Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных
машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому
проектуна тему

“Проверка целостности файловой системы NTFS”

по дисциплине

«Системное программное обеспечение вычислительных машин»

БГУИР КП 1–400201.315 ПЗ

Выполнил:
Студент гр. 050503
Липский Г.В.

Руководитель:
Глоба А.А

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день во многих компьютерах по умолчанию используется файловая система NTFS (new technology file system — «файловая система новой технологии»).

Чтобы компьютер работал быстро и исправно, время от времени его необходимо проверять и оптимизировать его работу. Одной из проверок должна быть проверка целостности файловой системы.

Данная работа посвящена разработке утилиты проверки целостности файловой системы NTFS.

Можно выделить следующие задачи проекта:

1. Ознакомиться с устройством и принципами работы NTFS системы;
2. Разработать утилиту проверки целостности файловой системы NTFS;
3. Увеличить объём знаний о языках C / C++, освоить новые алгоритмы языка;
4. Написать пояснительную записку.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор файловой системы NTFS

Для начала нужно понять что из себя представляет файловая система NTFS.

1.1.1 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 (в общем смысле этого слова). 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.

1.1.2 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 который система будет записывать информацию на накопителе. Весь диск состоит из большого массива 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.

1.1.3 Структура файловой системы NTFS

Диск NTFS условно делится на две части. Первые 12% 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 88% диска представляют собой обычное пространство для хранения файлов.

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.

1.2 Метафайлы и их атрибуты

1.2.1 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 (как и любой другой файл). 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 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. Запись 3 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 MFT (в записи 4). 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 в MFT. Следующая запись MFT указывает на файл начального загрузчика. Запись 8 используется для того, чтобы связать вместе все плохие блоки (чтобы обеспечить невозможность их использования для файлов). Запись 9 содержит информацию безопасности. Запись 10 используется для установления соответствия регистра. И наконец, запись 11 — 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.

1.2.2 Атрибуты записей MFT

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

1. Standart information - биты флагов, временные метки и т.д.;
2. File name - имя файла в Unicode;
3. Security descriptor - устарел. Информация безопасности находится в $Extend$Secure;
4. Attribute list - местоположение дополнительных записей MFT;
5. Object ID - уникальный для данного тома 64-битный идентификатор файла;
6. Reparse point - используется для монтирования и символических ссылок;
7. Volume name - название данного тома;
8. Volume information - версия тома;
9. Index root - используется для каталогов;
10. Index allocation - используется для очень больших каталогов;
11. Bitmap - используется для очень больших каталогов;
12. Logged utility stream - управляет журналированием в $LogFile;
13. Data - данные потока.

Список атрибутов нужен в том случае, когда атрибуты не помещаются в запись MFT. Из этого атрибута можно узнать, где искать записи расширения. Каждый элемент списка содержит 48-битный индекс по MFT (который говорит о том, где находится запись расширения) и 16-битный порядковый номер (для проверки того, что запись расширения соответствует базовой записи).

1.2.3 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 самой 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. Как уже упоминалось, такие 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.

Нерезидентные атрибуты. Для таких 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).

1.3 WinAPI функции для работы с файлами

В WinAPI для работы с файлами используются следующие функции:
1. CreateFile(szName, dwAccess, dwShareMode, lpSecurityAttributes, dwCreationDisposition, dwFlags, hTemplateFile);

В случае успешного создания или открытия файла, процедура CreateFile возвращает его дескриптор. В случае ошибки возвращается специальное значение INVALID_HANDLE_VALUE. Функция принимает 7 аргументов: szName – указатель на символьную строку с нулем в конце, устанавливающую имя объекта, который создается или открывается, dwAccess – тип доступа к объекту (чтение, запись или то и другое), dwShareMode – совместный доступ, lpSecurityAttributes – указатель на структуру SECURITY_ATTRIBUTES, которая устанавливает может ли возвращенный дескриптор быть унаследован дочерними процессами, dwCreationDisposition - Выполняемые действия с файлами, которые существуют и выполняемые действия с файлами, которые не существуют, dwFlags - Атрибуты и флаги файла, hTemplateFile - Дескриптор файла шаблона с правом доступа GENERIC_READ.

2. ReadFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);
Чтение из файла в буфер lpBuff размером dwBuffSize. В переменную dwCount записывается реальное количество прочитанных байт. lpBuff – размер буфера для чтения. Последний аргумент – это указатель на структуру OVERLAPPED. Эта структура требуется тогда, если параметр hFile создавался с флажком FILE_FLAG_OVERLAPPED.

3. WriteFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);
Аргументы и семантика процедуры WriteFile полностью аналогичны ReadFile.

4. CloseHandle(hFile);
Файловые дескрипторы закрываются с помощью CloseHandle. hFile – дескриптор файла.

2 ОБЗОР АНАЛОГОВ

2.1 Анализ аналогов программного средства

2.1.2 Утилита CHKDSK

Одним из главных источников будет являться утилита CHKDSK. CHKDSK (сокращение от англ. check disk — проверка диска) — стандартное приложение в операционных системах DOS и Microsoft Windows, которое проверяет жёсткий диск или дискету на ошибки файловой системы. CHKDSK также может исправлять найденные ошибки файловой системы.

Работа программы CHKDSK делится на три основных прохода, в течение которых CHKDSK проверяет все метаданные на томе, и дополнительный четвертый проход. Термин «метаданные» означает «данные о данных.» Метаданные являются надстройкой над файловой системой, в которой отслеживаются сведения обо всех файлах, хранящихся на томе. В метаданных содержатся сведения о кластерах, составляющих объем данных конкретного файла, о том, какие кластеры свободны, о кластерах, содержащих поврежденные сектора и т.д. С другой стороны, данные, содержащиеся в файле, обозначаются как «данные пользователя». В NTFS метаданные защищаются с помощью журнала транзакций. Процесс изменения метаданных делится на определенные логические этапы, или транзакции, которые фиксируются в журнале. Если последовательность действий по изменению метаданных логически не завершена, то выполняется откат по данным журнала транзакций на тот момент, когда это изменение еще не было начато. Другими словами, использование журнала транзакций, значительно повышает вероятность целостности метаданных.

