СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc168412018)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc168412019)

[2 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc168412020)

[2.1 Анализ существующих аналогов 7](#_Toc168412021)

[2.1.1 Сhkdsk 7](#_Toc168412022)

[2.1.2 System File Checker (SFC) 8](#_Toc168412023)

[2.1.3 PowerShell командлет Repair-Volume 8](#_Toc168412024)

[2.2 Необходимые теоретические сведения 9](#_Toc168412025)

[3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc168412028)

[3.1 Блок запуска утилиты 14](#_Toc168412029)

[3.2 Блок анализа 14](#_Toc168412030)

[3.3 Блок журналирования 14](#_Toc168412031)

[3.4 Блок пользовательского интерфейса 14](#_Toc168412032)

[4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 15](#_Toc168412033)

[4.1 Описание используемых структур 15](#_Toc168412034)

[4.2 Описание используемых функций 18](#_Toc168412035)

[5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 22](#_Toc168412036)

[5.1 Разработка алгоритмов 22](#_Toc168412037)

[6 РЕЗУЛЬТАТЫ РАБОТЫ 25](#_Toc168412038)

[7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 29](#_Toc168412039)

[ЗАКЛЮЧЕНИЕ 30](#_Toc168412040)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 31](#_Toc168412041)

[ПРИЛОЖЕНИЕ А 32](#_Toc168412042)

[ПРИЛОЖЕНИЕ Б 33](#_Toc168412043)

[ПРИЛОЖЕНИЕ В 34](#_Toc168412044)

[ПРИЛОЖЕНИЕ Г 35](#_Toc168412045)

# ВВЕДЕНИЕ

Файловая система является ключевым элементом операционных систем, определяющим способ организации и хранения данных на диске. Файловая система NTFS (New Technology File System) является одной из наиболее распространенных и широко используемых файловых систем в операционных системах семейства Windows. Целостность файловой системы играет ключевую роль в обеспечении надежности и безопасности хранения данных.

В данном курсовом проекте мы рассмотрим основные принципы и особенности работы файловой системы NTFS. Будут изучены ее основные характеристики, архитектура, методы организации данных и множество других аспектов, которые пригодятся для успешного взаимодействия с ней.

Проверка целостности файловой системы NTFS представляет собой важную процедуру, направленную на обнаружение повреждений или несоответствий в структуре файлов, каталогов и метаданных. Проведение анализа и решение выявленных проблем целостности NTFS-файловой системы имеет критическое значение для предотвращения потери данных, обеспечения непрерывной работы системы и поддержания высокого уровня безопасности информации.

# 1 ПОСТАНОВКА ЗАДАЧИ

Разработка утилиты для проверки целостности файловой системы NTFS. Программа будет производить проверку определённого раздела файловой системы по выбору пользователя. Утилита должна иметь минималистический интерфейс с целью оповещения пользователя о процессе анализа. Также программа будет принимать аргументы, управляющие её работой.

Программа будет иметь следующий функционал:

1. Вывод информации о структуре тома.

2. Проведение анализа и выявление несоответствий.

3. Логирование процедуры проверки.

Для выполнения поставленной задачи будут использоваться средства WinAPI и возможности языка C++.

# 2 ОБЗОР ЛИТЕРАТУРЫ

## 2.1 Анализ существующих аналогов

Анализ существующих аналогов позволяет выделить основной функционал и требования, уникальные моменты в каждой утилите и учесть их при разрабатывании собственной утилиты.

## 2.1.1 Сhkdsk

Встроенная утилита Chkdsk.exe (check disk) используется в Windows для проверки диска на ошибки. Сhkdsk проверяет файловую систему на физические и логические ошибки, находит поврежденные секторы (bad sectors) и исправляет найденные проблемы. Утилита поддерживает список параметров:

1. /F – Исправляет обнаруженные ошибки на диске.

2./R – Проводит обширное сканирование и восстановление поврежденных секторов на диске.

3. /X – Закрывает все открытые файлы на диске перед началом операции проверки.

4. /V – Подробный вывод информации о выполнении проверки.

5. /B – Перезагружает компьютер и выполняет chkdsk перед загрузкой операционной системы.

После окончания проверки диска вы увидите подробную статистику диска, информацию о поврежденных секторах и файлах, предпринятых действиях по восстановлению. Работа утилиты представлена на рис. 1.1.

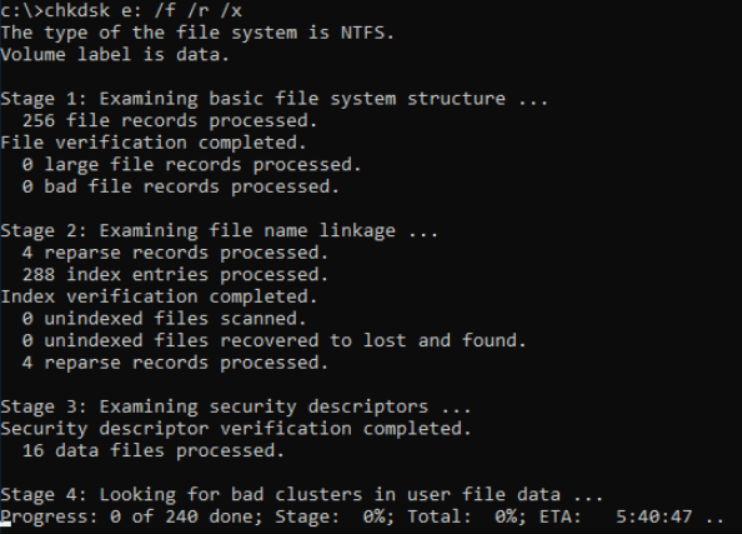


Рисунок 2.1.1 — Скриншот работы утилиты Сhkdsk

## 2.1.2 System File Checker (SFC)

Утилита SFC (System File Checker) представляет собой инструмент в Windows, который используется для проверки целостности и восстановления системных файлов операционной системы. Она запускается из командной строки с повышенными привилегиями в ОС Windows. Однако при использовании многозагрузочных конфигураций (если на одном компьютере установлены 2 версии Windows) утилита System File Checker не находит операционную систему, которую необходимо восстановить. Работа утилиты представлена на рис. 1.2.

