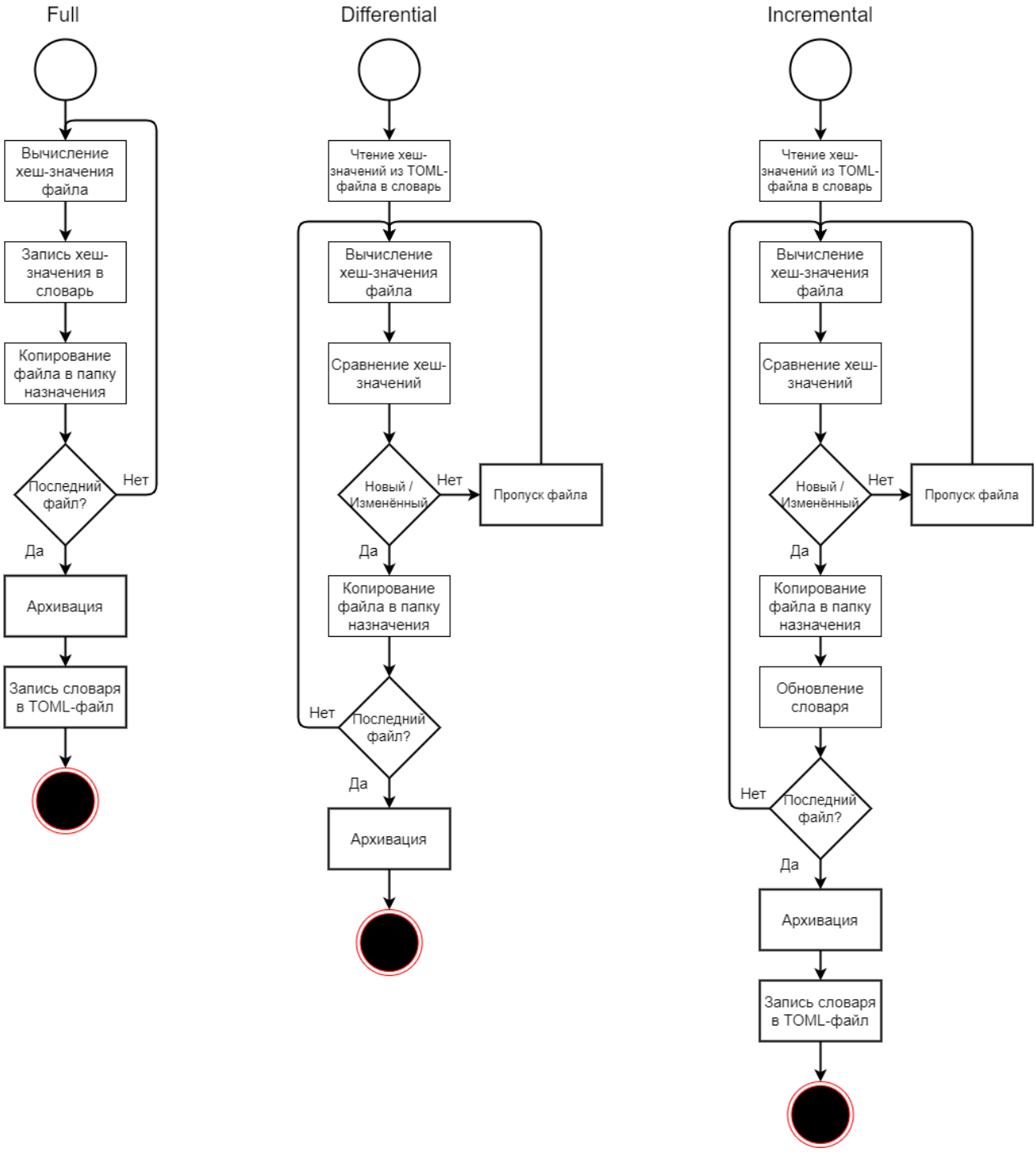


Рисунок 10 – Диаграммы последовательности режимов архивации Full, Incremental и Differential

В данной работе возможность создания iso-образа из исходных файлов, являющаяся требуемой функцией разрабатываемого приложения для

архивации, с точки зрения проектирования выделена как отдельный режим архивации.

ISO – формат образа диска, содержащий файловую систему стандарта ISO 9660, целью которого является обеспечить совместимость носителей под разными операционными системами, такими, как Unix, Mac OS, Windows.

2.4 Архивация NTFS-прав

Основная часть всех пользовательских данных хранится на дисках в виде файлов, к которыми пользователи тем или иным образом получают доступ. Права доступа выдаются на уровне файловой системы NTFS (New Technology File System). Каждый файл в системе хранит служебную информацию – атрибуты и ACL (Access Control List). Атрибуты файла описывают его свойства, а ACL указывает права доступа пользователей, то есть какие пользователи имеют права на взаимодействие с файлом и какие именно действия могут с этим файлом производить [10].

В NTFS существует механизм наследования. Он служит для быстрого и удобного изменения прав доступа к файлам, находящимся во вложенных каталогах

Правило наследования простое и состоит в следующем: каждый дочерний объект в момент своего создания автоматически наследует права доступа ближайшего родительского объекта.

Стоит отметить, что в случаях перемещения или копирования объекта его правила наследования изменяются:

– При копировании объекта с одного тома на другой копируемый объект всегда получает права или разрешения того раздела или расположенного в нём каталога, в который он копируется. Те же правила действуют при перемещении файлов между разными томами.

– При перемещении в пределах одного тома, перемещаемый объект сохраняет свою ACL, изменяется только ссылка на него в таблице MFT.

– При копировании в пределах одного тома копируемый объект получает ACL от ближайшего вышестоящего родительского каталога.

Исходя из требований к разрабатываемому приложению, необходимо реализовать возможность сохранения NTFS-прав исходных данных при архивации, а также их восстановления при деархивации.

Так как контроль над правами осуществляется на уровне системы, самым простым и надёжным путём будет прибегнуть к встроенным возможностям системы. В случае Microsoft Windows, целевой операционной системы, в данной роли выступает утилита «icacls» [4]. С её помощью NTFS-права архивируемых файлов можно сохранить в отдельный файл, например, в текстовый документ (tempACL.txt). Этот документ архивируется вместе с остальными файлами, а при восстановлении файлов права считываются из документа всё той же утилитой, после чего сам документ удаляется.