Можно выделить следующие преимущества утилиты CHKDSK: это утилита, которой может воспользоваться любой пользователь, даже не обладая большими знаниями в области компьютерных наук. В функционал утилиты входит сканирование жёсткого диска, нахождение и устранение ошибок, что гарантирует оптимальное функционирование диска. Одним из преимуществ является быстрое выполнение всех процедур, заданных пользователем. В большинстве случаев утилита отлично справляется с проверкой и восстановлением жесткого диска. Также преимуществом можно считать поддержку русского языка, т.к. многие рядовые пользователи компьютеров не понимают английский. Нельзя не причислить к преимуществам ещё то, что утилиту CHKDSK не нужно устанавливать, она встроена в Windows по умолчанию.

У CHKDSK есть свои недостатки: в первую очередь это отсутствие удобного пользовательского интерфейса. Недостатком можно считать минимальную информативность и отсутствие дополнительных функций для полного анализа состояния диска.

2.1.2 Утилита **DiskEditor**

Утилитой для проверки получаемых значений в ходе выполнения работы будет приложение DiskEditor. С помощью DiskEditor можно получить доступ к метафайлам NTFS, посмотреть содержимое этих файлов, узнать размер, расположение на диске и другую полезную информацию. Выбор пал именно на DiskEditor, т.к. недостатков для себя я не обнаружил. А преимущества по отношению к аналогам имеются. Некоторые из них я описал выше. К преимуществам также можно отнести понятный и удобный интерфейс, наличие встроенного редактора бинарных файлов.

2.2 Постановка задачи

Ознакомившись с принципами работы аналогов можно поставить следующую задачу:

Программа должна выполнять следующие функции:

- Сканирование файловой системы;
- Определение размера и структуры жесткого диска;
- Определение наличия, размера и расположения метаданных на диске;
- Вывод информации на экран.

3 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

3.1 Программные компоненты

Для разработки приложения взяты стандартные библиотеки языка C и C++.

Список библиотек представлен ниже:

`Windows.h` – для работы с файловыми системами устройств и мультимедиа.

`Winioctl.h` – включает в себя программные интерфейсы IOSTL для работы с NTFS (см. https://docs.microsoft.com/en-us/windows/win32/api/winioctl/).

`stdio.h` – стандартный заголовочный файл ввода-вывода

`tchar.h` – определяет тип данных TCHAR, необходимый для работы с WinAPI функциями.

`shellapi.h` – заголовочный файл, включающий в себя функцию CommandLineToArgvW для работы с командной строкой.

`vector` – Стандартный шаблон обобщённого программирования языка C++ std::vector<T> – реализация динамического массива.

`functional` – заголовочный файл, предоставляющий набор шаблонов классов для работы с функциональными объектами, а также набор вспомогательных классов для их использования в алгоритмах стандартной библиотеки.

3.2 Структура приложения

Программа представляет собой консольное приложение, в котором можно выделить несколько блоков.

Главным блоком является блок анализа. Он разделён на два блока: блок анализа MFT и блок считывания структуры диска.

В основе блока анализа MFT лежит BootSector, сканируя который мы получаем информацию о структуре и размере MFT. Помимо этого в блок анализа MFT можно включить функцию DeviceIoControl, которая заполняет NTFS_VOLUME_DATA_BUFFER, если её вызвать с кодом FSCTL_GET_NTFS_VOLUME_DATA.

Блок считывания структуры диска включает в себя следующие функции: GetDriveGeometry, которая собирает информацию о диске и printgeometry, которая эту информацию выводит на экран.

Ещё один блок в программе – это блок обработки записей из MFT. Чтение записей обеспечивает функции readRecord и findAttribute. Копирование и вывод на экран осуществляется за счёт функций findAttribute, seek, ReadFile и WriteFile.

3.3 Описание использованных структур

3.3.1 Структура BootSector

Структура необходима для проверки на тип файловой системы (NTFS, FAT32, EXFAT). Так же с помощью этой структуры можно определить размер кластера, размер записи MFT, местонахождение записи $MFT на диске.

Структура включает в себя большое количество полей без которых невозможно сканирование Boot-сектора. Однако непосредственно в коде программы используются далеко не все поля. Поле oemID сохраняет тип файловой системы. С помощью этого поля происходит проверка является ли файловая система именно NTFS-системой. Из названий полей bytePerSector, sectorPerTrack, sectorPerCluster, clusterPerRecord и clusterPerBlock понятно, что после сканирования Boot-сектора в них сохранится значение количества чего-то в чём-либо (например количество байт в секторе). Поле totalSector хранит информацию о количестве секторов в разделе диска. Поля MFTCluster и MFTMirrCluster содержат информацию о местонахождении $MFT и $MFTMirr.

3.3.2 Структура RecordHeader

Структура нужна для определения типа записи MFT (файл ли это или директория). Поле flag содержит в себе эту информацию. Поле baseRecord содержит в себе номер базовой записи.

Структура используется в функции findAttribute.

3.3.3 Структура AttributeHeaderR

Структура предназначена для работы с резидентными данными.

Поле typeID используется в функции findAttribute для сравнения со значением, которое было передано в функцию параметром (0x20 - $ATTRIBUTE_LIST, 0x30 - $FileName, 0x80 - $Data). Поле formCode может принимать два значения: 0x00 – резидентная форма (значение находится в записи файла) и 0x01 – нерезидентная форма (значение содержится в других секторах на диске). Поле length содержит размер записи атрибута.

3.3.4 Структура AttributeHeaderNR

Аналогична структуре AttributeHeaderNR, только для работы с нерезидентными данными.

3.3.5 Структура Run

Структура используется для копирования записи MFT в бинарный файл.

В структуре содержится два поля: offset и length. Offset – хранит смещение в файле, length – длину записанных данных.

3.3.6 Структура AttributeRecord

Структура необходима для хранения свойств атрибутов записи MFT.

В структуре используются поля typeID - идентификатор атрибута, recordLength и recordNumber – длина и номер записи MFT.

4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

4.1 Описание функционирования программы.