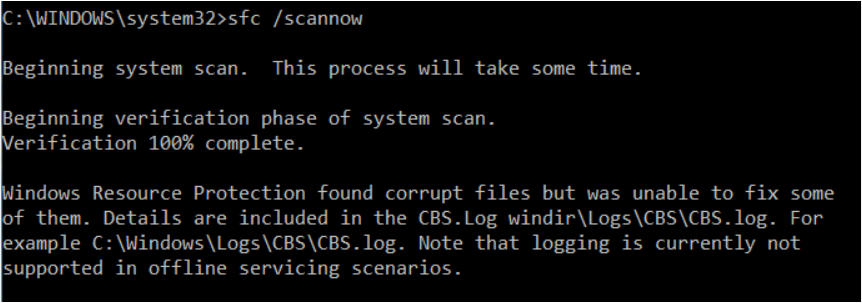


Рисунок 2.1.2 — Скриншот работы утилиты SFC

## 2.1.3 ****PowerShell командлет Repair-Volume****

Как следует из названия командлета, Repair-Volume выполняет ремонт тома. Он поставляется с опциями, соответствующими существующим возможностям chkdsk. Он позволяет проводить проверку целостности тома и его восстановление. Все обнаруженные повреждения добавляются в системный файл $Corrupt. На рис. 1.3 представлена проверка целостности системного диска C:/.

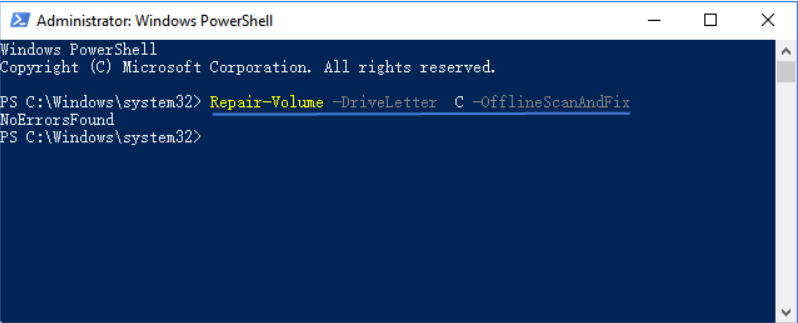


Рисунок 2.1.3 – Скриншот работы командлета Repair-Volume

## 2.2 Необходимые теоретические сведения

# 2.2.1 Файловая система NTFS

Файловая система NTFS представляет собой выдающееся достижение структуризации: **каждый** элемент системы представляет собой файл — даже служебная информация. Самый главный файл на NTFS называется MFT, или Master File Table — общая таблица файлов. Именно он размещается в MFT зоне и представляет собой централизованный каталог всех остальных файлов диска, и, как не парадоксально, себя самого. MFT поделен на записи фиксированного размера (обычно 1 Кбайт), и каждая запись соответствует какому-либо файлу. Первые 16 файлов носят служебный характер и недоступны операционной системе — они называются метафайлами, причем самый первый метафайл — сам MFT. Эти первые 16 элементов MFT — единственная часть диска, имеющая фиксированное положение.

# 2.2.2 Блоки данных в файловой системе NTFS

Как и любая другая система, NTFS делит все полезное место на кластеры – блоки данных, используемые единовременно. NTFS поддерживает почти любые размеры кластеров — от 512 байт до 64 Кбайт, неким стандартом же считается кластер размером 4 Кбайт.

Кластер – это блок, в котором система хранит информацию в файловой системе. Вся файловая система состоит из большого количества этих блоков, каждый из которых содержит в себе определенное количество данных. Размер кластера не влияет на объем диска, но он может повлиять на то, как система работает с файлами на вашем носителе и насколько эффективно использует доступное ей пространство.

**2.2.3 Структура файловой системы NTFS**

Диск NTFS условно делится на две части. Первые 12.5% диска отводятся под так называемую MFT зону — пространство, в которое растет метафайл MFT. Запись каких-либо данных в эту область невозможна. MFT-зона всегда держится пустой — это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 87.5% диска представляют собой обычное пространство для хранения файлов.

На рис. 2.2.1 представлена схема диска NFTS.

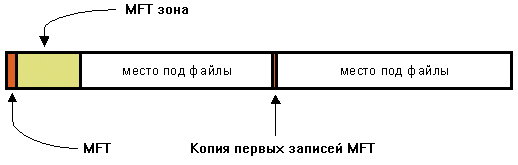


Рисунок 2.2.1 – Схема диска NTFS

Интересно, что вторая копия первых трех записей, для надежности — они очень важны — хранится ровно посередине диска. Остальной MFT-файл может располагаться, как и любой другой файл, в произвольных местах диска — восстановить его положение можно с помощью его самого, «зацепившись» за самую основу — за первый элемент MFT.

**2.2.4 Метафайлы в файловой системе NTFS**

Первые 16 файлов NTFS (метафайлы) носят служебный характер. Каждый из них отвечает за какой-либо аспект работы системы.

Метафайлы находятся корневом каталоге NTFS диска — они начинаются с символа имени «$», хотя получить какую-либо информацию о них стандартными средствами сложно. Любопытно, что и для этих файлов указан вполне реальный размер — можно узнать, например, сколько операционная система тратит на каталогизацию всего вашего диска, посмотрев размер файла $MFT.

Первые 16 записей MFT резервируются именно для этих файлов. Каждая из этих записей описывает нормальный файл, который имеет атрибуты и блоки данных. Каждый из этих файлов имеет имя, которое начинается со знака доллара (чтобы обозначить его как файл метаданных). Первая запись описывает сам файл MFT. В частности, в ней говорится, где находятся блоки файла MFT (чтобы система могла найти файл MFT). Очевидно, что Windows нужен способ нахождения первого блока файла MFT, чтобы найти остальную информацию по файловой системе. Windows смотрит в загрузочном блоке — именно туда записывается адрес первого блока файла MFT при форматировании тома.

Запись 1 ($MFTMirr) является дубликатом начала файла MFT. Эта информация настолько ценная, что наличие второй копии может быть просто критическим (в том случае, если один из первых блоков MFT перестанет читаться). Вторая запись — файл журнала ($LogFile). Запись 3 ($Volume) содержит информацию о томе (такую, как его размер, метка и версия). Как уже упоминалось, каждая запись MFT содержит последовательность пар «заголовок атрибута — значение». Атрибуты определяются в файле $AttrDef. Информация об этом файле содержится в 4 записи MFT. Затем идет корневой каталог, который сам является файлом и может расти до произвольного размера. Он описывается записью номер 5 в MFT.