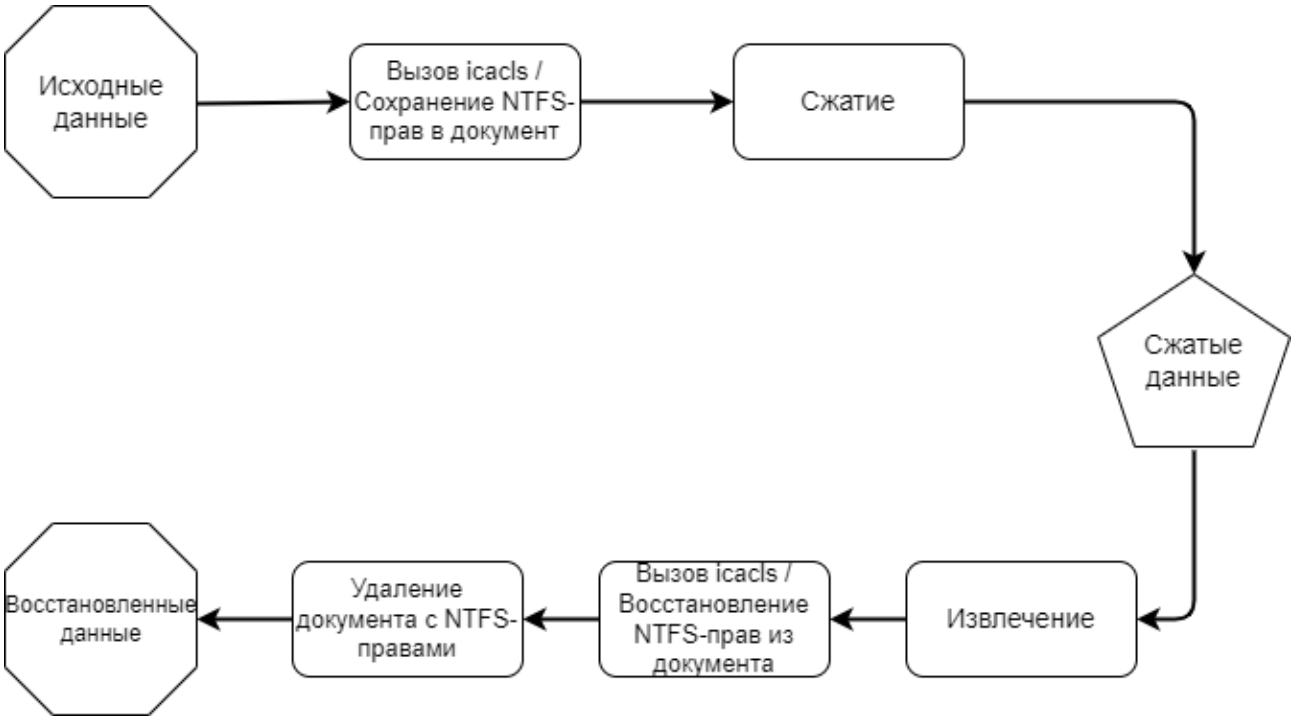


Рисунок 11 – Схема процесса архивации и восстановления NTFS-прав

На рисунке 11 в виде схемы приведена описанная выше последовательность работы алгоритма.

2.5 Проектирование системы долгих архивов

Ключевым и обязательным элементом разрабатываемого приложения для архивации данных, требуемым заказчиком, является автоматическая архивация старых, неиспользуемых файлов в фоновом режиме, так названная заказчиком «система долгих архивов».

Для проектирования данного элемента приложения стоит в первую очередь подробнее рассмотреть его концепцию. Какие файлы считать старыми? Старые файлы – те файлы, которые не открывались для чтения или записи уже более определённого порога времени, иными словами, не использующиеся пользователем или системой длительное время. С целью увеличения эффективности использования дискового пространства возникает необходимость сжатия подобных файлов для уменьшения занимаемого ими места. Автоматизация выполнения данной задачи и является предназначением проектируемого элемента приложения.

Основополагающим элементом автоматизации выступает конфигурационный файл, из которого будут считываться такие параметры как: список путей к исходным данным, путь к месту хранения архивов, порог времени, тип времени (минуты, часы, дни), временной интервал, тип интервала (минуты, часы, дни); а также флаги определяющие сохранять ли NTFS-права и удалять ли исходные данные, после их архивации.

В самой функции архивации для каждого файла будет производиться проверка. Для каждого файла считывается текущее время и последнее время доступа к файлу; если их разность больше порога времени, считанного из config-файла, то файл архивируется, в противном случае – пропускается. Также, в зависимости от значения флага, исходный файл удаляется, либо же сохраняется.

Данный алгоритм необходимо выполнять в фоновом потоке по расписанию (в цикле с определённым интервалом). Временной интервал запуска также будет считываться из конфигурационного файла.

Выводы к главе 2

Данная глава посвящена проектированию функций разрабатываемого приложения для архивации и выбору средств для их реализации.

В результате проделанной работы было составлено техническое задание на разработку, отражающее требования заказчика к возможностям разрабатываемого приложения; сделан выбор, касающийся формата архива, языка программирования и среды разработки. Выбраны ZIP, Python и VS Code соответственно. Также были спроектированы функции, составляющие основу разрабатываемого приложения, с целью дальнейшей их реализации.

Глава 3 Реализация и тестирование приложения для архивации данных

3.1 Реализация функций архивации

Основываясь на проектном решении, описанном в главе 2, разработаны программные реализации алгоритмов режимов архивации.

На рисунке 12 показан листинг кода, реализующего импорт всех используемых библиотек.

```
1  from argparse import ArgumentParser
2  import datetime
3  import hashlib
4  import os
5  import sys
6  import zipfile
7  import pycdlib
8  from shutil import copy2 as copy, rmtree
9  import subprocess
10 import toml
11 from pathlib import Path, PureWindowsPath
12 import PySimpleGUI as sg
13 from psgtray import SystemTray
14 from apscheduler.schedulers.background import BackgroundScheduler
```

Рисунок 12 – Листинг импорта библиотек

Листинг программного кода, реализующего режим полной архивации показан на рисунке 13:

```
def full_zip(conf: dict, save_acl: bool):
    working_dir: Path = os.path.abspath(os.curdir)
    now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
    records: dict = {}

    destination_path: Path = conf["destination"] + "Full_" + now

    try:
        os.makedirs(destination_path)
    except FileExistsError:
        pass

    if type(conf["source-directories"]) is list:
        for path in conf["source-directories"]:
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)
    else:
        path = conf["source-directories"]
        os.makedirs(destination_path + "/" + Path(path).name)
        backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)

    os.chdir(destination_path)
    if save_acl:
        tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
        subprocess.call(["icacls", PureWindowsPath(destination_path),
                         "/save", tmp_acl_path, "/t"])

    zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                               compression=zipfile.ZIP_DEFLATED, allowZip64=True)
    for dir, subdir, files in os.walk(destination_path):
        for file in files:
            zip_file.write(os.path.join(dir, file),
                           os.path.relpath(os.path.join(dir, file), destination_path))
    zip_file.close()

    try:
        Path(tmp_acl_path).unlink()
    except:
      pass

    for file in os.listdir(destination_path):
        if os.path.isdir(file):
            rmtree(destination_path + "/" + file)

    os.chdir(working_dir)
    os.rmdir(destination_path)
    write_toml(records, full_records_path)
```

Рисунок 13 – Листинг кода реализации функции полной архивации

Листинг кода, отвечающего за реализацию режима разностной архивации показан на рисунке 14:

```
85  def diff_zip(conf: dict, save_acl: bool):
86      working_dir: Path = os.path.abspath(os.curdir)
87      now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
88      records: dict = toml.load(full_records_path)
89
90      destination_path: Path = conf["destination"] + "Diff_" + now
91
92      try:
93          os.makedirs(destination_path)
94      except FileExistsError:
95          pass
96
97      if type(conf["source-directories"]) is list:
98          for path in conf["source-directories"]:
99              os.makedirs(destination_path + "/" + Path(path).name)
100             backup(False, path, destination_path + "/" + Path(path).name, records, save_acl)
101     else:
102         path = conf["source-directories"]
103         os.makedirs(destination_path + "/" + Path(path).name)
104         backup(False, path, destination_path + "/" + Path(path).name, records, save_acl)
105
106     os.chdir(destination_path)
107     if save_acl:
108         tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
109         subprocess.call(["icacls", PureWindowsPath(destination_path),
110                          "/save", tmp_acl_path, "/t"])
111
112     zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
113                                compression=zipfile.ZIP_DEFLATED, allowZip64=True)
114     for dir, subdir, files in os.walk(destination_path):
115         for file in files:
116             zip_file.write(os.path.join(dir, file),
117                            os.path.relpath(os.path.join(dir, file), destination_path))
118     zip_file.close()
119
120     try:
121         Path(tmp_acl_path).unlink()
122     except:
123       pass
124
125     for file in os.listdir(destination_path):
126         if os.path.isdir(file):
127             rmtree(destination_path + "/" + file)
128
129     os.chdir(working_dir)
130     os.rmdir(destination_path)
```