Программа запускается из командной строки с двумя или четырьмя аргументами. Два первых агрумента обязательные: первый аргумент – название исполняемого файла, второй – буква раздела диска (например C, D, E и пр.), в котором будут производиться определённые в программе операции. Ещё два аргумента нужны для запуска отдельного блока программы: третий аргумент – номер записи MFT, четвёртый аргумент – название файла, в который будет скопировано содержимое записи. Аргументы командной строки запоминаются с помощью функции CommandLineToArgvW.

При запуске программы пользователь может ознакомиться со структурой жесткого диска. На экран выводятся следующие значения: название диска, количество цилиндров, дорожек, секторов и их размер, размер диска. С помощью функции CreateFile, был извлечён дескриптор устройства физического диска, после этого вызывая функцию DeviceIoControl с управляющим кодом IOCTL_DISK_GET_DRIVE_GEOMETRY, чтобы заполнить структуру DISK_GEOMETRY информацией о диске, получаются результаты, которые выводятся на экран.

Далее происходит подсчёт количества кластеров, количество свободных кластеров на диске с помощью той же функции DeviceIoControl, но с управляющим кодом FSCTL_GET_NTFS_VOLUME_DATA.

С помощью функции ReadFile заполняется объект структуры BootSector, в котором содержится информация о количестве кластеров на диске, количестве свободных кластеров, длине записи в MFT.

Вызвав функцию readList получаем количество записей в MFT. Далее записи считываются в цикле с помощью функции readRecord.

В случае, если пользователь введёт четыре аргумента (копирование записи в файл), выполняется функция WriteFile если атрибут резидентный (умещается в записи MFT). Если атрибут нерезидентный, то в первую очередь выполняется функция seek, принимающая в качестве аргументов дескриптор файла и позицию для смещения.

Более подробно алгоритмы работы функций описаны в разделе “Разработка программных модулей”.

5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

5.1 Принцип работы функции `BOOL GetDriveGeometry (LPWSTR wszPath, DISK_GEOMETRY* pdg).`

Функция предназначена для получения структуры жёсткого диска.

Результат выполнения функции CreateFileW с первым аргументом wszPath заносим в HANDLE hDevice, который, в свою очередь, является первым аргументов функции DeviceIoControl, которую вызываем после CreateFileW. Результат выполнения функции заносим в BOOL bResult, которую и возвращает функция GetDriveGeometry.

5.2. Принцип работы функции `void printgeometry().`

Эта функция нужна для вывода на экран результатов, полученных с помощью функции GetDriveGeometry.

5.3. Принцип работы функции `LPBYTE findAttribute(RecordHeader* record, DWORD recordSize, DWORD typeID).`

Функция нужна для получения необходимого атрибута записи record, переданного параметром typeID. Используется в функции readList, описанной в пункте 5.5.

5.4. Принцип работы функции `CommandLineToArgvW(GetCommandLine(), &argc).`

Функция заголовочного файла shellapi.h. Анализируют введённые данные в командную строку. Если функция завершается успешно, возвращаемое значение – ненулевой указатель на созданный список параметров, который является массивом строк.

Если функция завершается с ошибкой, возвращаемое значение - NULL.

5.5 Принцип работы функции `int readList(HANDLE h, LONGLONG recordIndex, DWORD typeID, const vector<Run>& MFTRunList, DWORD recordSize, DWORD clusterSize, DWORD sectorSize, LONGLONG totalCluster, vector<Run>* runList, LONGLONG* contentSize = NULL).`

Данная функция необходима для определения резидентности атрибута записи.

Функция принимает дескриптор файла h, номер записи в MFT recordIndex, typeID – идентификатор атрибута MFT-записи: 0x80 - $Data, 0x20 - $AttributeList, 0x30 - $FileName , MFTRunList – массив структур Run, аргументы recordSize, clusterSize, sectorSize – размер записи, кластера и сектора.

Внутри себя функция readList вызывает функцию readRecord (п. 5.10).

На выходе получается значение int stage, по значению которого определяется резидентность атрибута.

5.6. Принцип работы функции `CreateFile(szName, dwAccess, dwShareMode, lpSecurityAttributes, dwCreationDisposition, dwFlags, hTemplateFile).`

WinAPI функция, лежит в заголовочном файле Windows.h. Принимает семь аргументов. Аргумент szName задает имя файла, а dwAccess — желаемый доступ к файлу, обычно это GENERIC_READ, GENERIC_WRITE. Параметр dwShareMode определяет, что могут делать с файлом другие процессы, пока мы с ним работаем. Возможные значения — FILE_SHARE_READ, FILE_SHARE_WRITE, FILE_SHARE_DELETE и их комбинации. Параметр dwCreationDisposition определяет, как именно мы хотим открыть файл, может быть, например, CREATE_NEW, CREATE_ALWAYS, OPEN_EXISTING, OPEN_ALWAYS.

5.7. Принцип работы функции `void seek(HANDLE h, ULONGLONG position).`

Функция предназначена для установки позиции в файле h на позицию position. Внутри себя вызывает функцию SetFilePointerEx, которая перемещает указатель позиции в файле открытого файла.

5.8. Принцип работы функции `BOOL ReadFile(hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORlpNumberOfBytesRead, LPOVERLAPPED lpOverlapped).`

WinAPI функция ReadFile читает данные из файла, начиная с позиции, обозначенной указателем файла. hFile - дескриптор файла, который читается. lpBuffer - указатель на буфер, который принимает прочитанные данные из файла. nNumberOfBytesToRead - число байтов, которые читаются из файла. pNumberOfBytesRead - указатель на переменную, которая получает число прочитанных байтов. lpOverlapped - указатель на структуру OVERLAPPED. Эта структура требуется тогда, если параметр hFile создавался с флажком FILE_FLAG_OVERLAPPED.

5.9. Принцип работы функции `readRecord(HANDLE h, LONGLONG recordIndex, const vector<Run>& MFTRunList, DWORD recordSize, DWORD clusterSize, DWORD sectorSize, BYTE* buffer).`

Из названия функции понятно, что она нужна для чтения записей из MFT-зоны. Функция считает смещение offset, которое передаётся в функцию seek, чтобы считать нужную информацию из MFT-зоны. После того как нужная позиция была установлена, вызывается функция ReadFile. Если произошла ошибка чтения, вызывается исключение. Если файл прочтён, то происходит вызов функции fixRecord.