Свободное пространство тома отслеживается при помощи битового массива. Сам битовый массив — тоже файл, его атрибуты и дисковые адреса даны в записи 6 ($Bitmap) в MFT. Следующая запись MFT ($Boot) указывает на файл начального загрузчика. Запись 8 ($BadClus) используется для того, чтобы связать вместе все плохие блоки (чтобы обеспечить невозможность их использования для файлов). Запись 9 ($Secure) содержит информацию безопасности. Запись 10 ($Upcase) используется для установления соответствия регистра. И наконец, запись 11 ($Extend) — это каталог, содержащий различные файлы для таких вещей, как дисковые квоты, идентификаторы объектов, точки повторной обработки и т. д. Последние четыре записи MFT зарезервированы для использования в будущем.

**2.2.5 Атрибуты записей MFT**

NTFS определяет 13 атрибутов, которые могут появиться в записях MFT:

1. $STANDART\_INFORMATION – общая информация: биты флагов, временные метки и т.д.;

2. $ATTRIBUTE\_LIST – список других атрибутов файла;

3. $FILE\_NAME - имя файла в Unicode и другая информация о файле;

4. $SECURITY\_DESCRIPTOR ­– время обращения и свойства безопасности файла;

5. $VOLUME\_NAME – имя тома

6. $OBJECT\_ID - уникальный для данного тома 64-битный идентификатор файла;

7. $VOLUME\_INFORMATION ­– версия файловой системы и др.;

Reparse point - используется для монтирования и символических ссылок;

8. $DATA – содержимое файла;

9. $INDEX\_ROOT – корневой узел индексного дерева;

10. $INDEX\_ALLOCATION ­– узлы индексного дерева;

11. $BITMAP – битовая карта файла $MFT и его индексов;

12. $REPARSE\_POINT – данные точек подключения, выполняющие функции мягких ссылок;

13. $LOGGED\_UTILITY\_STREAM – содержит ключи и информацию о зашифрованных атрибутах.

**2.2.6 Резидентные и нерезидентные атрибуты**

После нахождения записи MFT, самой важной задачей является поиск необходимого атрибута, например с данными. Атрибуты бывают двух видов: резидентные (resident) и нерезидентные (non-resident). Резидентный атрибут умещается в записи MFT, а нерезидентный нет.

В заголовке MFT записи хранится байтовое смещение первого атрибута, относительно заголовка записи. Прибавляя это смещение к смещению записи, мы получим смещение первого атрибута. Если все атрибуты для файла не вмещаются в одну MFT запись, тогда для файла создаются расширенные записи (extra records). В таком случае основная (первичная) запись называется базовой и хранит атрибут $ATTRIBUTE\_LIST, в котором хранятся ссылки на расширенные файловые записи.

Резидентные атрибуты хранят свое тело в записи MFT и для них флаг non\_resident в заголовке установлен в ноль. Для считывания данных такого атрибута достаточно определить смещение тела как сумму смещений заголовка атрибута и поля r.value\_offset, а затем считать r.value\_length байт в память.

Для нерезидентных атрибутов флаг non\_resident установлен в 1 и их тела хранятся в отдельных кластерах, на которые указывают отрезки. Отрезок (run) хранит цепочки кластеров, в которых находится содержимое атрибута. Массив отрезков называется списком отрезков (run list).

**2.2.7 Списки отрезков**

Тела нерезидентных атрибутов дефрагментированы и хранятся в разных кластерах. На эти кластеры указывают так называемые списки отрезков, которые представляют собой пару offset – length, где offset – смещение начала отрезка в кластерах, а length – длина этого отрезка. В самой системе эти отрезки представлены последовательностью байт.

Фактически, список отрезков – это массив структур переменного размера. Размер каждого из полей структуры указывается в предыдущем байте. Первый элемент структуры содержит размер отрезка в кластерах, а второй номер кластера. Байт, описывающий размеры полей размера и номера кластеров называется байтом длин. Младший полубайт байта длин содержит длину поля размера, а старший длину поля номера кластера. Данные в полях структуры хранятся в формате Intel, т. е. младший байт по младшему адресу.

Пусть нерезидентный атрибут имеет список отрезков, представленный на рис. 2.2.2.

32 90 3A 00 00 0C | 32 30 0F DA A7 1B | 32 A0 36 5E 89 05 | 00

Рисунок 2.2.2 – Список отрезков нерезидентного атрибута

Первый байт – байт длин описывает длины полей первого отрезка. Младший полубайт равен 2, значит на поле длины приходится два байта и длина отрезка равна 0x3A90. Далее, старший полубайт байта длин равен трем и стартовый кластер равен 0xC0000. Получаем, что первый отрезок начинается с кластера 0xC0000 и заканчивается границей 0xC0000 + 0x3A90 = C3A90. Для перехода к следующему элементу следует прибавить размеры полей, и единицу для байта длин, т. е. 3 + 2 + 1. Для второго отрезка по байту длин видно, что размер полей такой же как в предыдущем отрезке, т. е. младший полубайт равен двойке, значит размер поля длины два и равен 0xF30, старший полубайт равен трем и стартовый VCN равен 0x1BA7DA. Для преобразования VCN в LCN, нужно сложить первый LCN - 0xC0000 и VCN - 0x1BA7DA. Получаем 0xC0000 + 0x1BA7DA = 0x27A7DA. Тогда второй отрезок начинается с кластера 0x27A7DA и продолжается до 0x27A7DA+0xF30 = 0x27B70A. Для перехода к следующему отрезку нужно добавить размеры полей плюс единицу для самого байта длин. Третий отрезок начинается с байта длин - 32 и размер в кластерах отрезка равен 0x36A0, VCN равен 0x5895E. Для преобразования VCN-LCN нужно сложить предыдущий стартовый LCN с данным VCN, т. е. 0x27A7DA + 0x5895E = 0x2D3138. Получаем третий отрезок 0x2D3138 - 2D67D8. Следующий байт за отрезком нулевой, следовательно список отрезков закончен. Итак исходный атрибут размещается в кластерах 0xC0000 – 0xC3A90, 0x27A7DA - 0x27B70A, 0x2D3138 – 0x2D67D8 (не включая последний кластеры).

# 3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость программного продукта в будущем путем добавления новых блоков.