Рисунок 14 – Листинг кода реализации функции разностной архивации

Листинг кода, отвечающего за реализацию режима добавочной архивации показан на рисунке 15:

```python
def incr_zip(conf: dict, save_acl: bool):
    working_dir: Path = os.path.abspath(os.curdir)
    now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
    compare: str = hashlib.md5(open(full_records_path, "rb").read()).hexdigest()
    try:
        records: dict = toml.load(incr_records_path)
        if records[full_records_path] != compare:
            records: dict = toml.load(full_records_path)
            records[full_records_path] = compare
    except:
        records: dict = toml.load(full_records_path)
        records[full_records_path] = compare

    destination_path: str = conf["destination"] + "Incr_" + now

    try:
        os.makedirs(destination_path)
    except FileExistsError:
        pass

    if type(conf["source-directories"]) is list:
        for path in conf["source-directories"]:
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(False, path, destination_path + "/" + Path(path).name, records, save_acl)
    else:
        path = conf["source-directories"]
        os.makedirs(destination_path + "/" + Path(path).name)
        backup(False, path, destination_path + "/" + Path(path).name, records, save_acl)

    os.chdir(destination_path)
    if save_acl:
        tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
        subprocess.call(["icacls", PureWindowsPath(destination_path),
                         "/save", tmp_acl_path, "/t"])

    zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                               compression=zipfile.ZIP_DEFLATED, allowZip64=True)
    for dir, subdir, files in os.walk(destination_path):
            for file in files:
                zip_file.write(os.path.join(dir, file),
                               os.path.relpath(os.path.join(dir, file), destination_path))
    zip_file.close()

    try:
        Path(tmp_acl_path).unlink()
    except:
      pass

    for file in os.listdir(destination_path):
        if os.path.isdir(file):
            rmtree(destination_path + "/" + file)

    os.chdir(working_dir)
    os.rmdir(destination_path)
    write_toml(records, incr_records_path)
```

Рисунок 15 – Листинг кода реализации функции добавочной архивации

В приведённых выше фрагментах кода за процесс архивации отвечает библиотека zipfile. Также, в функции каждого из режимов архивации, перед

непосредственно самим процессом архивации происходит вызов функции backup. Листинг кода данной функции показан на рисунке 16:

```
239 def backup(save_records: bool, path: Path, destination: Path, records: dict, save_acl: bool):
240     previous_path: Path = os.path.abspath(os.curdir)
241     if os.path.isdir(path):
242         os.chdir(path)
243 
244         for item in os.listdir():
245             item_path: Path = Path(os.path.abspath(path + "/" + item)).as_posix()
246             item_destination_path: Path = os.path.abspath(destination + "/" + item)
247             if os.path.isdir(item_path):
248                 os.makedirs(item_destination_path)
249                 prev: dict = backup(save_records, item_path, item_destination_path,
250                                     records, save_acl)
251                 if not save_records and len(os.listdir(item_destination_path)) == 0:
252                     os.rmdir(item_destination_path)
253 
254             else:
255                 compare: str = hashlib.md5(open(item_path, "rb").read()).hexdigest()
256 
257                 if save_records or (item_path not in records.keys()
258                     or records[item_path] != compare):
259 
260                     records[Path(item_path).as_posix()] = compare
261 
262                     if save_acl:
263                         subprocess.call(["robocopy", Path(item_path).parent,
264                                          Path(item_destination_path).parent,
265                                          Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
266                     else:
267                         copy(item_path, item_destination_path)
268     else:
269         item_path = Path(path)
270         item_destination_path: Path = os.path.abspath(destination +"/"+ Path(item_path).name)
271         compare: str = hashlib.md5(open(item_path, "rb").read()).hexdigest()
272 
273         if save_records or (item_path not in records.keys() or records[item_path] != compare):
274             records[Path(item_path).as_posix()] = compare
275 
276             if save_acl:
277                 subprocess.call(["robocopy", Path(item_path).parent,
278                                  Path(item_destination_path).parent,
279                                  Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
280             else:
281                 copy(item_path, item_destination_path)
282 
283 
284     os.chdir(previous_path)
```

Рисунок 16 – Листинг кода функции backup

Функция backup отвечает за копирование исходных файлов в директорию назначения, ведя при этом записи (records) и проводя этом «отсев» файлов в соответствии с выбранным режимом архивации.

Данная функция также вызывается внутри функции создания iso-образа, листинг кода которой показан на рисунке 17:

```
315 def iso_make(conf: dict, save_acl: bool):
316     working_dir: Path = os.path.abspath(os.curdir)
317     now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
318     records: dict = {}
319 
320     destination_path: Path = conf["destination"] + "ISO_" + now
321 
322     try:
323         os.makedirs(destination_path)
324     except FileExistsError:
325         pass
326 
327     if type(conf["source-directories"]) is list:
328         for path in conf["source-directories"]:
329             os.makedirs(destination_path + "/" + Path(path).name)
330             backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)
331     else:
332         path = conf["source-directories"]
333         os.makedirs(destination_path + "/" + Path(path).name)
334         backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)
335 
336     os.chdir(destination_path)
337     if save_acl:
338         tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
339         subprocess.call(["icacls", PureWindowsPath(destination_path),
340                          "/save", tmp_acl_path, "/t"])
341 
342     iso = pycdlib.PyCdlib()
343     iso.new(joliet=True, rock_ridge="1.09", interchange_level=1)
344     for root, dirs, files in os.walk(destination_path):
345         for dir in dirs:
346             for file in files:
347                 iso.add_file(os.path.join(root, file),joliet_path=f'/{file}',rr_name=file)
348     iso.write(f'{destination_path}.iso')
349     iso.close()
350     try:
351         Path(tmp_acl_path).unlink()
352     except:
353       pass
354 
355     for file in os.listdir(destination_path):
356         if os.path.isdir(file):
357             rmtree(destination_path + "/" + file)
358 
359     os.chdir(working_dir)
360     os.rmdir(destination_path)
```

Рисунок 17 – Листинг кода реализации функции создания iso-образа

В приведённом фрагменте кода создание iso-образа из исходных файлов производится при помощи библиотеки pycdlib.

На рисунке 18 показан листинг кода, реализующего функцию автоматической архивации:

```
190 def auto_zip(conf: dict, save_acl: bool):
191     now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
192 
193     working_dir: Path = os.path.abspath(os.curdir)
194     destination_path: Path = conf["destination"] + "Auto_" + now
195 
196     try:
197         os.makedirs(destination_path)
198     except FileExistsError:
199         pass
200 
201     if type(conf["source-directories"]) is list:
202         for path in conf["source-directories"]:
203             os.makedirs(destination_path + "/" + Path(path).name)
204             auto_backup(path, destination_path + "/" + Path(path).name, conf, save_acl)
205     else:
206         path = conf["source-directories"]
207         os.makedirs(destination_path + "/" + Path(path).name)
208         auto_backup(path, destination_path + "/" + Path(path).name, conf, save_acl)
209 
210     os.chdir(destination_path)
211     if save_acl:
212         tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
213         subprocess.call(["icacls", PureWindowsPath(destination_path),
214                          "/save", tmp_acl_path, "/t"])
215 
216     zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
217                                compression=zipfile.ZIP_DEFLATED, allowZip64=True)
218     for dir, subdir, files in os.walk(destination_path):
219         for file in files:
220             zip_file.write(os.path.join(dir, file),
221                            os.path.relpath(os.path.join(dir, file), destination_path))
222     zip_file.close()
223 
224     try:
225         Path(tmp_acl_path).unlink()
226     except:
227       pass
228 
229     for file in os.listdir(destination_path):
230         if os.path.isdir(file):
231             rmtree(destination_path + "/" + file)
232     conf["prev-arch"] = now
233     os.chdir(working_dir)
234     os.rmdir(destination_path)
235     write_toml(conf, auto_conf_path)
```

Рисунок 18 – Листинг кода реализации функции автоматической архивации

В теле данной функции вызывается функция auto_backup, которая копирует исходные файлы в директорию назначения, игнорируя те, что не попадают под критерии «старых» файлов.