5.10. Принцип работы функции `BOOL DeviceIoControl(HANDLE hDevice, DWORD dwIoControlCode, LPVOID lpInBuffer, DWORD nInBufferSize, LPVOID lpOutBuffer, DWORD nOutBufferSize, LPDWORD lpBytesReturned, LPOVERLAPPED lpOverlapped).`

Функция DeviceIoControl отправляет управляющий код непосредственно указанному драйверу устройства, заставляя соответствующее устройство выполнить соответствующую операцию. dwIoControlCode - управляющий код для операции. Это значение идентифицирует конкретную операцию для выполнения и тип устройства, на котором она должна осуществиться. lpOutBuffer - указатель на буфер вывода данных, который должен получить данные, возвращенные операцией.

6 ТЕСТИРОВАНИЕ

Тестирование является одним из важнейших этапов жизненного цикла, направленным на повышение качественных характеристик. Качество программного средства очень сильно зависит от того, насколько хорошо он выполняет задачи, для которых был создан. Скрытые ошибки могут сильно изменить результат работы программного продукта и тем самым привести к ошибочному выполнению поставленных задач.

Тестирование программного обеспечения - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом, это процесс многократного выполнения программы с целью обнаружения ошибок.

Тестирование программ является одной из составных частей общего понятия - "отладка программ". Если тестирование - это процесс, направленный на выявление ошибок, то целью отладки являются локализация и исправление выявленных в процессе тестирования ошибок.

Если в программном обеспечении есть ошибки или дефекты, они могут быть обнаружены на раннем этапе производства ПО и сразу же устранены. Правильно протестированный программный продукт обеспечивает надежность, безопасность и высокую производительность, что в дальнейшем приводит к экономии времени, денег и удовлетворенности клиентов.

Любая, даже самая незначительная ошибка ПО, может привести к неприятным последствиям.

На этапе тестирование проверяется правильность выполнения основных действий, совершаемых в ходе работы приложения:

Запуск утилиты для проверки файловой системы, отличной от NTFS. В данном случае exFAT (рис. 6.1)

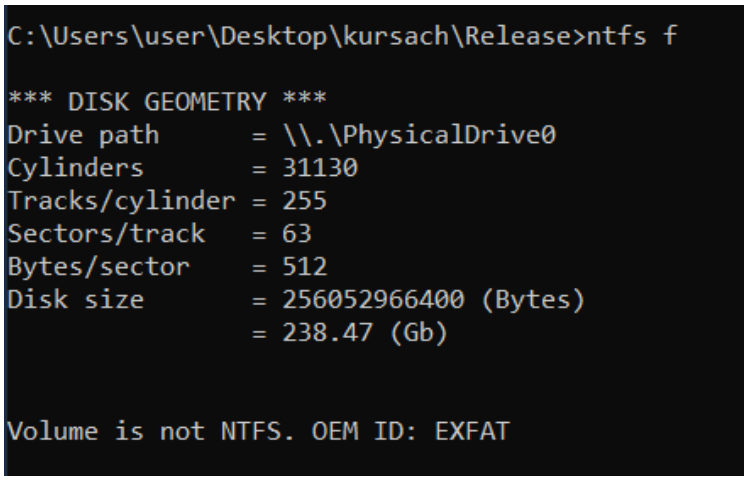


Рисунок 6.1

Запуск утилиты с двумя аргументами (рис. 6.2).

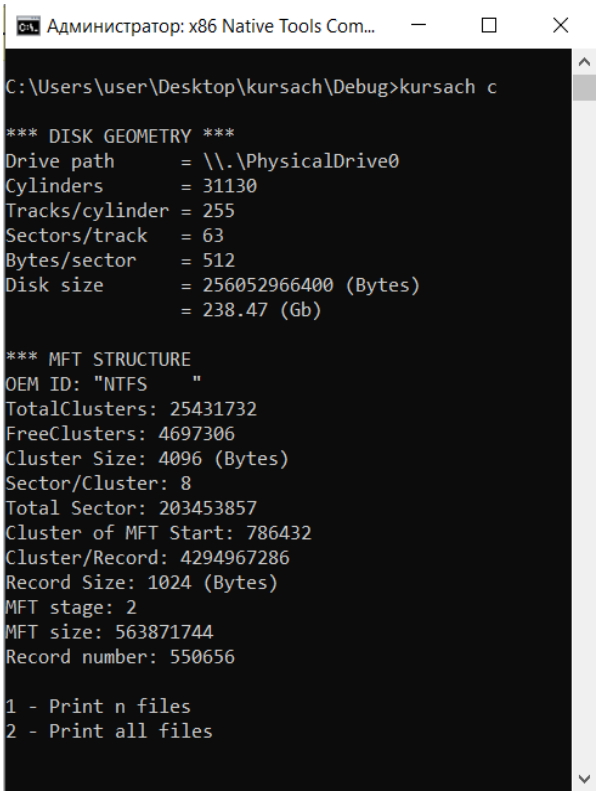


Рисунок *6.2*

Просмотр записей MFT (рис. 6.3).