## 3.1 Блок запуска утилиты

Блок запуска утилиты предназначен для обработки входных параметров и дальнейшей корректной работы программы. Он также производит попытку доступа к разделу и получения базовой составляющей для дальнейшей работы – информации об расположении файловой системы. Данный блок пресекает работу утилиты для любых файловых систем, отличных от NTFS.

## 3.2 Блок анализа

Блок анализа является основным модулем, обеспечивающим выполнение утилитой своих функций. Он проверяет все записи файловой системы на корректность. Данный блок работает в паре с блоком журналирования.

## 3.3 Блок журналирования

Блок журналирования предназначен для сохранения в соответствующие файлы промежуточных результатов анализа для дальнейшего с ними ознакомления.

## 3.4 Блок пользовательского интерфейса

Блок пользовательского интерфейса предназначен для взаимодействия с пользователем и отображения сведений о процессе прохождения анализа.

# 4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

## 4.1 Описание используемых структур

**1. Структура BootSector**

Структура представляет собой загрузочный сектор и необходима для проверки на тип файловой системы (NTFS, FAT32, EXFAT). Так же эта структура позволяет определить размеры необходимых составляющих диска, размер записи MFT, местонахождение записи $MFT на диске.

Поля:

UCHAR jump[3] – команда перехода на загрузочный код;

ULARGE\_INTEGER oemID – тип файловой системы ;

WORD bytesPerSector – кол-во байт на сектор;

UCHAR sectorsPerCluster – кол-во секторов в кластере;

USHORT zero0– равно 0;

UCHAR zero1 – равно 0;

USHORT zero2 ­– равно 0;

USHORT zero3 – равно 0;

UCHAR mediaDescriptor ­– тип носителя;

USHORT zero4 – равно 0;

WORD sectorsPerTrack – не используется;

WORD headNumber – не используется;

DWORD hiddenSector – не используется ;

ULONG zero5 – равно 0;

UCHAR unused0[4] – не используется;

LONGLONG totalSectors – кол-во секторов в разделе;

LONGLONG MFTCluster – кластер, определяющий местоположение записи $MFT;

LONGLONG MFTMirrCluster – кластер, определяющий местоположение копии MFT;

CHAR clustersPerRecord – кол-во кластеров на запись MFT;

UCHAR unused1[3] – не используется;

CHAR clustersPerIndex – кол-во кластеров на индексную запись;

UCHAR unused2[3] – не используется;

LONGLONG serialNumber – серийной номер тома;

DWORD checkSum – не используется;

UCHAR bootCode[0x1aa] – загрузочный код;

UCHAR endMarker[2] ­– маркер конца.

**2. Структура RecordHeader**

Структура представляет собой заголовок записи в таблице MFT.

Поля:

UCHAR signature[4]; – сигнатура записи, обычно FILE;

USHORT updateSeqOffset;

USHORT updateSeqNumber;

LONGLONG lsn;

USHORT sequenceNumber;

USHORT hardLinkCount; – число жёстких ссылок

USHORT attributeOffset; – смещение первого атрибута относительно заголовка записи;

USHORT flag; – флаг, определяющий тип записи и её использование;

ULONG usedSize; – размер заголовка;

ULONG allocatedSize; – выделенный размер;

LONGLONG baseRecord; – ссылка на базовую запись;

USHORT nextAttributeID;

USHORT unused; – не используется

ULONG MFTRecord; – порядковый номер записи в таблице.

**3. Структура AttributeHeaderR**

Структура представляет собой заголовок резидентного атрибута.

Поля:

ULONG typeID – тип атрибута;

USHORT length – длина заголовка;

USHORT reserved – зарезервировано;

UCHAR formCode – 0x00 для резидентного атрибута;

UCHAR nameLength – длина имени атрибута;

USHORT nameOffset – смещение имени атрибута относительно заголовка атрибута;

USHORT flag – флаги;

USHORT attributeID – идентификатор атрибута;

ULONG contentLength – размер, в байтах, тела атрибута;

USHORT contentOffset – смещение в байтах тела атрибута относительно заголовка;

WORD unused – не используется.

**4. Структура AttributeHeaderNR**

Структура представляет собой заголовок нерезидентного атрибута.

Поля:

ULONG typeID – тип атрибута;

USHORT length – длина заголовка;

USHORT reserved – зарезервировано;

UCHAR formCode – 0x01 для нерезидентного атрибута;

UCHAR nameLength – длина имени атрибута;

USHORT nameOffset – смещение имени атрибута относительно заголовка атрибута;

USHORT flag – флаги;

USHORT attributeID – идентификатор атрибута;

LONGLONG startVCN – начальный виртуальный номер кластера списка отрезков;

LONGLONG endVCN – конечный виртуальный номер кластера списка отрезков;

USHORT runListOffset – смещение списка отрезков;

USHORT compressionUnit – размер блока сжатия

UCHAR unused[4] – не используется;

LONGLONG allocatedContentSize ­– выделенный размер тела;

LONGLONG contentSize – размер тела;

LONGLONG initializedContentSize – инициализированный размер тела.

**5. Структура FileName**

Структура представляет собой содержимое атрибута $FILE\_NAME.

Поля:

LONGLONG parentDirectory – адрес родительского каталога;

LONGLONG dateCreated – время создания файла;

LONGLONG dateModified – время модификации файла;

LONGLONG dateMFTModified – время модификации MFT;

LONGLONG dateAccessed – время обращения к файлу;

LONGLONG allocatedSize – выделенный размер файла;

LONGLONG usedSize – размер файла;

ULONG flag – флаги;

ULONG reparseValue – точка подключения;

UCHAR nameLength – длина имени;

UCHAR nameType – пространство имён;

UCHAR name[] – имя.

**6. Структура Run**

Структура представляет собой отрезок, входящий в список отрезков.

Поля:

LONGLONG offset – начальный кластер отрезка;

LONGLONG length – длина отрезка.

Методы:

Run() – конструктор по умолчанию;

Run(LONGLONG offset, LONGLONG length) – конструктор с параметрами.

**7. Перечисление MFT\_RECORD\_FLAGS**

Перечисление представляет собой флаги записей в таблице MFT, которые проверяются в ходе анализа.

Поля:

MFT\_RECORD\_NOT\_USED = 0 – запись является файлом и не используется;

MFT\_RECORD\_IN\_USE = 1 – запись является файлом и используется;

MFT\_RECORD\_IS\_DIRECTORY = 2 ­­– запись является директорией.