На рисунке 19 показан листинг кода функции auto_backup.

```
286 def auto_backup(path: Path, destination: Path, conf: dict, save_acl: bool):
287     previous_path: Path = os.path.abspath(os.curdir)
288     os.chdir(path)
289     time_delta = conf["time-delta"]
290 
291     for item in os.listdir():
292         file_accessed = datetime.datetime.fromtimestamp(os.path.getatime(item))
293         item_path: Path = Path(os.path.abspath(path + "/" + item)).as_posix()
294         item_destination_path: Path = os.path.abspath(destination + "/" + item)
295         if os.path.isdir(item_path):
296             os.makedirs(item_destination_path)
297             prev: dict = auto_backup(item_path, item_destination_path, conf, save_acl)
298             if len(os.listdir(item_destination_path)) == 0:
299                 os.rmdir(item_destination_path)
300 
301         else:
302             if datetime.datetime.now() - file_accessed > datetime.timedelta(minutes = time_delta):
303                 if save_acl:
304                     subprocess.call(["robocopy", Path(item_path).parent,
305                                      Path(item_destination_path).parent,
306                                      Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
307                 else:
308                     copy(item_path, item_destination_path)
309 
310                 if conf['delete-source'] == True:
311                     Path(item_path).unlink()
312     os.chdir(previous_path)
```

Рисунок 19 – Листинг кода функции auto_backup

Перед тем как переходить к разработке интерфейса, осталось рассмотреть функцию восстановления (извлечения) файлов. Листинг её кода приведён на рисунке 20:

```
363 def extract_files(conf: dict):
364     path = Path(conf["source-directories"])
365     destination_path = conf["destination"]
366     target_folder = destination_path + "/" + Path(path).stem
367 
368     if path.is_file() and path.suffix == '.zip':
369         Path(destination_path).mkdir(parents=True, exist_ok=True)
370         zip_file = zipfile.ZipFile(path, mode='r')
371         zip_file.extractall(path=Path(target_folder))
372         zip_file.close
373 
374         tmp_acl_path = Path(target_folder + "/" + "tempACL.txt")
375         if os.path.exists(tmp_acl_path):
376             subprocess.call(["icacls", PureWindowsPath(destination_path),
377                              "/restore", tmp_acl_path])
378             Path(tmp_acl_path).unlink()
```

Рисунок 20 – Листинг кода функции восстановления файлов

В данной функции, по аналогии с функциями архивации, извлечение файлов из ZIP-архива производится с использованием библиотеки zipfile.

3.2 Разработка пользовательского интерфейса

Реализация пользовательского интерфейса приложения для архивации данных выполнена с использованием библиотеки PySimpleGUI [21].

На рисунке 21 показан итоговый вид приложения для архивации с пометками к элементам интерфейса:

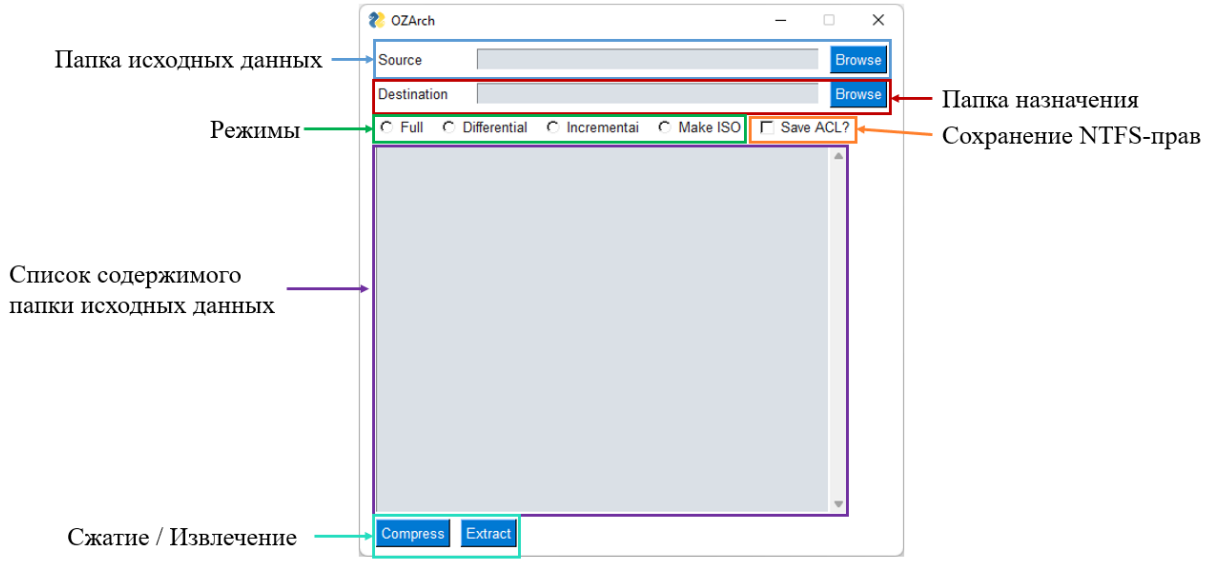


Рисунок 21 – Итоговый вид пользовательского интерфейса

При попытке закрыть программу, открывается диалоговое окно. Пользователю предоставляется выбор закрыть программу или свернуть на панель задач (в трей).

На рисунке 22 продемонстрировано контекстное меню иконки панели задач, в котором представлены возможности раскрыть основное окно, закрыть программу, а также запустить или остановить автоматическую архивацию в фоновом режиме .

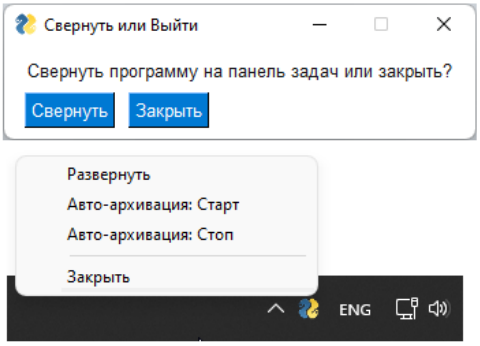


Рисунок 22 – Диалоговое окно выхода и контекстное меню иконки в трее

Листинг кода пользовательского интерфейса разбит на 3 части: планировка элементов основного окна и меню, инициализация основного окна и логика событий (event), события меню.