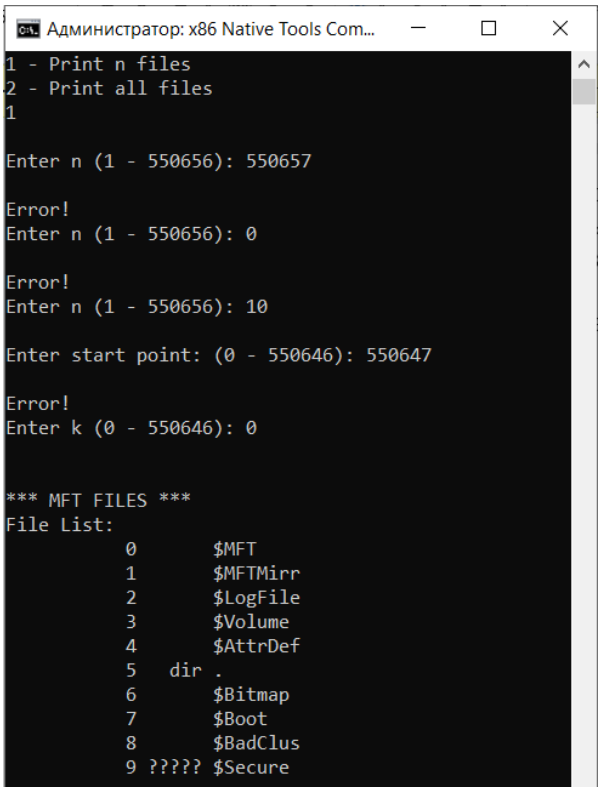


Рисунок *6.3*

Запуск утилиты с четырьмя аргументами (рис. 6.4).

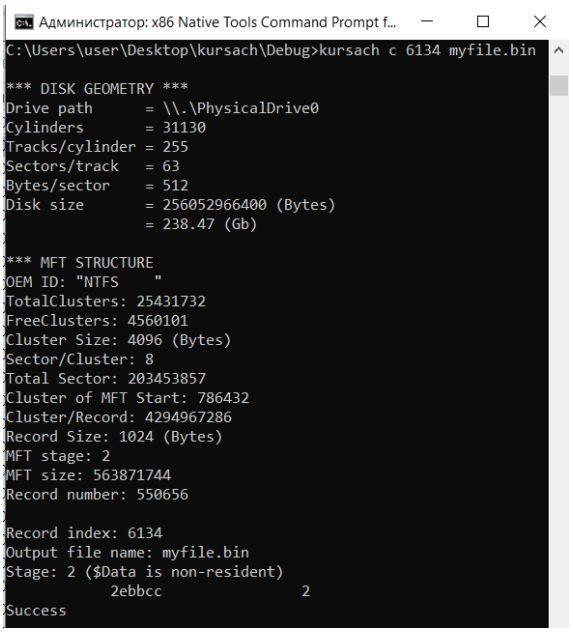


Рисунок 5.6

Копирование резидентной записи MFT в файл (рис. 6.5).

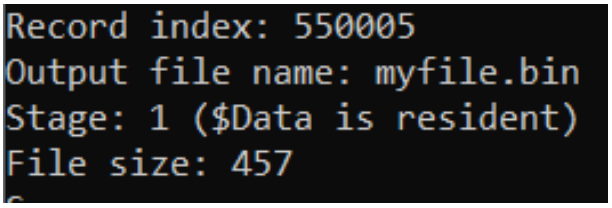


Рисунок 6.5.

Сравнение файлов (рис. 6.6).

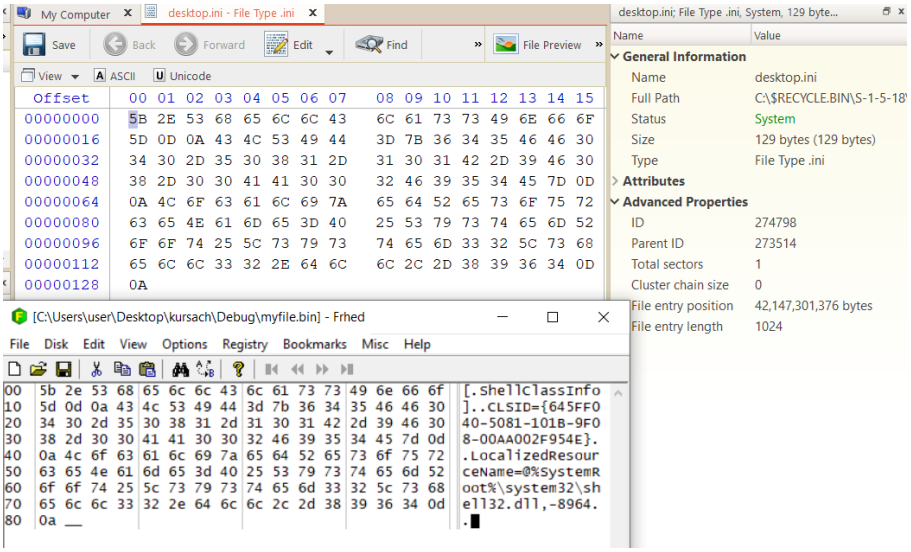


Рисунок 6.6

Копирование нерезидентной записи MFT в файл (рис. 6.7).

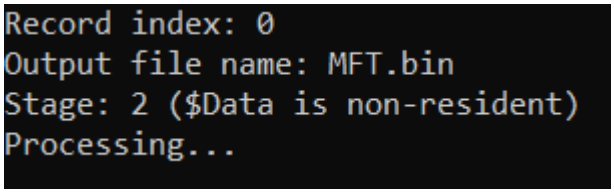


Рисунок *6.7*

Сравнение файлов (рис. 6.8).