**8. Перечисление ATTR\_TYPES**

Перечисление представляет собой типы атрибутов, которые могут содержать записи и непосредственно используются в ходе работы программы.

Поля:

AT\_FILE\_NAME = 0x30 – атрибут типа $FILENAME;

AT\_DATA = 0x80 – атрибут типа $DATA;

AT\_BITMAP = 0xb0 – атрибут типа $BITMAP;

AT\_END = 0xffffffff – обозначение конца списка атрибутов.

## 4.2 Описание используемых функций

1. void readBootSector(HANDLE handle, BootSector &bootSector, ULONG &sectorSize, ULONG &clusterSize, ULONG &recordSize, LONGLONG &totalClusters);

Функция считывает начальные байты раздела для получения содержимого загрузочного сектора, проверяет тип файловой системы на соответствие NTFS и рассчитывает размер записи в таблице MFT.

2. void readMFTMain (HANDLE handle, std::vector<Run>& MFTRunList, LONGLONG& MFTSize, std::vector<Run>& BitMapRunList, LONGLONG& BitMapSize, ULONG sectorSize, ULONG clusterSize, ULONG recordSize);

Функция используется для работы с записью $MFT. Вызывает внутри себя функции для определения списка отрезков нерезидентного атрибута $DATA и нерезидентного атрибута $BITMAP файла $MFT.

3. void readRecord (HANDLE h, LONGLONG recordIndex, const std::vector<Run>& MFTRunList, ULONG recordSize, ULONG clusterSize, ULONG sectorSize, LPBYTE buffer);

Функция предназначена для чтения записи с индексом recordIndex путём определения смещения этой записи относительно начала таблицы MFT. Чтение производится по секторам. Результат чтения записывается в аргумент buffer типа LPBYTE, который будет указывать на начало записи.

4. LPBYTE findAttribute (RecordHeader\* record, ULONG recordSize, ULONG typeID);

Функция используется для поиска атрибута типа typeID в записи. Возвращает тип LPBYTE, который указывает на смещение начала нужного атрибута.

5. void seek(HANDLE h, ULONGLONG position);

Функция предназначена для перестановки указателя в файле с дескриптором h на позицию position.

6. void readRunList (HANDLE handle, LONGLONG recordIndex, ULONG typeID, const std::vector<Run>& MFTRunList, ULONG sectorSize, ULONG clusterSize, ULONG recordSize, std::vector<Run>\* runList, LONGLONG\* contentSize);

Функция предназначена для чтения списка отрезков нерезидентного атрибута типа typeID в записи с индексом recordIndex. Содержимое списка отрезков записывает в аргумент runList типа std::vector<Run>, а размер содержимого атрибута в аргумент contentSize типа LONGLONG.

7. std::vector<Run> parseRunList (LPBYTE runList);

Функция предназначена для преобразования закодированного списка отрезков в удобочитаемую форму типа offset – length. Возвращает список отрезков.

8. void mainAnalysis(HANDLE handle, const std::vector<Run>& MFTRunList, LONGLONG recordsNumber, const std::vector<Run>& BitMapRunList, LONGLONG BitMapSize, ULONG sectorSize, ULONG clusterSize, ULONG recordSize, const std::string& volume, std::atomic<LONGLONG>& progress, std::atomic<bool>& analysisCompleted);

Функция является базовой для процесса анализа. Содержит в себе методы для журналирования, обработки всех записей в таблице и вывод результатов для пользователя.

9. void processRecord(HANDLE handle, const std::vector<Run>& MFTRunList, LONGLONG recordIndex, ULONG recordSize, ULONG clusterSize, ULONG sectorSize, const std::vector<Run>& BitMapRunList, LONGLONG BitMapSize, std::fstream& log, std::ofstream& analysis, std::vector<UCHAR>& record, bool& errorOccur);

Функция предназначена для проверки одной записи в таблице на целостность. Целостность проверяется путем проверки флага записи и соответствия содержимого битовой карты с содержимым таблицы MFT. Параллельно с работой функции в log-файл заносятся соответствующие результаты.

10. void readBitmap(HANDLE handle, const std::vector<Run>& BitMapRunList, LONGLONG BitMapSize, ULONG clusterSize, std::vector<UCHAR>& blocks);

Функция предназначена для чтения содержимого атрибута $BITMAP таблицы MFT с целью использования в методе обработки записи. Возвращает вектор байт атрибута.

11. void writeToLog (std::fstream& log, const std::string& volume);

Функция предназначена для отделения позиции в log-файле и записи базовой информации о сканировании: диск, подлежащий проверке, дата и время.

12. void writeToAnalysis(std::ofstream& analysis, const std::string& volume);

Функция предназначена для отделения позиции в analysis-файле и записи базовой информации о сканировании: диск, подлежащий проверке, дата и время.

13. void progressBar(std::atomic<LONGLONG>& progress, LONGLONG recordsNumber, std::atomic<bool>& analysisCompleted);

Функция предназначена для отображения на экране индикатора прогресса сканирования. Позволяет пользователю примерно оценить время выполнения заданной задачи.

14. void basicInformation (BootSector bootSector, ULONG sectorSize,ULONG clusterSize, ULONG recordSize,LONGLONG totalClusters, LONGLONG MFTSize, LONGLONG recordsNumber, const std::string& volume);

Функция предназначена для вывода пользователю базовой информации о разделе раздела. Также помечает свое выполнение в log-файле.

# 5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 5.1 Разработка алгоритмов

**5.1.1 Алгоритм чтения записи void readRecord(HANDLE h, LONGLONGrecordIndex, const std::vector<Run>& MFTRunList, ULONG recordSize, ULONG clusterSize, ULONG sectorSize, LPBYTE buffer);**

Шаг 1. Начало;

Шаг 2. Определить, сколько секторов занимает запись и найти её смещение от начала таблицы в секторах;

Шаг 3. Определить номер кластера, в котором находится данный сектор;

Шаг 4. Определить, в каком отрезке MFT находится данный кластер по его vcn, если такого отрезка нет, то выбросить исключение и завершить программу;

Шаг 5. Рассчитать смещение сектора относительно начала MFT в байтах, используя смещение соответствующего отрезка MFT;

Шаг 6. Вызвать функцию seek() для перемещения указателя файла на нужное место;

Шаг 7. Прочитать данные из файла в буфер, если произошла ошибка чтения или реальное количество считанных байт не совпадает с фактическим, выбросить исключение и завершить программу;

Шаг 8. Если прочитаны не все сектора записи, перейти к шагу 3;

Шаг 9. Конец.