На рисунке 23 продемонстрирован листинг кода планировки элементов основного окна интерфейса и контекстного меню.

```
424         layout = [
425         [
426             sg.Text('Source', size=(10,1)),
427             sg.InputText(enable_events=True, key="-SOURCE-"),
428             sg.FolderBrowse()
429         ],
430 
431         [
432             sg.Text('Destination', size=(10,1)),
433             sg.InputText(enable_events=True, key="-DESTINATION-"),
434             sg.FolderBrowse()
435         ],
436 
437         [
438             sg.Radio('Full', 'type', key='-FULL-'),
439             sg.Radio('Differential', 'type', key='-DIFF-'),
440             sg.Radio('Incrementai', 'type', key='-INCR-' ),
441             sg.Radio('Make ISO', 'type', key='-ISO-' ),
442             sg.Checkbox('Save ACL?', key='-ACL-')
443         ],
444 
445         [
446             sg.Listbox(
447                 values = [], enable_events=True, size = (60,20),
448                 key="-FILE LIST-"
449             )
450         ],
451 
452         [
453             sg.Button('Compress'),
454             sg.Button('Extract')
455         ]
456         ]
457 
458         menu = ['',
459         [
460             'Развернуть',
461             'Авто-архивация: Старт',
462             'Авто-архивация: Стоп',
463             '---',
464             'Закрыть'
465         ]]
466 
```

Рисунок 23 – Листинг кода планировки элементов окна и меню

Листинг кода, реализующего инициализацию окна и логику событий показан на рисунке 24:

```
        window = sg.Window('OZArch', layout, enable_close_attempted_event=True)
        tray = SystemTray(menu, single_click_events=False, window=window,
                          tooltip ='OZArch', icon=sg.DEFAULT_BASE64_ICON)
        while True:
            event, values = window.read()
            save_acl = False

            if event == tray.key:
                event = values[event]

            if event in (sg.WIN_CLOSED, 'Закрыть'):
                break

            if event == sg.WIN_CLOSE_ATTEMPTED_EVENT:
                event, values = sg.Window('Свернуть или Выйти',
                                  [[sg.Text('Свернуть программу на панель задач или закрыть?')],
                                  [sg.Button('Свернуть'), sg.Button('Закрыть')]]).read(close=True)
                if event == 'Свернуть':
                    window.hide()
                    tray.show_icon()

                elif event == 'Закрыть':
                    try:
                        scheduler.shutdown()
                    except: break
                    break

            if event == "-SOURCE-":
                conf['source-directories'] = values["-SOURCE-"]
                try:
                    file_list = os.listdir(conf['source-directories'])
                except:
                    file_list = []
                window["-FILE LIST-"].update(file_list)

            elif event == "-FILE LIST-":
                try:
                    conf['source-directories'] = os.path.join(values["-SOURCE-"],
                                                              values["-FILE LIST-"][0])
                except:
                    print('ERROR')

            if event == "-DESTINATION-":
                conf["destination"] = str(values["-DESTINATION-"] + '/')

            if event == 'Compress':
                if values['-FULL-']:
                    full_zip(conf, values["-ACL-"])
                elif values['-DIFF-']:
                    diff_zip(conf, values["-ACL-"])
                elif values['-INCR-']:
                    incr_zip(conf, values["-ACL-"])
                elif values['-ISO-']:
                    iso_make(conf, values["-ACL-"])
                sg.popup('Operation Finished')

            if event == 'Extract':
                extract_files(conf)
                sg.popup('Operation Finished')
```

Рисунок 24 – Листинг кода инициализации окна и логики событий

На рисунке 25 показан листинг кода события меню иконки панели задач:

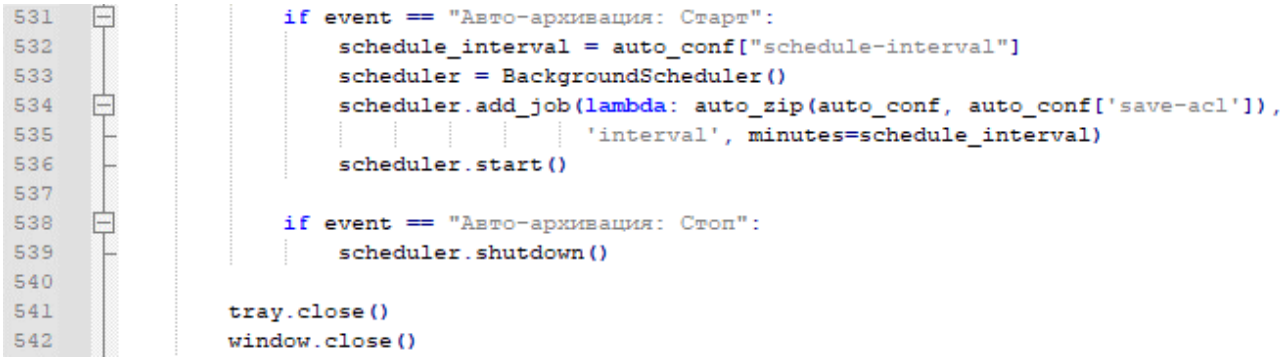


Рисунок 25 – Листинг кода событий меню иконки в трее

В событии «Авто-архивация: Старт» происходит инициализация фонового потока, в котором функция auto_zip выполняется по расписанию с определённым интервалом, реализованном при помощи библиотеки APScheduler [16].

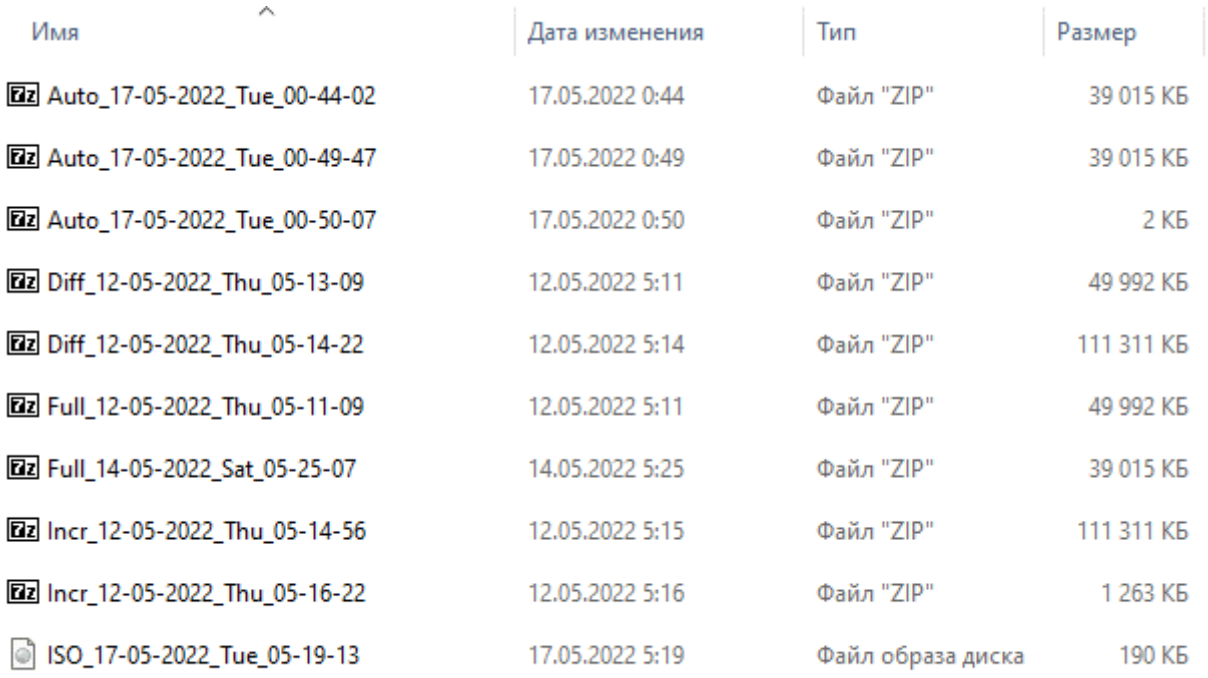
3.3 Тестирование разработанного приложения

Проверка работоспособности разработанного приложения осуществлена по методу функционального тестирования. Функциональное тестирование подразумевает под собой тестирование программного обеспечения, при котором проверяется соответствие разработанного приложения всем спецификациям / функциональным требованиям.

«Целью функциональных тестов является тестирование каждой функции программного приложения путем предоставления соответствующих входных данных и проверки выходных данных на соответствие функциональным требованиям. Функциональное тестирование в основном включает в себя тестирование черного ящика и не касается исходного кода приложения. Это тестирование проверяет пользовательский интерфейс, API, базу данных, безопасность, связь клиент / сервер и другие функции тестируемого приложения. Тестирование можно проводить вручную или с помощью автоматизации»[7].