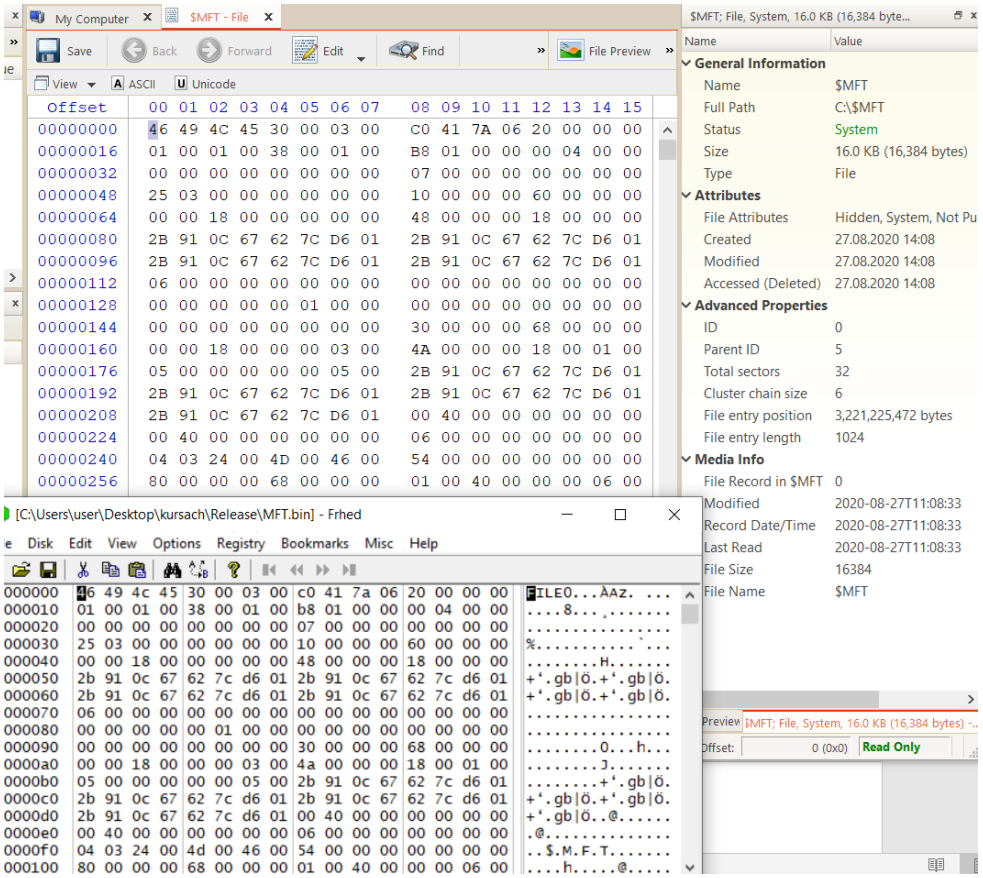


Рисунок *6.8*

В ходе тестирования программы был проведен анализ качества разработанного программного продукта, на основании которого можно утверждать, что данная система прошла контроль системы показателей качества и может быть использована по назначению.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска утилиты необходимо открыть командную строку (cmd) от имени администратора. После этого с помощью команды cd перейти в папку, где находится исполняемый exe-файл (рис. 7.1).

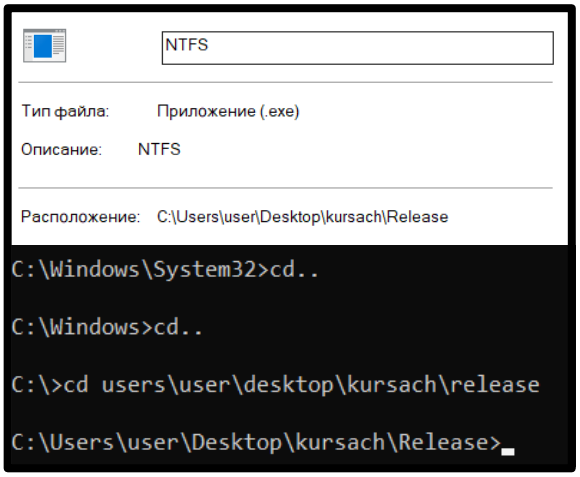


Рисунок *7.1*

После этого необходимо запустить исполняемый файл. Для этого есть два варианта запуска: для просмотра структуры диска, MFT зоны нужно ввести команду “[имя исполняемого файла] [раздел диска]” (например “NTFS C”). На экране должна появиться информация, изображенная на рисунке 6.1 раздела “Тестирование”.

Далее утилита попросит пользователя ввести число: 1 – вывести определённое число записей, 2 – вывести все записи (может занять много времени). В случае выбора пользователем первого варианта, утилита предложит выбрать сколько записей нужно вывести и с какой записи начинать отсчёт (например пользователь введёт числа 25 и 30. В таком случае будут выведены записи 30-54. См. рис 7.2.).

Для запуска утилиты с четырьмя аргументами используется команда “[имя исполняемого файла] [раздел диска] [номер записи для копирования] [имя файла, куда копировать]” (например “NTFS C 199999 myfile.bin”). Файл, который был передан последним аргументом появится в той же папке, где и находится исполняемый файл.

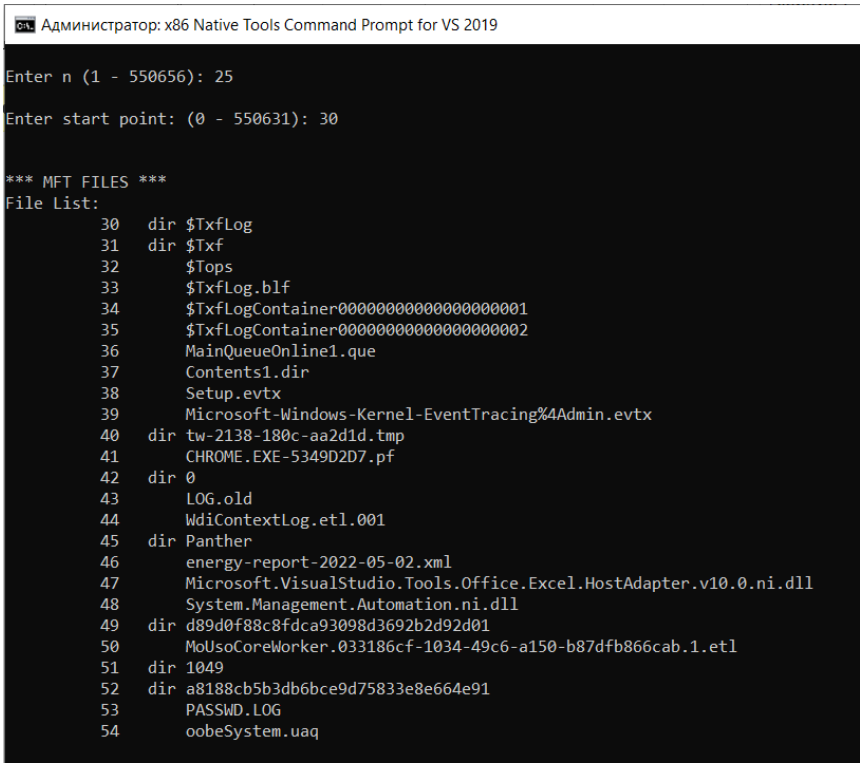

Администратор: x86 Native Tools Command Prompt for VS 2019
Enter n (1 - 550656): 25
Enter start point: (0 - 550631): 30
*** MFT FILES ***
File List:
30 dir $TxfLog
31 dir $Txf
32 $Tops
33 $TxfLog.blf
34 $TxfLogContainer00000000000000000001
35 $TxfLogContainer00000000000000000002
36 MainQueueOnline1.que
37 Contents1.dir
38 Setup.evtx
39 Microsoft-Windows-Kernel-EventTracing%4Admin.evtx
40 dir tw-2138-180c-aa2d1d.tmp
41 CHROME.EXE-5349D2D7.pf
42 dir 0
43 LOG.old
44 WdiContextLog.etl.001
45 dir Panther
46 energy-report-2022-05-02.xml
47 Microsoft.VisualStudio.Tools.Office.Excel.HostAdapter.v10.0.ni.dll
48 System.Management.Automation.ni.dll
49 dir d89d0f88c8fdca93098d3692b2d92d01
50 MoUsoCoreWorker.033186cf-1034-49c6-a150-b87dfb866cab.1.etl
51 dir 1049
52 dir a8188cb5b3db6bce9d75833e8e664e91
53 PASSWD.LOG
54 oobeSystem.uaq

Рисунок *7.2*

ЗАКЛЮЧЕНИЕ

В данной работе был проведён анализ поставленной проблемы с последовательным её решением.

Целью была разработка утилиты проверки целостности файловой системы NTFS. В результате проведения работы она была выполнена.

Цель была достигнута путём успешного выполнения основных задач курсового проекта.

Выявленные в ходе тестирования ошибки были устранены.

С помощью разработанной утилиты можно получить доступ к системным файлам, анализировать структуру диска, анализировать состояние MFT-зоны. Просматривать содержимое записей MFT-зоны.

Данный проект в дальнейшем может быть усовершенствован в следующих направлениях:

1. Расширение функционала, путём добавления новых функций.
2. Создание графического интерфейса для упрощения работы с утилитой для неопытных пользователей компьютера.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] ntfs.com - информационный сервис. - [Электронный ресурс]. - Режим доступа: https://www.ntfs.com/hard-disk-basics.htm.

[2] ixbt.com - информационный сервис. - [Электронный ресурс]. - Режим доступа: https://www.ixbt.com/storage/ntfs.html.

[3] fighters.ru - информационный сервис. - [Электронный ресурс]. - Режим доступа: https://fighters.ru/kakoi-razmer-klastera-vybrat-pri-formatirovanii-ne-mogu-opredelitsya/.

[4] docs.microsoft.com - техническая документация Майкрософт. - [Электронный ресурс]. - Режимы доступа:
https://docs.microsoft.com/en-us/windows/win32/devio/calling-deviceiocontrol.
https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ni-winioctl-fsctl_get_ntfs_volume_data.
https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-ntfs_volume_data_buffer.

[5] Э. Таненбаум - Современные операционные системы 4-е издание. 2015.

[6] citforum.ru - информационный сервис. – [Электронный ресунс]. – Режим доступа: http://citforum.ru/operating_systems/windows/ntfs/2.shtml

[7] eax.me - информационный сервис. - [Электронный ресурс]. – Режим доступа: https://eax.me/winapi-files/

[8] habr.com - информационный сервис. - [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/164193/

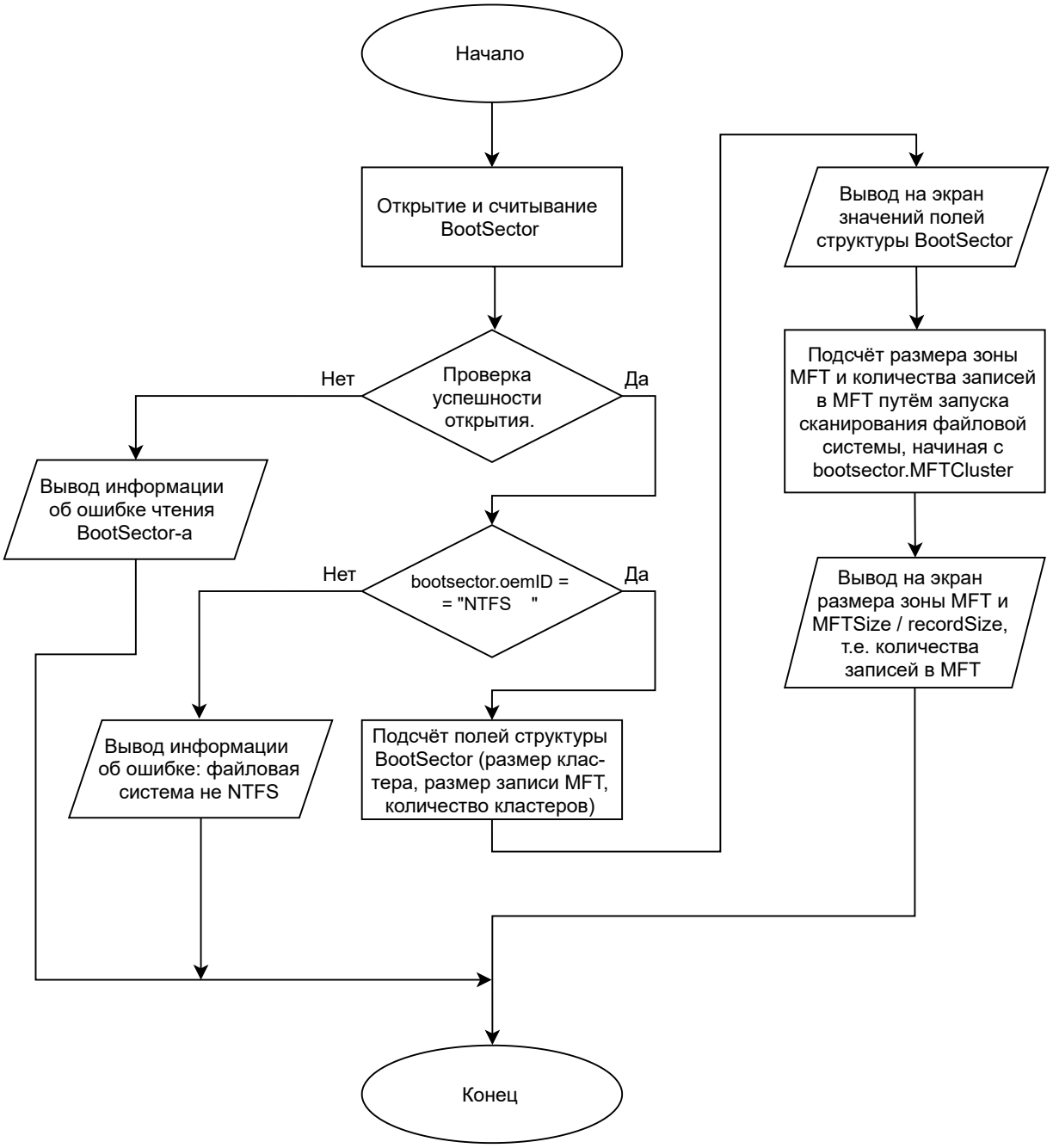
[9] writeblocked.org - информационный сервис. - [Электронный ресурс]. – Режим доступа: https://www.writeblocked.org/resources/NTFS_CHEAT_SHEETS.pdf

[10] Gary Nebbett - Windows NT/2000 Native API Reference, 1e издание (с.458-481).

ПРИЛОЖЕНИЕ А

(обязательное)

Блок-схема алгоритма чтения Boot-сектора



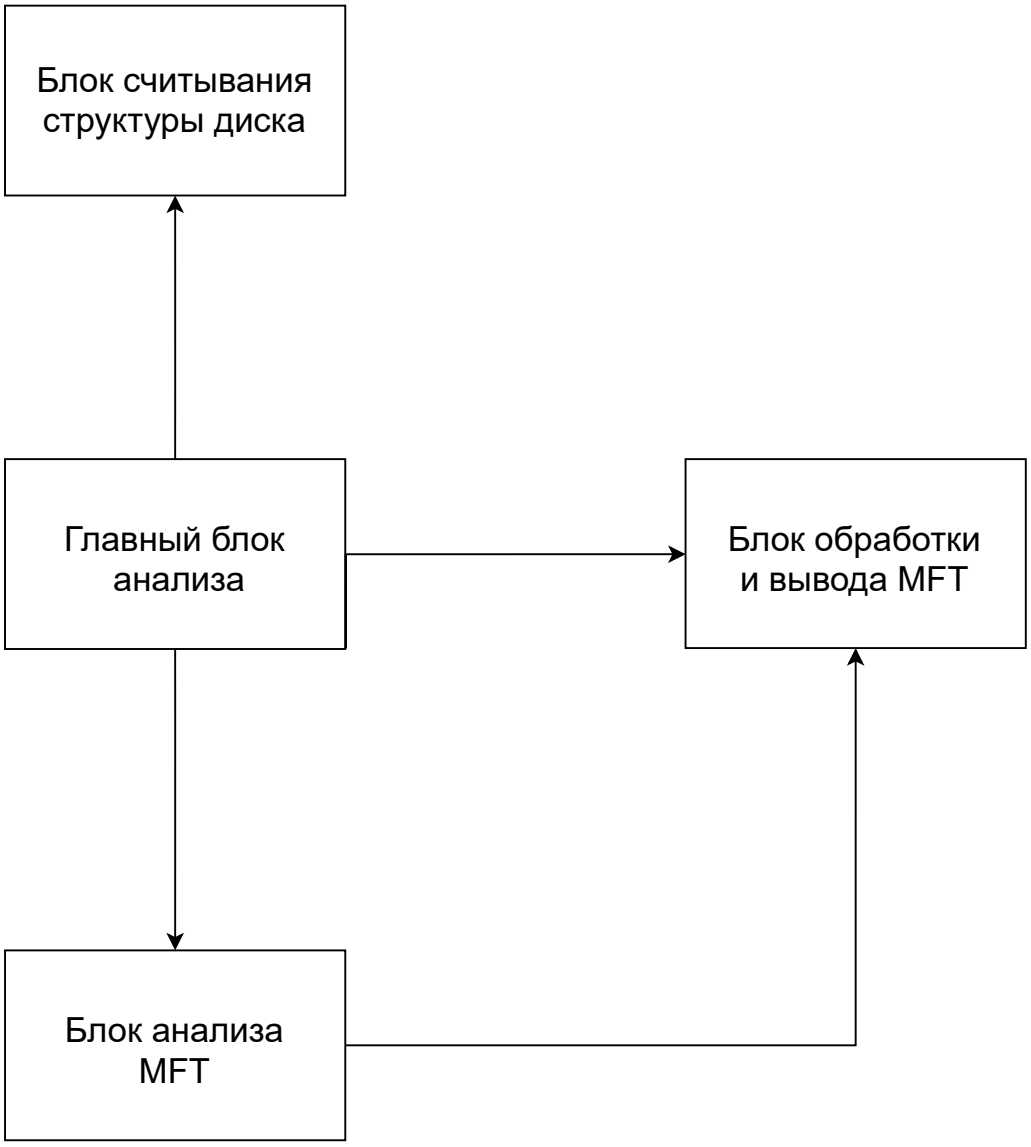
					ГУИР.400201.315 Д2	
Изм	Лист	№ документа	Подпись	Дата		
Разраб.		Липский			Блок-схема	Лит. У / Масса / Масштаб
Пров.		Глоба				Лист 1 / Листов 1
					ЭВМ, гр. 050503	

Формат А4

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема функциональных блоков программы



					ГУИР.400201.315 Д1		
Изм	Лист	№ документа	Подпись	Дата	Схема функциональных блоков программы	Лит.	Масса / Масштаб
Разраб.		Липский				У	
Пров.		Глоба				Лист 1	Листов 1
					ЭВМ, гр. 050503		

Формат А4