**5.1.2 Алгоритм чтение списка отрезков нерезидентного атрибута** **void readRunList (HANDLE handle, LONGLONG recordIndex, ULONG typeID, const std::vector<Run>& MFTRunList, ULONG sectorSize, ULONG clusterSize, ULONG recordSize, std::vector<Run>\* runList, LONGLONG\* contentSize);**

Шаг 1. Начало;

Шаг 2. Вызвать функцию readRecord() для чтения записи MFT;

Шаг 3. Вызвать функцию findAttribute() для получения заголовка атрибута, если заголовок равен nullptr, перейти к шагу 8.

Шаг 4. Проверить код атрибута, если он не равен 1, перейти к шагу 8.

Шаг 5. Вызвать функцию parseRunList() для преобразования кодированного списка отрезков в вектор типа Run;

Шаг 6. Перезаписать содержимое из временной переменной в выходной буфер;

Шаг 7. Если выходной буфер contentSize не равен nullptr, записать в него размер тела атрибута;

Шаг 8. Конец.

**5.1.3** **Алгоритм анализа записи таблицы MFT void processRecord (HANDLE handle, const std::vector<Run>& MFTRunList, LONGLONG recordIndex, ULONG recordSize, ULONG clusterSize, ULONG sectorSize, const std::vector<Run>& BitMapRunList, LONGLONG BitMapSize, std::fstream& log, std::ofstream& analysis, std::vector<UCHAR>& record, bool& errorOccur)**

Шаг 1. Начало;

Шаг 2. Прочитать запись, используя функцию readRecord();

Шаг 3. Проверить сигнатуру записи. Если она не равна “FILE”, перейти к шагу 11;

Шаг 4. Проверить наличие базовой записи. Если имеется, перейти к шагу 11;

Шаг 5. Вызвать функцию findAttribute() для поиска заголовка атрибута $FILENAME. Если он равен nullptr, перейти к шагу 11;

Шаг 6. Проверить флаг записи. В зависимости от результата сделать соответствующие пометки;

Шаг 7. Вызвать функцию readBitmap() для получения содержимого битовой карты таблицы MFT;

Шаг 8. Получить кластер, в котором располагается текущая запись;

Шаг 9. Определить какой части битовой карты и какому биту соответствует данный кластер;

Шаг 10. Проверить установлен ли бит в 1. Если нет, сделать соответствующие пометки;

Шаг 11. Конец.

**5.1.4 Алгоритм преобразования списка отрезков std::vector<Run> parseRunList (LPBYTE runList)**

Шаг 1. Начало;

Шаг 2. Сохранить указатель на список отрезков;

Шаг 3. Получить размер поля длины, анализируя младший полубайт байта длин;

Шаг 4. Получить размер поля номера, анализируя старший полубайт байта длин. Перейти к следующему байту;

Шаг 5. Выполнить побитовое сложение текущего байта с результатом значения длины;

Шаг 6. Перейти к следующему байту. Если i < размер поля длины, перейти к шагу 5;

Шаг 7. Выполнить побитовое сложение текущего байта с результатом значения смещения;

Шаг 8. Перейти к следующему байту. Если i < размер поля номера, перейти к шагу 7;

Шаг 9. Добавить текущий результат смещения к предыдущему значению;

Шаг 10. Занести значения смещения и длины отрезка в массив структур Run;

Шаг 11. Если указатель не равен 0x80, перейти к шагу 3;

Шаг 12. Конец.

# 6 РЕЗУЛЬТАТЫ РАБОТЫ

Работа программы при попытке проверки файловой системы, отличной от NFTS представлена на рис 6.1.

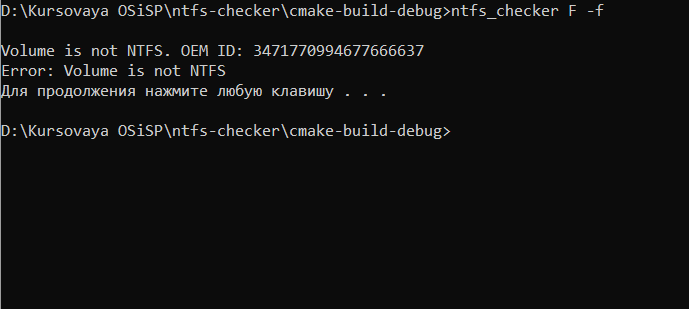


Рисунок 6.1 – Проверка файловой системы, не являющейся NFTS

Работа программы с выводом информации о файловой системе представлен на рис 6.2.

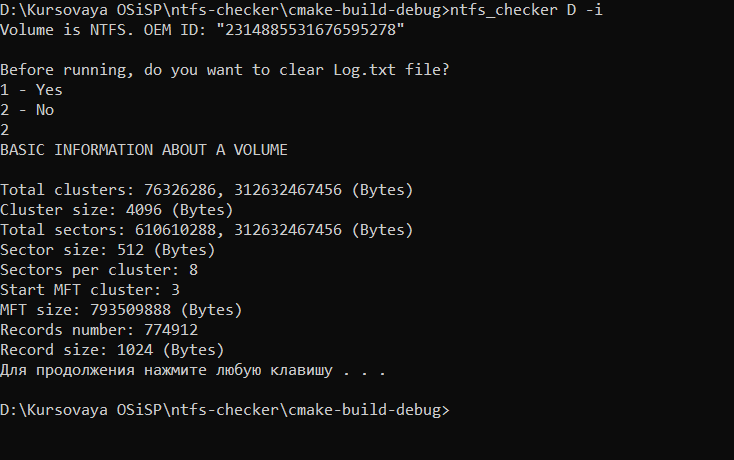


Рисунок 6.2 – Вывод информации о файловой системе

Работа программы с проверкой целостности файловой системы и обнаружением ошибок представлена рис 6.3.