Целью данного тестирования является исключительно проверка работоспособности всех функций архивации, согласно их алгоритмам.

На рисунке 26 приведены результаты тестирования в виде полученных архивов и iso-образа, в каждом из которых сохраняется исходная структура вложенности файлов, а степень сжатия архивов соответствует стандартам формата ZIP при использовании алгоритма сжатия DEFLATE.



Имя	Дата изменения	Тип	Размер
Auto_17-05-2022_Tue_00-44-02	17.05.2022 0:44	Файл "ZIP"	39 015 КБ
Auto_17-05-2022_Tue_00-49-47	17.05.2022 0:49	Файл "ZIP"	39 015 КБ
Auto_17-05-2022_Tue_00-50-07	17.05.2022 0:50	Файл "ZIP"	2 КБ
Diff_12-05-2022_Thu_05-13-09	12.05.2022 5:11	Файл "ZIP"	49 992 КБ
Diff_12-05-2022_Thu_05-14-22	12.05.2022 5:14	Файл "ZIP"	111 311 КБ
Full_12-05-2022_Thu_05-11-09	12.05.2022 5:11	Файл "ZIP"	49 992 КБ
Full_14-05-2022_Sat_05-25-07	14.05.2022 5:25	Файл "ZIP"	39 015 КБ
Incr_12-05-2022_Thu_05-14-56	12.05.2022 5:15	Файл "ZIP"	111 311 КБ
Incr_12-05-2022_Thu_05-16-22	12.05.2022 5:16	Файл "ZIP"	1 263 КБ
ISO_17-05-2022_Tue_05-19-13	17.05.2022 5:19	Файл образа диска	190 КБ

Рисунок 26 – Полученные в результате тестирования архивы и iso-образ

Таким образом, функциональное тестирование подтвердило полную работоспособность приложения.

Выводы к главе 3.

Третья глава посвящена программной реализации проектного решения и тестированию приложения.

В результате проделанной работы было разработано приложение для архивации данных, а функциональное тестирование подтвердило его работоспособность и возможность проведения автоматической архивации файлов в фоновом режиме.

Заключение

Выпускная квалификационная работа посвящена разработке приложения для архивации данных по заказу ООО «Управляющая компания» («Озон Фарм»).

В процессе выполнения ВКР были решены следующие задачи:

- проведено изучение теоретических основ архивации, включающее изучение некоторых существующих алгоритмов и типов архивации;
- основываясь на предъявленных заказчиком требованиях проведён подбор средств разработки, таких как: язык программирования, вспомогательные библиотеки и среда разработки; спроектированы основные функции и пользовательский интерфейс приложения для архивации данных;
- средствами языка программирования Python и его сторонних библиотек реализован программный код проектного решения;
- проведено функциональное тестирование разработанного приложения.

Функциональное тестирование подтвердило стабильную работоспособность приложения для архивации данных и возможность проведения с его помощью автоматической архивации неиспользуемых старых файлов в фоновом режиме, что в конечном итоге должно обеспечить повышение эффективности работы пользователя, за счёт избежания излишних временных затрат на ручную архивацию.

Результаты бакалаврской работы предоставляют научно-практический интерес и могут быть рекомендованы к использованию работниками отдела технической поддержки ООО «Управляющая компания» («Озон Фарм») с целью повышения эффективности работы отдела.

Список используемых источников

1. Алгоритмы архивации без потерь [Электронный ресурс] URL: https://intuit.ru/studies/courses/1069/206/lecture/5332 (дата обращения: 10.03.2022)
2. Буч Градди Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Буч Градди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. – М.: ООО “И.Д. Вильямс”, 2010. – 720 с.
3. Графический интерфейс на Python [Электронный ресурс] URL: https://habr.com/ru/company/edison/blog/480884/ (дата обращения: 26.03.2022)
4. Документация icacls [Электронный ресурс] URL: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls (дата обращения 7.04.2022)
5. Е.В. Михальчик Описание формата сжатия данных Deflate [Электронный ресурс] URL: http://compression.ru/download/articles/lz/mihalchik_deflate_decoding.html (дата обращения 10.03.2022)
6. Интегрированные среды разработки программ [Электронный ресурс]. URL: http://bourabai.ru/einf/ide.htm (дата обращения: 15.03.2022).
7. Липаев В. В. Тестирование компонентов и комплексов программ: учебник. Москва: СИНТЕГ, 2010. – 393 с.
8. Лучшие IDE и редакторы кода для Python [Электронный ресурс]. URL: https://tproger.ru/translations/python-ide/ (дата обращения: 15.03.2022).
9. Ненов А. Л. Разработка мультимедийных систем. Учебное пособие. Одесская государственная академия холода, 2012. – 81 с.
10. Обзор файловой системы NTFS [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/windows-server/storage/file-server/ntfs-overview (дата обращения 7.04.2022)
11. Подстановочные или словарно-ориентированные алгоритмы сжатия информации. Методы Лемпела-Зива [Электронный ресурс] URL: https://intuit.ru/studies/courses/2256/140/lecture/3914 (дата обращения 10.03.2022)

12. Проблемы алгоритмов архивации с потерями [Электронный ресурс] URL: https://intuit.ru/studies/courses/1580/206/lecture/5334 (дата обращения 10.03.2022)

13. Язык программирования Python [Электронный ресурс]. URL: https://intuit.ru/studies/courses/49/49/info (дата обращения: 20.02.2022).

14. A. Moffat, A. Turpin Compression and Coding Algorithms, 2002. – 422 p.

15. Antaeus Feldspar An Explanation of the Deflate Algorithm [Электронный ресурс] URL: https://zlib.net/feldspar.html (дата обращения 10.03.2022)

16. APScheduler library user guide [Электронный ресурс] URL: https://apscheduler.readthedocs.io/en/stable/userguide.html (дата обращения: 26.02.2022)

17. Jerzy A. Seidler Information Systems and Data Compression, 2013. – 494 p.

18. Maan Hameed Low Power Approach for Implementation of Huffman Coding: For High Data Compression, 2018. – 56 p.

19. Pathlib library documentation [Электронный ресурс] URL: https://docs.python.org/3/library/pathlib.html (дата обращения 17.03.2022)

20. PyCdlib library documentation [Электронный ресурс] URL:https://clalancette.github.io/pycdlib/ (дата обращения: 24.03.2022)

21. PySimpleGUI library documentation [Электронный ресурс] URL: https://pysimplegui.readthedocs.io/en/latest/ (дата обращения: 26.03.2022)

22. Python 3.10.4 documentation [Электронный ресурс] URL: https://docs.python.org/3.10 (дата обращения: 20.02.2022)

23. TOML documentation [Электронный ресурс] URL: https://github.com/toml-lang/toml/blob/main/README.md (дата обращения: 24.03.2022)

24. Zed. A. Shaw Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code, First Edition, 2017. – 320 p.

25. Zipfile library documentation [Электронный ресурс] URL: https://docs.python.org/3/library/zipfile.html (дата обращения: 24.03.2022)

