

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 проекту
на 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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
ОБЗОР ИСТОЧНИКОВ	5
СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ	7
ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	9
РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	11
ТЕСТИРОВАНИЕ	17
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	21
ЗАКЛЮЧЕНИЕ	23
ОБЗОР ЛИТЕРАТУРЫ	24

ВВЕДЕНИЕ

NTFS — основная файловая система в последних версиях Windows — предоставляет полный набор возможностей, включая дескрипторы безопасности, шифрование, дисковые квоты и расширенные метаданные. NTFS использует файл журнала и сведения о контрольных точках для восстановления согласованности файловой системы при перезагрузке компьютера после сбоя системы. После ошибки поврежденного сектора NTFS динамически изменяет конфигурацию кластера, содержащего поврежденный сектор, выделяет новый кластер для данных, отмечает исходный кластер как поврежденный и больше не использует старый кластер.

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. Никаких аномалий кластерной структуры NTFS не имеет, поэтому на эту, в общем-то, довольно банальную тему, сказать особо нечего.

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.

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 несут 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. 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.

Данная работа посвящена разработке утилиты проверки целостности файловой системы NTFS.

Задачи данного 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 ИСТОЧНИКОВ

1.1 Анализ аналогов программного средства

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.

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.

1.2 Постановка задачи

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.

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

2.1 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.

2.2 Структура приложения

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

Одним из них является блок просмотра информации о жёстком диске. Данный блок включает в себя следующие функции: `GetDriveGeometry`, которая собирает информацию о диске и `printgeometry`, которая эту информацию выводит на экран.

Можно выделить блок просмотра информации о структуре MFT. Сюда можно включить функцию DeviceIoControl, которая заполняет NTFS_VOLUME_DATA_BUFFER, если её вызвать с кодом FSCTL_GET_NTFS_VOLUME_DATA.

Ещё один блок в программе – это блок считывания записей из MFT, которое обеспечивает функции readRecord и findAttribute.

И последним в этом списке является блок копирования информации из MFT. Блок разделён на две части: для работы с резидентными данными и нерезидентными. В первом случае используются функции findAttribute и WriteFile, во втором случае используются функции seek, ReadFile и WriteFile.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Описание функционирования программы.

Программа запускается из командной строки с двумя или четырьмя аргументами. Два первых аргумента обязательные: первый аргумент – название исполняемого файла, второй – буква раздела диска (например С, D, Е 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.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Алгоритм 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)` по шагам:

Данная функция необходима для определения резидентности данных.

1. `readRecord(h, recordIndex, MFTRunList, recordSize, clusterSize, sectorSize, &record[0]).`

Считываем запись из MFT-зоны.

2. `RecordHeader* recordHeader = (RecordHeader*)&record[0].`

Инициализируем указатель на структуру `RecordHeader`.

3. `AttributeHeaderNR* attrListNR (AttributeHeaderNR*) findAttribute (recordHeader, recordSize, 0x20).`

Получаем список атрибутов (`$Attribute_List`).

4. `if (attrListNR == NULL)` шаги 5-12, иначе шаги 13-43.

5. `AttributeHeaderNR* headerNR =(AttributeHeaderNR*) findAttribute (recordHeader, recordSize, typeID);`

6. `if (headerNR->formCode == 0)` Шаг 7. Иначе шаги 8-12.

7. `stage = 1.` Атрибут резидентный.

8. `stage = 2.` Атрибут нерезидентный.

9. `vector<Run> runListTmp = parseRunList (LPBYTE(headerNR) + headerNR-> runListOffset, headerNR->length - headerNR->runListOffset, totalCluster).`

Инициализируем `runListTmp` через функцию `parseRunList`.

10. `runList->resize(runListTmp.size()).`

Изменяем размер `vector<Run>* runList`.

11. `for (size_t i = 0; i < runListTmp.size(); i++) (*runList)[i] = runListTmp[i].`

Записываем значение `runListTmp` в `*runList`.

12. if (contentSize != NULL) *contentSize = headerNR->contentSize.

contentSize - поле структуры AttributeHeaderNR.

13. if (attrListNR->formCode == 0) шаги 14-17, иначе шаги 18-29.

14. stage = 3. Список атрибутов резидентный.

15. AttributeHeaderR* attrListR = (AttributeHeaderR*)attrListNR;

16. attrListData.resize(attrListR->contentLength);

17. memcpy(&attrListData[0], LPBYTE(attrListR) + attrListR->contentOffset, attrListR->contentLength);

Копирование attrListR->contentLength байт в attrListData[0].

18. stage = 4. Список атрибутов нерезидентный.

19. if (attrListNR->compressSize != 0) throw _T("Compressed non-resident attribute list is not supported");

20. vector<Run> attrRunList = parseRunList(LPBYTE(attrListNR) + attrListNR->runListOffset, attrListNR->length - attrListNR->runListOffset, totalCluster);

Инициализируем attrRunList через функцию parseRunList.

21. attrListData.resize(attrListNR->contentSize);

Изменяем размер vector<BYTE> attrListData.

22. for (Run& run : attrRunList) шаги 23-29.

23. if (LONGLONG p >= attrListNR->contentSize) break;

Некоторые кластеры зарезервированы.

24. seek(h, run.offset * clusterSize); Устанавливаем позицию.

25. for (LONGLONG i = 0; i < run.length && p < attrListNR->contentSize; i++) Цикл чтения, шаги 26-29.

26. if (!ReadFile(h, &cluster[0], clusterSize, &read, NULL) || read != clusterSize) throw _T("Failed to read attribute list non-resident data"); Если не считалось, то бросаем исключение.

27. LONGLONG s = min(attrListNR->contentSize - p, clusterSize);

Это есть (((attrListNR->contentSize - p) < (clusterSize)) ? (attrListNR->contentSize - p) : (clusterSize))

```

28. memcpy(&attrListData[p], &cluster[0], s); Копируем s байт.
29. p += s;
30. for (LONGLONG p = 0; p + sizeof(AttributeRecord) <=
attrListData.size(); p += attr-> recordLength); Цикл, шаги 31-42
31. attr = (AttributeRecord*)&attrListData[p];
32. if (attr->typeID == typeID) шаги 33-42
33. vector<BYTE> extRecord(recordSize);
RecordHeader* extRecordHeader = (RecordHeader*)&extRecord[0];
34. readRecord(h, attr->recordNumber & 0xffffffffffffLL,
MFTRunList, recordSize, clusterSize, sectorSize, &extRecord[0]);
35. if (memcmp(extRecordHeader->signature, "FILE", 4) != 0)
throw _T("Extension record is invalid"); Исключение.
36. AttributeHeaderNR* extAttr = (AttributeHeaderNR*)findAttribute
(extRecordHeader, recordSize, typeID);
Инициализация extAttr с помощью функции findAttribute.
37. if (extAttr == NULL) throw _T("Attribute is not found in
extension record"); Исключение.
38. if (extAttr->formCode == 0) throw _T("Attribute in extension
record is resident");
39. vector<Run> runListTmp = parseRunList(LPBYTE(extAttr) +
extAttr -> runListOffset, extAttr-> length - extAttr->
runListOffset, totalCluster); Инициализация runListTmp.
40. runList->resize(runNum + runListTmp.size());
41. for (size_t i = 0; i < runListTmp.size(); i++) (*runList)
[runNum + i] = runListTmp[i];
42. runNum += runListTmp.size();

```

4.2 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 аргументом функции `DeviceIoControl`, которую вызываем после `CreateFileW`. Результат выполнения функции заносим в `BOOL bResult`, которую и возвращает функция `GetDriveGeometry`.

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

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

4.4. Принцип работы функции `LPBYTE findAttribute(RecordHeader* record, DWORD recordSize, DWORD typeId)`.

Функция нужна для определения списка атрибутов или определения их отсутствия. Используется в функции `readList`, описанной в пункте 4.1.

4.5. Принцип работы функции

`CommandLineToArgvW(GetCommandLine(), &argc)`.

Функция заголовочного файла `shellapi.h`. Анализируют введенные 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`.

4.6. 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.

4.7. 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.

4.8. Принцип работы функции `BOOL ReadFile(hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, 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. lpNumberOfBytesRead - 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.

4.9. Принцип работы функции `void fixRecord(BYTE* buffer, DWORD recordSize, DWORD sectorSize)`.

Функция запоминает имя файла, которое передаётся первым аргументом. Также функция запоминает порядковый номер записи.

4.10. 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`.

4.11. 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.

5 ТЕСТИРОВАНИЕ

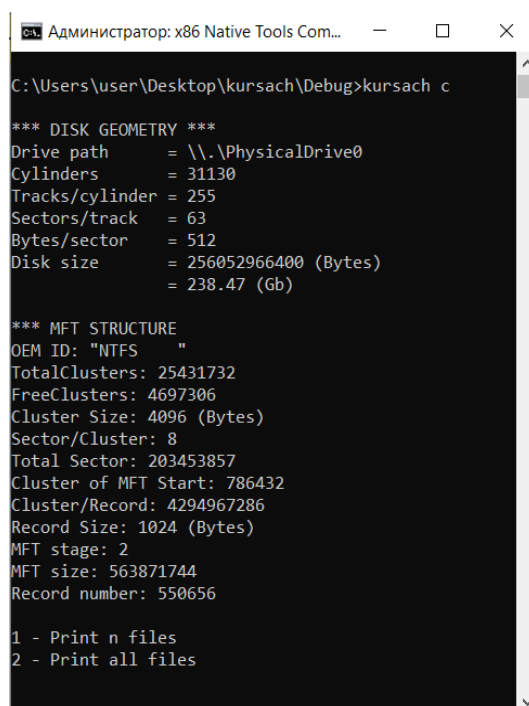
Тестирование является одним из важнейших этапов жизненного цикла, направленным на повышение качественных характеристик. Качество программного средства очень сильно зависит от того, насколько хорошо он выполняет задачи, для которых был создан. Скрытые ошибки могут сильно изменить результат работы программного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

Тестирование - 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.

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

Запуск утилиты с двумя аргументами (рис. 5.1).



```
Администратор: x86 Native Tools Com...
C:\Users\user\Desktop\kursach\Debug>kursach c

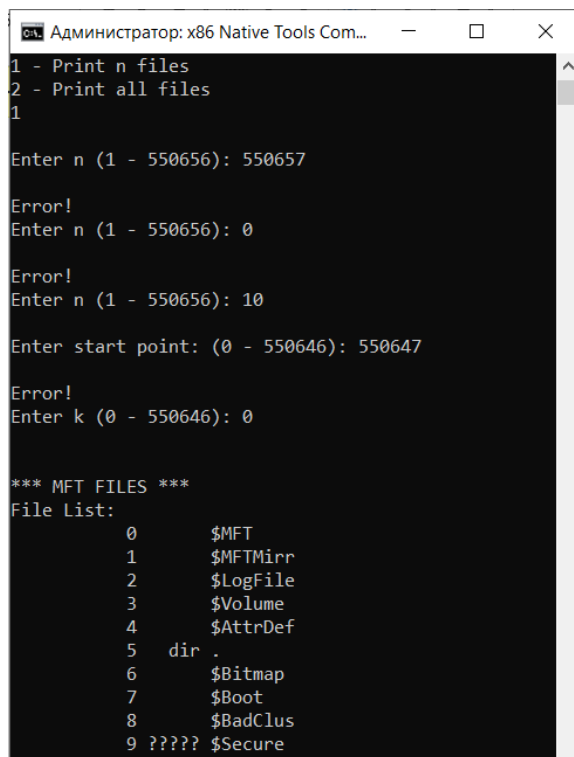
*** DISK GEOMETRY ***
Drive path      = \\.\PhysicalDrive0
Cylinders       = 31130
Tracks/cylinder = 255
Sectors/track   = 63
Bytes/sector    = 512
Disk size       = 256052966400 (Bytes)
                 = 238.47 (Gb)

*** MFT STRUCTURE
OEM ID: "NTFS"
TotalClusters: 25431732
FreeClusters: 4697306
Cluster Size: 4096 (Bytes)
Sector/Cluster: 8
Total Sector: 203453857
Cluster of MFT Start: 786432
Cluster/Record: 4294967286
Record Size: 1024 (Bytes)
MFT stage: 2
MFT size: 563871744
Record number: 550656

1 - Print n files
2 - Print all files
```

Рисунок 5.1

Просмотр записей MFT (рис. 5.2).



```
Администратор: x86 Native Tools Com...
1 - Print n files
2 - Print all files
1

Enter n (1 - 550656): 550657

Error!
Enter n (1 - 550656): 0

Error!
Enter n (1 - 550656): 10

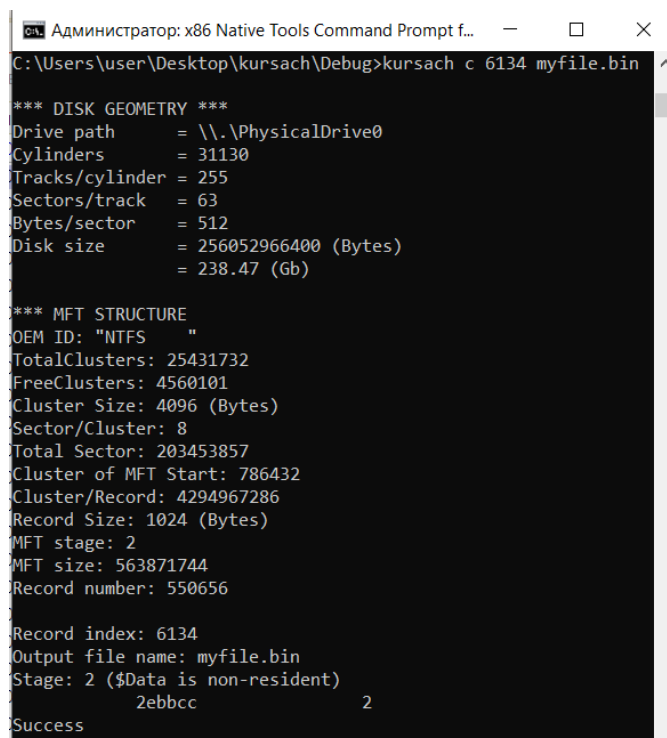
Enter start point: (0 - 550646): 550647

Error!
Enter k (0 - 550646): 0

*** MFT FILES ***
File List:
      0      $MFT
      1      $MFTMirr
      2      $LogFile
      3      $Volume
      4      $AttrDef
      5      dir .
      6      $Bitmap
      7      $Boot
      8      $BadClus
      9      $Secure
```

Рисунок 5.2

Запуск утилиты с четырьмя аргументами (рис. 5.3).



```
Администратор: x86 Native Tools Command Prompt f...
C:\Users\user\Desktop\kursach\Debug>kursach c 6134 myfile.bin

*** DISK GEOMETRY ***
Drive path      = \\.\PhysicalDrive0
Cylinders       = 31130
Tracks/cylinder = 255
Sectors/track   = 63
Bytes/sector     = 512
Disk size       = 256052966400 (Bytes)
                = 238.47 (Gb)

*** MFT STRUCTURE ***
OEM ID: "NTFS"
TotalClusters: 25431732
FreeClusters: 4560101
Cluster Size: 4096 (Bytes)
Sector/Cluster: 8
Total Sector: 203453857
Cluster of MFT Start: 786432
Cluster/Record: 4294967286
Record Size: 1024 (Bytes)
MFT stage: 2
MFT size: 563871744
Record number: 550656

Record index: 6134
Output file name: myfile.bin
Stage: 2 ($Data is non-resident)
                2ebbcc                2
Success
```

Рисунок 5.3

Копирование резидентной записи MFT в файл (рис. 5.4).

```

Администратор: x86 Native Tools Command Prompt for VS 2019

C:\Users\user\Desktop\kursach\Debug>kursach c 274798 myfile.bin

Record index: 274798
Output file name: myfile.bin
Stage: 1 ($Data is resident)
File size: 129
Success
  
```

Рисунок 5.4.

Сравнение файлов (рис. 5.5).

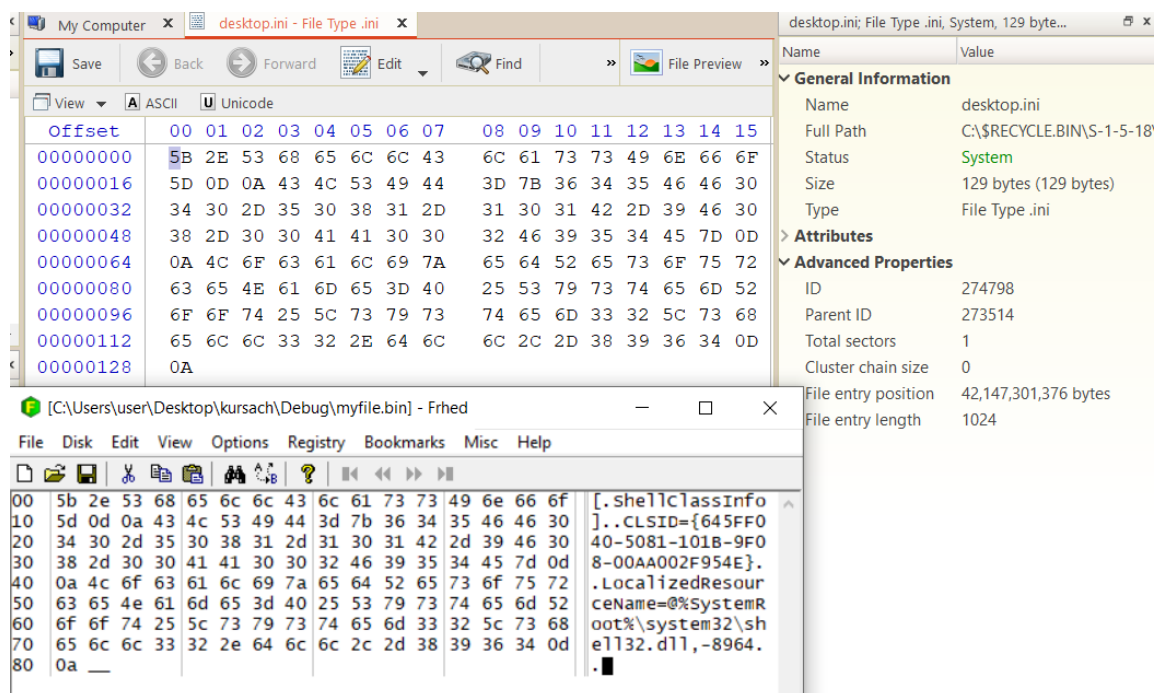


Рисунок 5.5

Копирование нерезидентной записи MFT в файл (рис. 5.6).

```

Администратор: x86 Native Tools Command Prompt for VS 2019

C:\Users\user\Desktop\kursach\Release>kursach c 0 MFT.bin

Record index: 0
Output file name: MFT.bin
Stage: 2 ($Data is non-resident)

c0000 c820
9cbe91 5961
b2de14 b80
107682c 43c0
cf8520 12bf
eca590 9640
  
```

Рисунок 5.6

Сравнение файлов (рис. 5.7).

The screenshot displays two windows from a file comparison tool. The top window, titled 'My Computer', shows a hex dump of a file named 'MFT.bin'. The hex dump is organized into columns labeled 'Offset' and 'Data'. The 'Offset' column shows values from 00000000 to 00000256. The 'Data' column shows hex values and their corresponding ASCII representations. The bottom window, titled '\$MFT - File', shows the file's properties. The properties include:

- General Information:** Name (\$MFT), Full Path (C:\\$MFT), Status (System), Size (16.0 KB (16,384 bytes)), Type (File).
- Attributes:** File Attributes (Hidden, System, Not Pu), Created (27.08.2020 14:08), Modified (27.08.2020 14:08), Accessed (Deleted) (27.08.2020 14:08).
- Advanced Properties:** ID (0), Parent ID (5), Total sectors (32), Cluster chain size (6), File entry position (3,221,225,472 bytes), File entry length (1024).
- Media Info:** File Record in \$MFT (0), Modified (2020-08-27T11:08:33), Record Date/Time (2020-08-27T11:08:33), Last Read (2020-08-27T11:08:33), File Size (16384), File Name (\$MFT).

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.

6 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. 6.1).

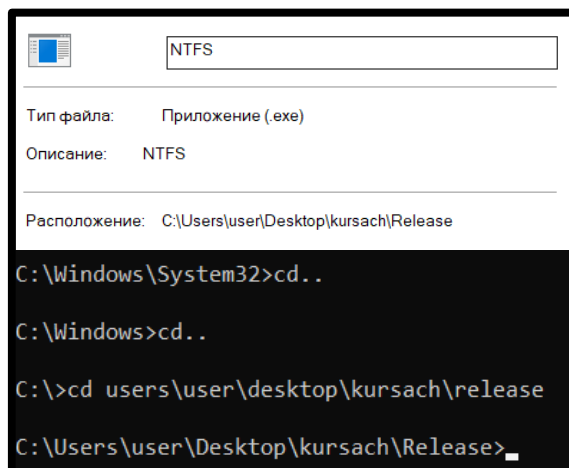


Рисунок 6.1

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 5.1 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 6.2.).

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 $Txfllog
31  dir $Txf
32  $Tops
33  $Txfllog.blf
34  $TxfllogContainer00000000000000000001
35  $TxfllogContainer00000000000000000002
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 6.2

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.

ОБЗОР 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>.

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

Метафайлы находятся в 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.

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

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.

[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/winiocctl/ns-winiocctl-fsctl_get_ntfs_volume_data.

https://docs.microsoft.com/en-us/windows/win32/api/winiocctl/ns-winiocctl-ntfs_volume_data_buffer.

С помощью этого блока кода можно определить фактический объём жесткого диска и его структуру (количество цилиндров, дорожек, секторов).

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

Первые 16 записей MFT резервируются для файлов метаданных NTFS. 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. Для латинских букв A — Z соответствие

регистра очевидно (по крайней мере для тех, кто разговаривает на романских языках). Однако соответствие регистров для других языков (таких, как греческий, армянский или грузинский) для говорящих на романских языках не столь очевидно, поэтому данный файл рассказывает, как это сделать. 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.

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).

[6] habr.com - информационный сервис. - [Электронный ресурс]. - Режим

доступа: <https://habr.com/ru/post/164193/>

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

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 хэндл. В случае ошибки возвращается специальное значение INVALID_HANDLE_VALUE.

2. ReadFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);

Чтение из файла в буфер lpBuff размером dwBuffSize. В переменную dwCount записывается реальное количество прочитанных байт.

3. WriteFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);

Аргументы и семантика процедуры WriteFile полностью аналогичны ReadFile.

4. CloseHandle(hFile);

Файловые дескрипторы закрываются с помощью CloseHandle.

[8] citforum.ru - информационный сервис. - [Электронный ресурс]. - Режим доступа: http://citforum.ru/operating_systems/windows/ntfs/2.shtml

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`).