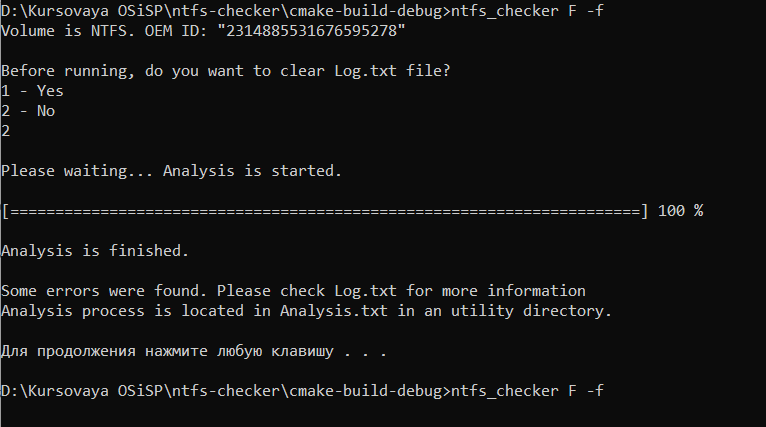


Рисунок 6.3 – Проверка целостности файловой системы с обнаружением ошибок

Работа программы при успешной проверке целостности представлена на рис 6.4.

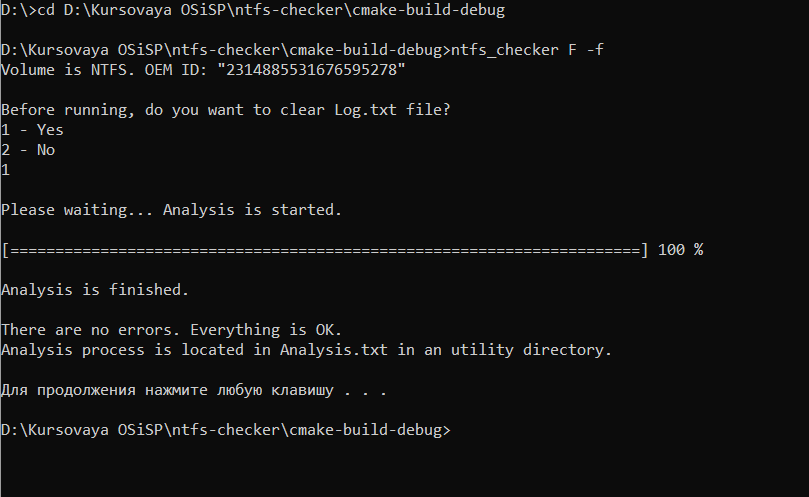


Рисунок 6.4 – Успешная проверка целостности файловой системы

При проверке целостности файловой системы процесс анализа записей таблицы MFT заносится в файл Analysis.txt в директории утилиты, а информация об повреждённых файлах и несоответствиях заносится в файл Log.txt той же директории. На рис 6.5 и рис 6.6 представлено содержимое файлов Analysis.txt и Log.txt соответственно.