Приложение А

Фрагменты программного кода

```
from argparse import ArgumentParser
import datetime
import hashlib
import os
import sys
import zipfile
import pycdlib
from shutil import copy2 as copy, rmtree
import subprocess
import toml
from pathlib import Path, PureWindowsPath
import PySimpleGUI as sg
from psgtray import SystemTray
from apscheduler.schedulers.background import BackgroundScheduler

if getattr(sys, 'frozen', False):
    application_path = os.path.dirname(sys.executable)
elif __file__:
    application_path = os.path.dirname(__file__)
os.chdir(application_path)

conf = None

conf_path: Path = "./conf.toml"
auto_conf_path: Path = "./auto_conf.toml"
full_records_path: Path = "./Records/full_records.toml"
incr_records_path: Path = "./Records/incr_records.toml"
```

```
    def write_toml(d: dict, path: Path):
        with open(path, "w", encoding = 'utf-8') as f:
            toml.dump(d, f)



    def full_zip(conf: dict, save_acl: bool):
        working_dir: Path = os.path.abspath(os.curdir)
        now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-
%S")
        records: dict = {}

        destination_path: Path = conf["destination"] + "Full_" + now

        try:
            os.makedirs(destination_path)
        except FileExistsError:
            pass

        if type(conf["source-directories"]) is list:
            for path in conf["source-directories"]:
                os.makedirs(destination_path + "/" + Path(path).name)
                backup(True, path, destination_path + "/" + Path(path).name, records,
save_acl)
        else:
            path = conf["source-directories"]
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(True, path, destination_path + "/" + Path(path).name, records,
save_acl)
```

```
    os.chdir(destination_path)
    if save_acl:
        tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
        subprocess.call(["icacls", PureWindowsPath(destination_path),
                         "/save", tmp_acl_path, "/t"])

    zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                               compression=zipfile.ZIP_DEFLATED, allowZip64=True)
    for dir, subdir, files in os.walk(destination_path):
            for file in files:
                zip_file.write(os.path.join(dir, file),
                               os.path.relpath(os.path.join(dir, file), destination_path))
    zip_file.close()

    try:
        Path(tmp_acl_path).unlink()
    except:
      pass

    for file in os.listdir(destination_path):
        if os.path.isdir(file):
            rmtree(destination_path + "/" + file)

    os.chdir(working_dir)
    os.rmdir(destination_path)
    write_toml(records, full_records_path)


def diff_zip(conf: dict, save_acl: bool):
```

```
        working_dir: Path = os.path.abspath(os.curdir)
        now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-
%S")
        records: dict = toml.load(full_records_path)

        destination_path: Path = conf["destination"] + "Diff_" + now

        try:
            os.makedirs(destination_path)
        except FileExistsError:
            pass

        if type(conf["source-directories"]) is list:
            for path in conf["source-directories"]:
                os.makedirs(destination_path + "/" + Path(path).name)
                backup(False, path, destination_path + "/" + Path(path).name, records,
save_acl)
        else:
            path = conf["source-directories"]
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(False, path, destination_path + "/" + Path(path).name, records,
save_acl)

        os.chdir(destination_path)
        if save_acl:
            tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
            subprocess.call(["icacls", PureWindowsPath(destination_path),
                             "/save", tmp_acl_path, "/t"])
```

```
        zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                                   compression=zipfile.ZIP_DEFLATED, allowZip64=True)
        for dir, subdir, files in os.walk(destination_path):
            for file in files:
                zip_file.write(os.path.join(dir, file),
                               os.path.relpath(os.path.join(dir, file), destination_path))
        zip_file.close()

        try:
            Path(tmp_acl_path).unlink()
        except:
          pass

        for file in os.listdir(destination_path):
            if os.path.isdir(file):
                rmtree(destination_path + "/" + file)

        os.chdir(working_dir)
        os.rmdir(destination_path)


    def incr_zip(conf: dict, save_acl: bool):
        working_dir: Path = os.path.abspath(os.curdir)
        now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-
%S")
        compare: str = hashlib.md5(open(full_records_path, "rb").read()).hexdigest()
        try:
            records: dict = toml.load(incr_records_path)
            if records[full_records_path] != compare:
```

```
            records: dict = toml.load(full_records_path)
            records[full_records_path] = compare
        except:
            records: dict = toml.load(full_records_path)
            records[full_records_path] = compare

        destination_path: str = conf["destination"] + "Incr_" + now

        try:
            os.makedirs(destination_path)
        except FileExistsError:
            pass

        if type(conf["source-directories"]) is list:
            for path in conf["source-directories"]:
                os.makedirs(destination_path + "/" + Path(path).name)
                backup(False, path, destination_path + "/" + Path(path).name, records,
save_acl)
        else:
            path = conf["source-directories"]
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(False, path, destination_path + "/" + Path(path).name, records,
save_acl)

        os.chdir(destination_path)
        if save_acl:
            tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
            subprocess.call(["icacls", PureWindowsPath(destination_path),
                             "/save", tmp_acl_path, "/t"])
```

```
        zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                                   compression=zipfile.ZIP_DEFLATED, allowZip64=True)
        for dir, subdir, files in os.walk(destination_path):
            for file in files:
                zip_file.write(os.path.join(dir, file),
                               os.path.relpath(os.path.join(dir, file), destination_path))
        zip_file.close()

        try:
            Path(tmp_acl_path).unlink()
        except:
          pass

        for file in os.listdir(destination_path):
            if os.path.isdir(file):
                rmtree(destination_path + "/" + file)

        os.chdir(working_dir)
        os.rmdir(destination_path)
        write_toml(records, incr_records_path)


    def auto_zip(conf: dict, save_acl: bool):
        now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")

        working_dir: Path = os.path.abspath(os.curdir)
        destination_path: Path = conf["destination"] + "Auto_" + now
```

```
        try:
            os.makedirs(destination_path)
        except FileExistsError:
            pass

        if type(conf["source-directories"]) is list:
            for path in conf["source-directories"]:
                os.makedirs(destination_path + "/" + Path(path).name)
                auto_backup(path, destination_path + "/" + Path(path).name, conf,
save_acl)
        else:
            path = conf["source-directories"]
            os.makedirs(destination_path + "/" + Path(path).name)
            auto_backup(path, destination_path + "/" + Path(path).name, conf,
save_acl)

        os.chdir(destination_path)
        if save_acl:
            tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
            subprocess.call(["icacls", PureWindowsPath(destination_path),
                             "/save", tmp_acl_path, "/t"])

        zip_file = zipfile.ZipFile(str(destination_path + '.zip'), mode='w',
                                   compression=zipfile.ZIP_DEFLATED, allowZip64=True)
        for dir, subdir, files in os.walk(destination_path):
            for file in files:
                zip_file.write(os.path.join(dir, file),
                               os.path.relpath(os.path.join(dir, file), destination_path))
        zip_file.close()
```

```
        try:
            Path(tmp_acl_path).unlink()
        except:
          pass

        for file in os.listdir(destination_path):
            if os.path.isdir(file):
                rmtree(destination_path + "/" + file)
        conf["prev-arch"] = now
        os.chdir(working_dir)
        os.rmdir(destination_path)
        write_toml(conf, auto_conf_path)


    def backup(save_records: bool, path: Path, destination: Path, records: dict,
save_acl: bool):
        previous_path: Path = os.path.abspath(os.curdir)
        if os.path.isdir(path):
            os.chdir(path)

            for item in os.listdir():
                item_path: Path = Path(os.path.abspath(path + "/" + item)).as_posix()
                item_destination_path: Path = os.path.abspath(destination + "/" + item)
                if os.path.isdir(item_path):
                    os.makedirs(item_destination_path)
                    prev: dict = backup(save_records, item_path, item_destination_path,
                                        records, save_acl)
                    if not save_records and len(os.listdir(item_destination_path)) == 0:
                        os.rmdir(item_destination_path)
```

```
            else:
                compare: str = hashlib.md5(open(item_path, "rb").read()).hexdigest()

                if save_records or (item_path not in records.keys()
                    or records[item_path] != compare):

                    records[Path(item_path).as_posix()] = compare

                    if save_acl:
                        subprocess.call(["robocopy", Path(item_path).parent,
                                         Path(item_destination_path).parent,
                                         Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
                    else:
                        copy(item_path, item_destination_path)
        else:
            item_path = Path(path)
            item_destination_path: Path = os.path.abspath(destination +"/"+
Path(item_path).name)
            compare: str = hashlib.md5(open(item_path, "rb").read()).hexdigest()

            if save_records or (item_path not in records.keys() or records[item_path]
!= compare):
                records[Path(item_path).as_posix()] = compare

                if save_acl:
                    subprocess.call(["robocopy", Path(item_path).parent,
                                     Path(item_destination_path).parent,
                                     Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
```

```
                else:
                    copy(item_path, item_destination_path)

            os.chdir(previous_path)

        def auto_backup(path: Path, destination: Path, conf: dict, save_acl: bool):
            previous_path: Path = os.path.abspath(os.curdir)
            os.chdir(path)
            time_delta = conf["time-delta"]

            for item in os.listdir():
                file_accessed = datetime.datetime.fromtimestamp(os.path.getatime(item))
                item_path: Path = Path(os.path.abspath(path + "/" + item)).as_posix()
                item_destination_path: Path = os.path.abspath(destination + "/" + item)
                if os.path.isdir(item_path):
                    os.makedirs(item_destination_path)
                    prev: dict = auto_backup(item_path, item_destination_path, conf,
save_acl)
                    if len(os.listdir(item_destination_path)) == 0:
                        os.rmdir(item_destination_path)
                else:
                    if datetime.datetime.now() - file_accessed > datetime.timedelta(minutes
= time_delta):
                        if save_acl:
                            subprocess.call(["robocopy", Path(item_path).parent,
                                             Path(item_destination_path).parent,
                                             Path(item_path).name, '/e','/MIR','/SEC','/SECFIX'])
```

```
                else:
                    copy(item_path, item_destination_path)

                if conf['delete-source'] == True:
                    Path(item_path).unlink()
        os.chdir(previous_path)

    def iso_make(conf: dict, save_acl: bool):
        working_dir: Path = os.path.abspath(os.curdir)
        now: str = datetime.datetime.now().strftime("%d-%m-%Y_%a_%H-%M-%S")
        records: dict = {}
        destination_path: Path = conf["destination"] + "ISO_" + now

        try:
            os.makedirs(destination_path)
        except FileExistsError:
            pass

        if type(conf["source-directories"]) is list:
            for path in conf["source-directories"]:
                os.makedirs(destination_path + "/" + Path(path).name)
                backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)
        else:
            path = conf["source-directories"]
            os.makedirs(destination_path + "/" + Path(path).name)
            backup(True, path, destination_path + "/" + Path(path).name, records, save_acl)
```

```
            os.chdir(destination_path)
            if save_acl:
                tmp_acl_path = Path(str(destination_path+"\\tempACL.txt"))
                subprocess.call(["icacls", PureWindowsPath(destination_path),
                                 "/save", tmp_acl_path, "/t"])

            iso = pycdlib.PyCdlib()
            iso.new(joliet=True, rock_ridge="1.09", interchange_level=1)
            for root, dirs, files in os.walk(destination_path):
                for dir in dirs:
                    for file in files:
                        iso.add_file(os.path.join(root,
file),joliet_path=f'/{file}',rr_name=file)
            iso.write(f'{destination_path}.iso')
            iso.close()
            try:
                Path(tmp_acl_path).unlink()
            except:
              pass

            for file in os.listdir(destination_path):
                if os.path.isdir(file):
                    rmtree(destination_path + "/" + file)

            os.chdir(working_dir)
            os.rmdir(destination_path)

        def extract_files(conf: dict):
            path = Path(conf["source-directories"])
```

```
    destination_path = conf["destination"]
    target_folder = destination_path + "/" + Path(path).stem

    if path.is_file() and path.suffix == '.zip':
        Path(destination_path).mkdir(parents=True, exist_ok=True)
        zip_file = zipfile.ZipFile(path, mode='r')
        zip_file.extractall(path=Path(target_folder))
        zip_file.close

        tmp_acl_path = Path(target_folder + "/" + "tempACL.txt")
        if os.path.exists(tmp_acl_path):
            subprocess.call(["icacls", PureWindowsPath(destination_path),
                             "/restore", tmp_acl_path])
            Path(tmp_acl_path).unlink()

if __name__ == "__main__":
    conf: dict = toml.load(conf_path)
    auto_conf: dict = toml.load(auto_conf_path)

        sg.theme('Reddit')
        layout = [
        [
            sg.Text('Source', size=(10,1)),
            sg.InputText(enable_events=True, key="-SOURCE-"),
            sg.FolderBrowse()
        ],

        [
            sg.Text('Destination', size=(10,1)),
```

```
    sg.InputText(enable_events=True, key="-DESTINATION-"),
    sg.FolderBrowse()
],

[
    sg.Radio('Full', 'type', key='-FULL-'),
    sg.Radio('Differential', 'type', key='-DIFF-'),
    sg.Radio('Incrementai', 'type', key='-INCR-' ),
    sg.Radio('Make ISO', 'type', key='-ISO-' ),
    sg.Checkbox('Save ACL?', key='-ACL-')
],

[
    sg.Listbox(
        values = [], enable_events=True, size = (60,20),
        key="-FILE LIST-"
    )
],

[
    sg.Button('Compress'),
    sg.Button('Extract')
]
]

menu = ['',
[
    'Развернуть',
    'Авто-архивация: Старт',
```

```
            'Авто-архивация: Стоп',
            '---',
            'Закрыть'
        ]]

        window = sg.Window('OZArch', layout, enable_close_attempted_event=True)
        tray = SystemTray(menu, single_click_events=False, window=window,
                          tooltip ='OZArch', icon=sg.DEFAULT_BASE64_ICON)
        while True:
            event, values = window.read()
            save_acl = False

            if event == tray.key:
                event = values[event]

            if event in (sg.WIN_CLOSED, 'Закрыть'):
                break

            if event == sg.WIN_CLOSE_ATTEMPTED_EVENT:
                event, values = sg.Window('Свернуть или Выйти',
                                  [[sg.Text('Свернуть программу на панель задач или закрыть?')],
                                  [sg.Button('Свернуть'),
sg.Button('Закрыть')]]).read(close=True)

                if event == 'Свернуть':
                    window.hide()
```

```
            tray.show_icon()

        elif event == 'Закрыть':
            try:
                scheduler.shutdown()
            except: break
            break

    if event in ('Развернуть', sg.EVENT_SYSTEM_TRAY_ICON_DOUBLE_CLICKED):
        window.un_hide()
        window.bring_to_front()

    if event == "Авто-архивация: Старт":
        schedule_interval = auto_conf["schedule-interval"]
        scheduler = BackgroundScheduler()
        scheduler.add_job(lambda: auto_zip(auto_conf, auto_conf['save-acl']),
                          'interval', minutes=schedule_interval)
        scheduler.start()
    if event == "Авто-архивация: Стоп":
        scheduler.shutdown()

    if event == "-SOURCE-":
        conf['source-directories'] = values["-SOURCE-"]
        try:
            file_list = os.listdir(conf['source-directories'])
        except:
            file_list = []
```

```
            window["-FILE LIST-"].update(file_list)

        elif event == "-FILE LIST-":
            try:
                conf['source-directories'] = os.path.join(values["-SOURCE-"],
                                                          values["-FILE LIST-"][0])
            except:
                print('ERROR')

        if event == "-DESTINATION-":
            conf["destination"] = str(values["-DESTINATION-"] + '/')

        if event == 'Compress':
            if values['-FULL-']:
                full_zip(conf, values["-ACL-"])
            elif values['-DIFF-']:
                diff_zip(conf, values["-ACL-"])
            elif values['-INCR-']:
                incr_zip(conf, values["-ACL-"])
            elif values['-ISO-']:
                iso_make(conf, values["-ACL-"])
            sg.popup('Operation Finished')

        if event == 'Extract':
            extract_files(conf)
            sg.popup('Operation Finished')

    tray.close()
    window.close()
```