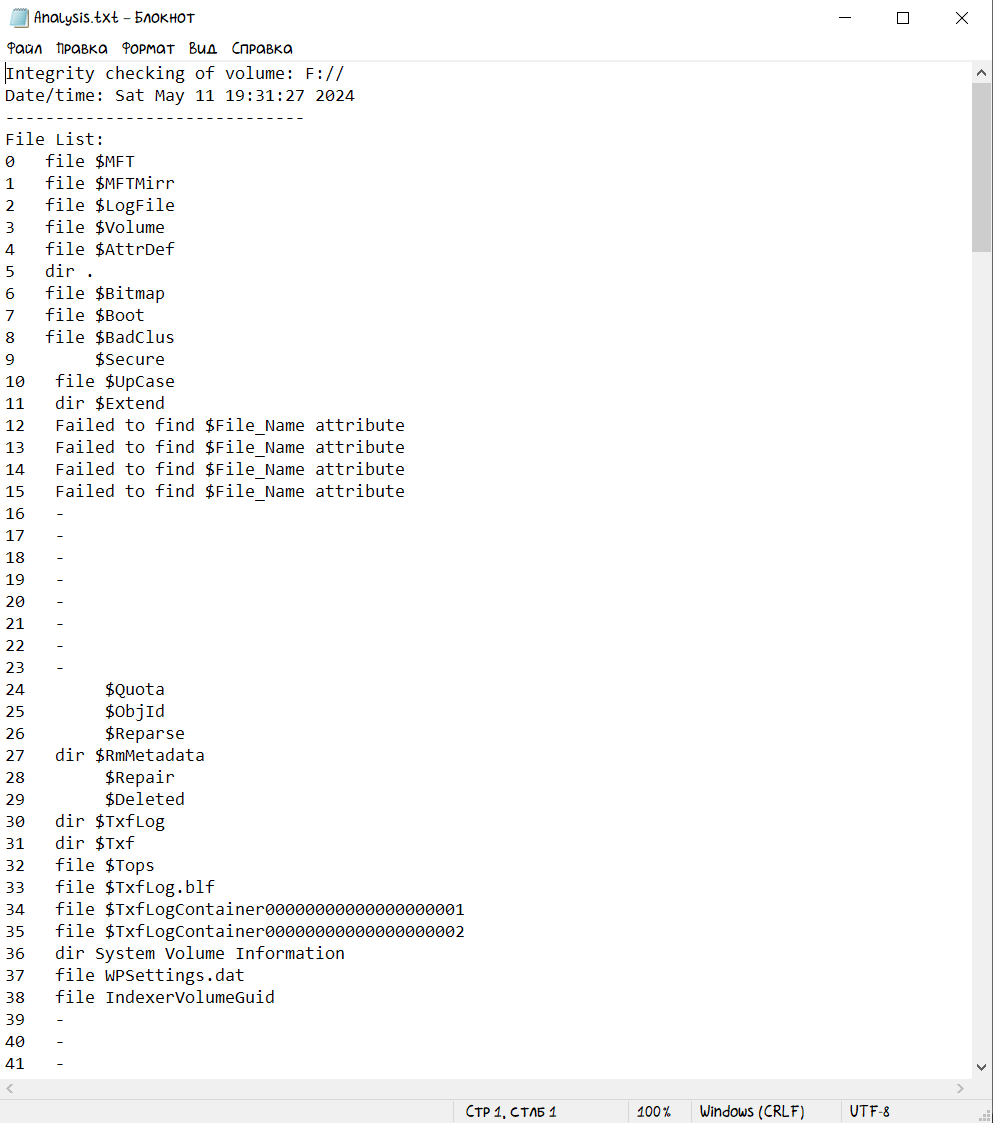


Рисунок 6.5 – Содержимое файла Analysis.txt

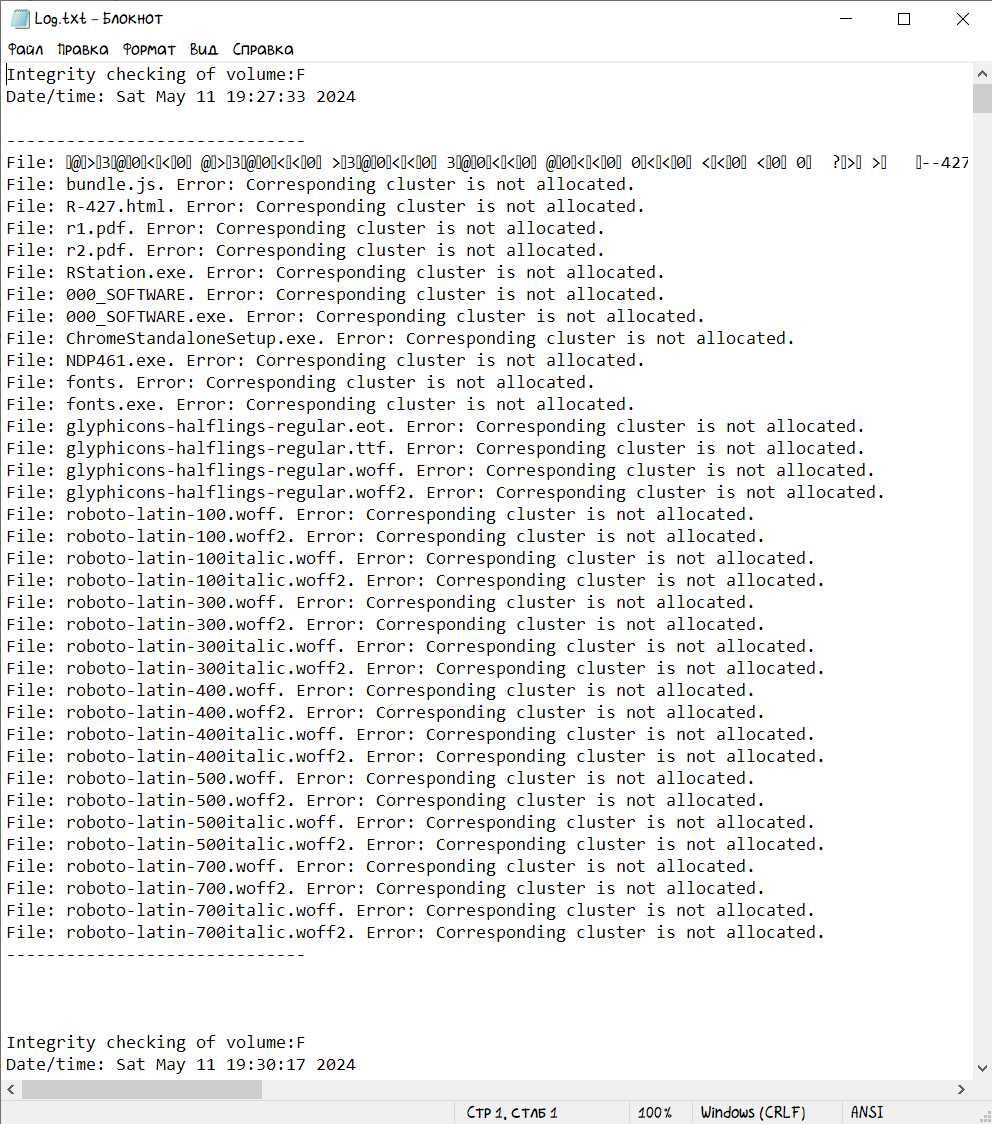


Рисунок 6.6 – Содержимое файла Log.txt

# 7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска утилиты нужно запустить в командную строку от имени администратора и перейти в директорию, где располагается исполняемый файл. Далее необходимо в консоли прописать название утилиты, раздел диска, файловую систему которого нужно проверить, и один из параметров.

Формат команды: ntfs\_checker [буква раздела] -[параметр].

Поддерживаются следующие параметры:

1. i ­­– предоставляет общую информацию о разделе;

2. f – производит сканирование файловой системы.

Пример запуска утилиты в консоли представлен на рис 7.1.

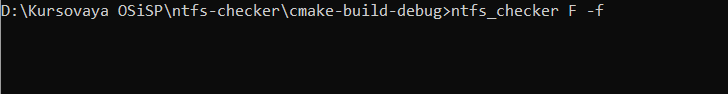


Рисунок 7.1 – Запуск утилиты через консоль

# ЗАКЛЮЧЕНИЕ

Целью данной курсовой работы было рассмотрение методов и инструментов, предназначенных для проверки целостности файловой системы NTFS. В ходе исследования были рассмотрены основные принципы функционирования файловой системы NTFS, ее структура и особенности, а также причины нарушения целостности.

В ходе данной курсовой работы была разработана утилита для проверки целостности файловой системы NTFS. Утилита создавалась по принципу используемых сегодня системных утилит проверки и восстановления, таких как Chkdsk и SFC.

В процессе разработки внимание было уделено основным этапам, на которых должен базироваться анализ и проверка. Реализация каждого этапа в программе позволило добиться успешной работы и правильного функционирования утилиты.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] citforum.ru [Электронный ресурс]. ­­– Электронные данные. – Режим доступа: http://citforum.ru/operating\_systems/windows/ntfs/2.shtml – Дата доступа 09.05.2024

[2] samag.ru [Электронный ресурс]. – Электронные данные. – Режимы доступа: https://samag.ru/archive/article/375 – Дата доступа: 09.05.2024

[3] samag.ru [Электронный ресурс]. – Электронные данные. – Режимы доступа: https://samag.ru/archive/article/395 – Дата доступа: 09.05.2024

[4] Кэрриэ, Б. Криминалистический анализ файловых систем / Б. Кэрриэ. – СПб, 2007. – 480с.

[5] Таненбаум, Э. Современные операционные системы. 4-е издание / Э. Таненбаум, Х. Бос. – СПб, 2015. – 1120с.

# ПРИЛОЖЕНИЕ А

(обязательное)

Схема структурная

# ПРИЛОЖЕНИЕ Б

(обязательное)

Диаграмма последовательности

# ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода

# ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов