Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Кафедра «Компьютерные системы и программные технологии»**

**К У Р С О В А Я Р А Б О Т А**

**Методы размещения, доступа и учёта свободного пространства в NTFS, NTFS-3G**

по дисциплине «Операционные системы»

Выполнил

студент гр. 3530901/70203

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванов И.Д.

(подпись)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург

2020

Санкт-Петербургский политехнический университет Петра Великого

**ЗАДАНИЕ**

**НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА**

**(КУРСОВОЙ РАБОТЫ)**

студенту группы 3530901/70203 Иванову Илье Дмитриевичу

*(номер группы)* *(фамилия, имя, отчество)*

1. ***Тема проекта (работы):*** Методы размещения, доступа и учёта свободного пространства в NTFS, NTFS-3G
2. ***Срок сдачи законченного проекта (работы):*** 31.01.2020

***3. Содержание пояснительной записки***:введение,основнаячасть (методы размещения, методы доступа, методы учёта свободного пространства), заключение, список используемых источников.

***Дата получения задания***: « 15 »ноября2019г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

*(подпись)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванов И.Д.

*(подпись студента)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(дата)*

**Оглавление**

[**Введение** 4](#_Toc30515962)

[**Основная часть** 4](#_Toc30515963)

[**1.** **Методы размещения** 4](#_Toc30515964)

[**1.1.** **Теоретические сведения** 4](#_Toc30515965)

[**1.2.** **Практика** 13](#_Toc30515966)

[**2.** **Методы доступа** 18](#_Toc30515967)

[**3.** **Методы учёта свободного пространства** 20](#_Toc30515968)

[**3.1.** **Теоретические сведения** 20](#_Toc30515969)

[**3.2.** **Практика** 22](#_Toc30515970)

[**Заключение** 24](#_Toc30515971)

[**Список использованных источников** 24](#_Toc30515972)

# **Введение**

**NTFS** (new technology file system) — стандартная [файловая система](https://ru.wikipedia.org/wiki/Файловая_система) для семейства [операционных систем](https://ru.wikipedia.org/wiki/Операционная_система) [Windows NT](https://ru.wikipedia.org/wiki/Windows_NT) фирмы [Microsoft](https://ru.wikipedia.org/wiki/Майкрософт).

**NTFS-3G** — [свободный](https://ru.wikipedia.org/wiki/Свободное_программное_обеспечение) [драйвер](https://ru.wikipedia.org/wiki/Драйвер) файловой системы [NTFS](https://ru.wikipedia.org/wiki/NTFS). На данный момент заявлена работа в следующих ОС: [Linux](https://ru.wikipedia.org/wiki/Linux), [FreeBSD](https://ru.wikipedia.org/wiki/FreeBSD), [NetBSD](https://ru.wikipedia.org/wiki/NetBSD), [Solaris](https://ru.wikipedia.org/wiki/Solaris), [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X), [BeOS](https://ru.wikipedia.org/wiki/BeOS) и [Haiku](https://ru.wikipedia.org/wiki/Haiku).

В любой Файловой системе решаются следующие задачи:

* Размещение файлов
* Доступ к файлам
* Управление свободным пространством

Для решения данных задач существуют различные подходы и механизмы, использование которых обусловлено особенностями ОС и спецификой конкретных ситуаций. В данной работе будут подробно рассмотрены методы, с помощью которых приведённые задачи решаются в NTFS.

Практическая часть работы получена с использованием NTFS-3G на следующей системе:

asdf@BigAwesomeTurtle:~/Documents/course\_os$ uname -a

Linux BigAwesomeTurtle 5.2.0-kali2-686-pae #1 SMP Debian 5.2.9-2kali1 (2019-08-22) i686 GNU/Linux

Дистрибутив – Kali Linux

# **Основная часть**

# **Методы размещения**

## **Теоретические сведения**

Существует множество методов, решающих задачу размещения файлов. Они делятся на методы предварительного и динамического размещения.

При предварительном размещении должен быть заранее известен максимальный размер файла. Пространство на диске выделяется исходя из этого размера. Данный метод используется при компиляции файлов, при передаче по сети, а также при хранении системный файлов.

При динамическом размещении само размещение осуществляется порциями.

Порцией называется непрерывный участок дискового пространства, состоящий из одного или нескольких кластеров (блоков). Конкретный размер порции определяется как компромисс между эффективной работой с одним файлом и общей эффективностью работы файловой системы.

При выборе размера порции должны учитываться следующие условия:

- непрерывность пространства диска (используемого и неиспользуемого) повышает общую производительность;

- наличие большого количества небольших порций увеличивает размер таблицы размещения файлов;

- наличие порций фиксированного размера (например, порция = блоку) упрощает перераспределение пространства;

- наличие порций переменной длины или малого размера минимизирует распределенное, но фактически неиспользуемое пространство (т.е. снижает фрагментацию).

Анализ перечисленных условий позволяет определить два наиболее оптимальных варианта распределения пространства:

Большие непрерывные порции переменной длины:

(+) Повышение производительности;

(+) Минимизация фрагментации;

(+) Таблицы размещения файлов небольшие;

(-) Трудности в повторном использовании выделенного пространства.

Фиксированные порции небольшого размера:

(+) Обеспечение большей гибкости в распределении пространства;

(+) Порции выделяются по мере необходимости, что позволяет эффективно организовывать операции по изменению файлов.

(-) Таблицы размещения файлов большие и сложно структурируемые;

(-) Непрерывность пространства диска фактически отсутствует;

Любой из рассмотренных вариантов с той или иной степенью эффективности совместим со стратегией как предварительного, так и динамического размещения файлов.

В реальных файловых системах используются следующие методы размещениям, основанные на сочетании преимуществ описанных стратегий и вариантов размещения:

1. Непрерывное размещение

Файл размещается предварительно как непрерывная группа блоков. Необходимость в таблице размещения файлов минимизируется: для каждого файла нужно зафиксировать лишь первый блок и общее количество выделенных блоков.

Данный метод наилучшим образом подходит для размещения последовательных файлов.

Основная проблема – внешняя фрагментация, которая усложняет поиск непрерывных порций подходящего размера. Для этого необходимо выполнять упаковку информации с корректным изменением таблицы размещения файлов.

2. Цепочечное размещение

Размещение производится по одному блоку, каждый предыдущий блок содержит указатель на следующий.

Каждая запись таблицы размещения файлов состоит из номера начального блока и количества блоков.

При необходимости дополнительного выделения памяти под файл в цепочку добавляется очередной блок.

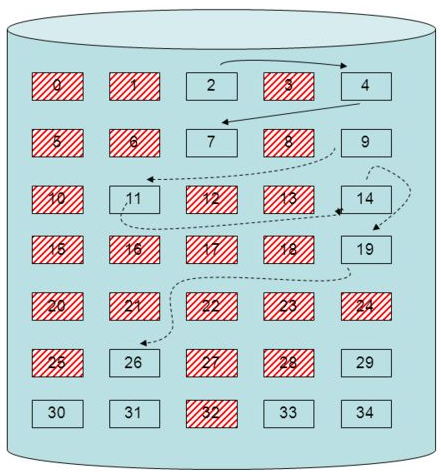


Рис.1.1. Цепочечное размещение

Внешняя фрагментация отсутствует, поскольку блоки выделяются по одному.

Наилучшим образом данный метод поддерживает размещение последовательных файлов.

Ключевыми недостатками являются:

- Невозможность одновременного доступа к нескольким блокам;

- Потеря «однородности» информации (каждый блок помимо собственной информации файла резервирует место и хранит служебную информацию о связях блоков).

ОС, поддерживающие данный метод, имеют служебные процедуры уплотнения (сжатия). После сжатия большинство файлов размещаются в непрерывные цепочки связанных блоков.

3. Индексированное с порциями блоков

Используются так называемые индекс-блоки. В таблице размещения имени файла ставится в соответствие блок, содержащий указатели на все блоки, соответствующие данному файлу. По мере необходимости файлу может выделен дополнительное пространство в виде нескольких доступных блоков.

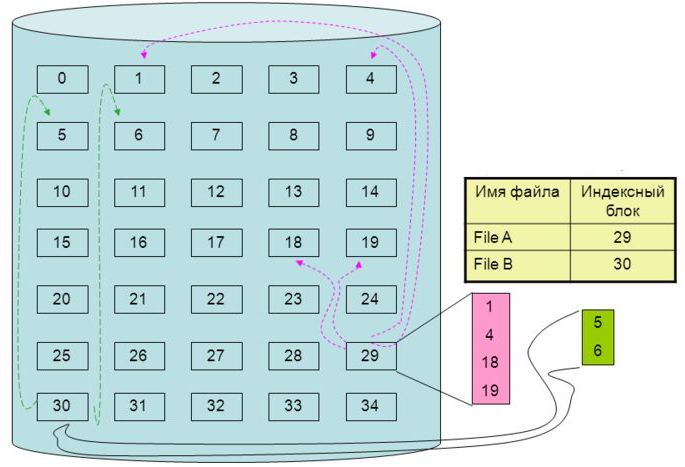


Рис.1.2. Индексированное размещение с порциями блоков

Способ является достаточно быстрым, однако минусом является необходимость хранить большой объём вспомогательной информации при большом числе блоков.

4. Индексированное с порциями переменного размера

При реализации варианта выделения пространства непрерывными порциями переменной длины имеется возможность одновременного доступа к нескольким блокам в рамках одной порции. Индексный блок содержит указатель на первый блок порции и ее длину. Таблица размещения файлов не усложняется.

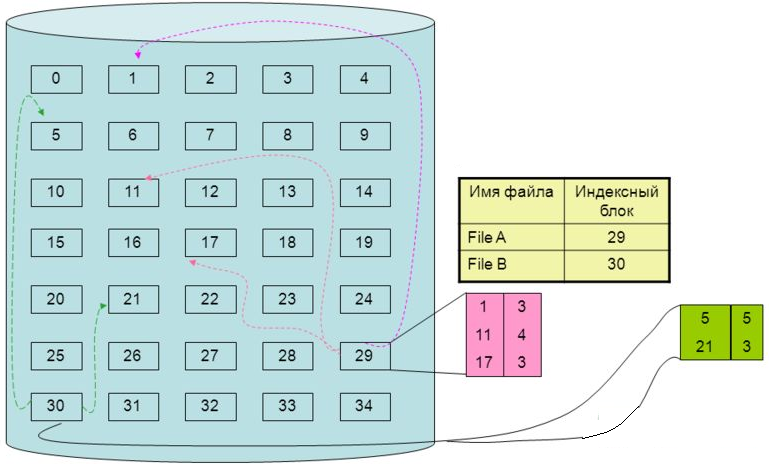


Рис.1.3. Индексированное размещение с порциями переменного размера

Таблица 1. Сравнение методов размещения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Непрерывный** | **Цепочечный** | **Индексированный** | |
| **Предварительное размещение** | Необходимо | Возможно | Возможно | |
| **Фиксированный/переменный размер порций** | Переменный | Фиксированный | Фиксированный | Переменный |
| **Размер порций** | Большой | Маленький | Маленький | Средний |
| **Частота размещений** | Одинарное  размещение | Любая | Высокая | Низкая |
| **Время размещения** | Среднее | Длительное | Короткое | Среднее |
| **Размер таблицы размещения файлов** | Маленький | Маленький | Большой | Средний |

Индексированный метод является наиболее универсальным, поскольку позволяет достаточно эффективно распределять пространство диска в условиях стратегии динамического размещения при обоих возможных вариантах: фиксированных блоках и непрерывных переменных порциях.

Индексированный метод поддерживает как последовательный, так и прямой доступ к файлам и, поэтому, является наиболее популярным методом файлового размещения и реализован во всех современных ОС.

В NTFS используется индексированный метод с порциями переменного размера с примесью цепочечного.

Структура NTFS:

Минимальной единицей размещения в NTFS является кластер, ФС не распознает физические сектора, поэтому NTFS без проблем поддерживает работу нестандартных дисков (у которых размер сектора не равен 512 байтам). Кластеры выделяются под размещение файлов по мере необходимости целым количеством. При этом допускается фрагментация – кластеры необязательно должны образовывать непрерывную порцию.

Максимальный размер файла, поддерживаемый NTFS, составляет 232 байт.

Том NTFS:

Файловая система использует концепцию «тома». Том – логический раздел диска, состоящий из некоторого количества кластеров и используемый ФС для распределения дискового пространства. Том состоит из служебной информации ФС, набора файлов и перераспределенного пространства для размещения новых файлов. Максимальный размер тома – 264 байт.

Каждый элемент тома представляет собой файл с определенным набором свойств (атрибутов). Даже данные, хранящиеся в том или ином файле являются его свойством (атрибутом).



Рис.1.4. Структура тома NTFS

Загрузочный сектор тома занимает несколько физических секторов (до 16) и содержит информацию о схеме тома и структурах файловой системы, начальную загрузочную запись и код загрузки.

Главная файловая таблица (Master File Table – MFT; $Mft) всегда представляет собой первый файл тома NTFS. Она содержит как минимум одну запись для каждого файла и каталога тома, включая запись для самой MFT. Каждая запись MFT имеет фиксированный размер, который определяется в момент форматирования диска и находится в диапазоне 1024–4096 байт.

Производительность системы существенно повышается, если записи MFT хранятся в соседних кластерах диска, т.е. когда MFT не фрагментирован. Для выполнения этого условия NTFS резервирует область (около 12% тома), которая называется зонойMFT*,* и стремится не использовать эту область ни для чего, кроме хранения записей MFT.

Первые 24 записи в MFT зарезервированы под метафайлы. Их описание приведено на Рис.1.5.

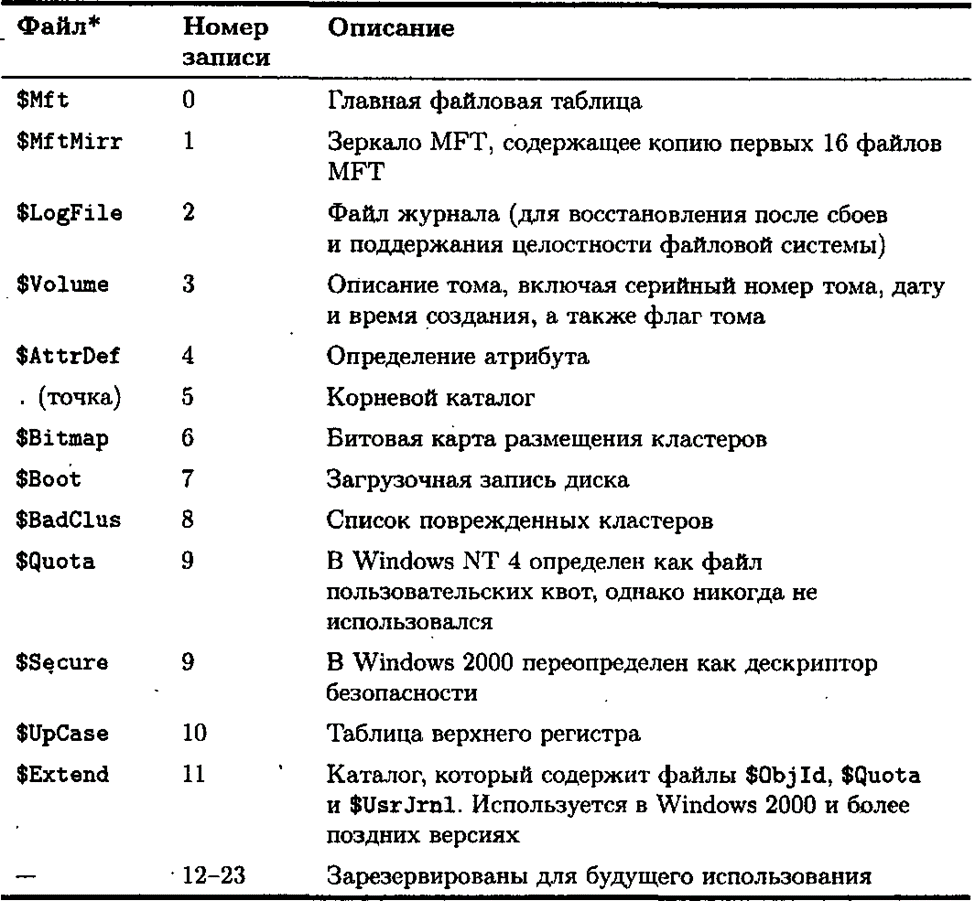
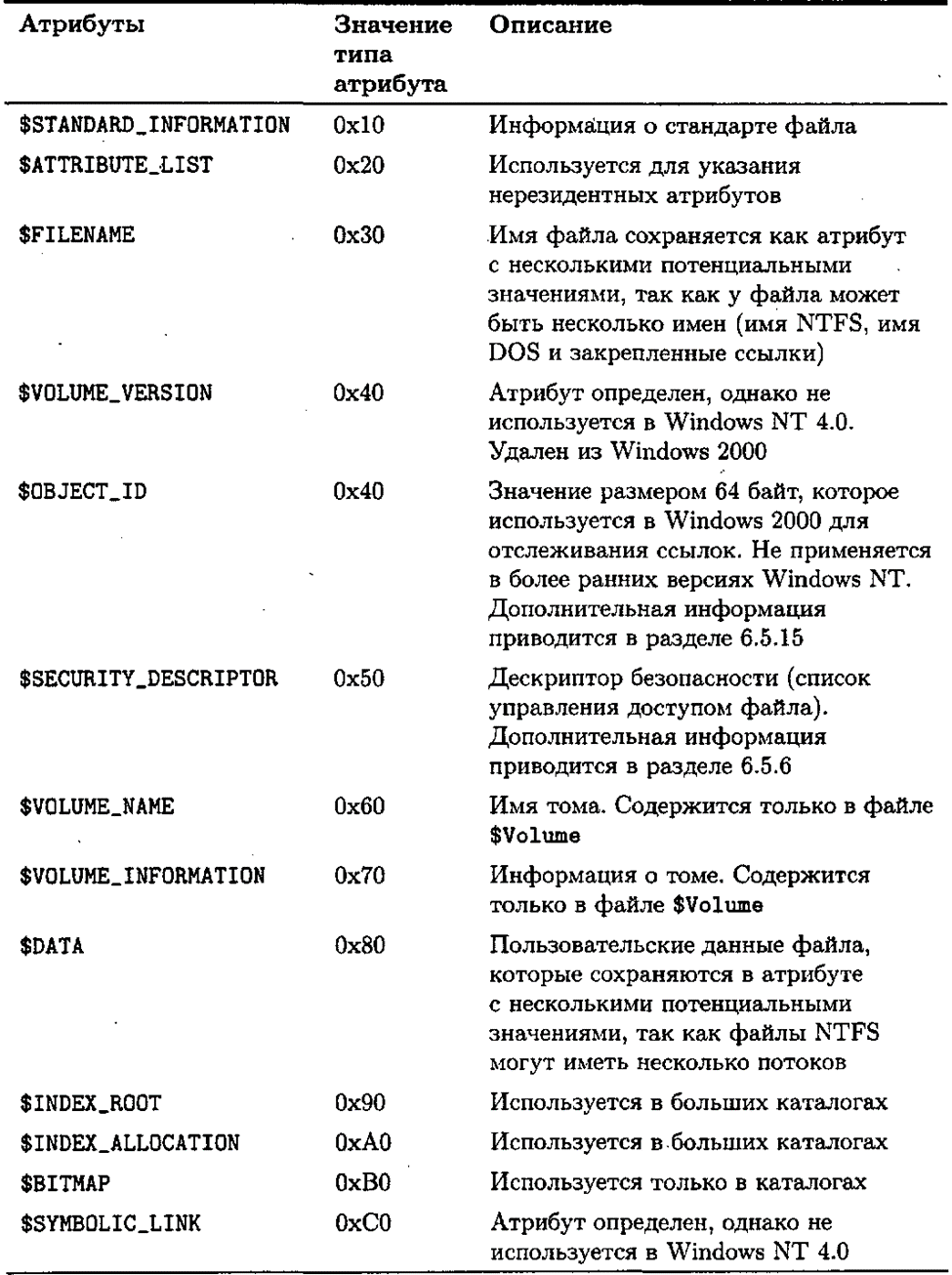


Рис.1.5. Метафайлы

Запись MFT содержит стандартный заголовок, после которого идёт последовательность атрибутов.

На Рис.1.6. приведена информация о различных атрибутах, которые может иметь файл в NTFS.

**

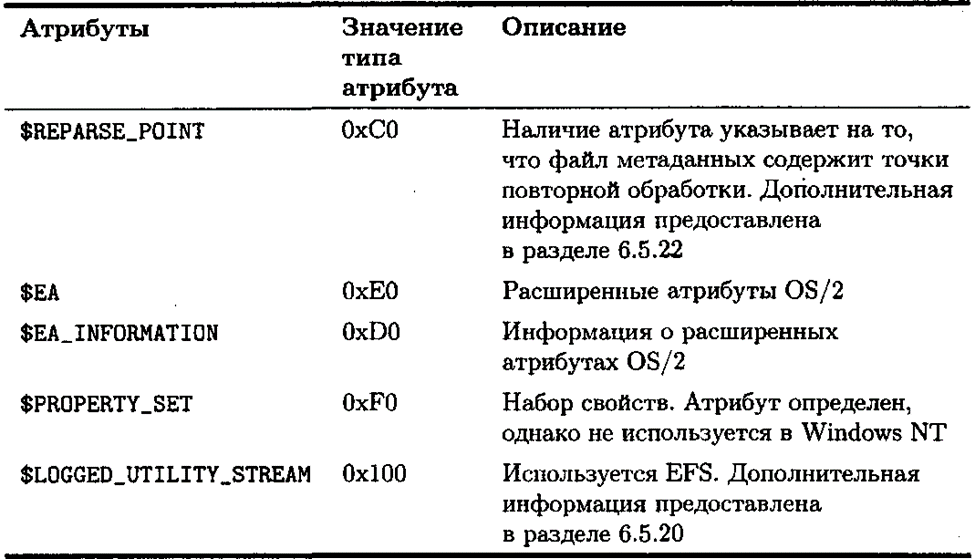


Рис.1.6. Атрибуты файла

Если данные атрибута имеют небольшой размер, они будут сохранены непосредственно в записи MFT. С другой стороны, если данные слишком велики для хранения в MFT, сохраняется информация о расположении этих данных - индексный кластер, в который помещаются указатели на начальные кластеры порций, распределенных под файл в области размещения файлов и длины порций, т.е. реализуется **индексированный метод с порциями переменного размера.**

Каталоги в NTFS также являются файлами. Файловая система хранит их способом, ускоряющим просмотр их содержимого. Если данные каталога не помещаются в MFT, то NTFS выделяет кластер и последовательную структуру, аналогичную области хранения данных. Все записи каталогов хранятся в деревьях В+.

ДеревоВ+ *–* это структура данных, которая поддерживает упорядоченное хранение информации и эффективна в контексте операций поиска, удаления, вставки и просмотра данных. Деревья В+ состоят из «узловых» записей, содержащих ключи, а также указателей, которые соединяют узлы дерева. Преимущество использования такой структуры данных состоит в тенденции к расширению дерева, а не к увеличению в глубину, что позволяет сохранить приемлемую производительность даже для каталогов, которые содержат большое количество записей.

Записи в каталоге хранятся в отсортированном виде. Каждая запись содержит имя файла, указатель на запись файла в MFT, а также дату/временную метку (эта информация уже хранится в записи файла MFT). Это позволяет добиться высокой производительности при просмотре содержимого каталога.

## **Практика**

Для демонстрации работы был создан образ диска с файловой системой NTFS (test.ntfs). Информация об исследуемой файловой системе приведена на Рис.1.7. и Рис.1.8.

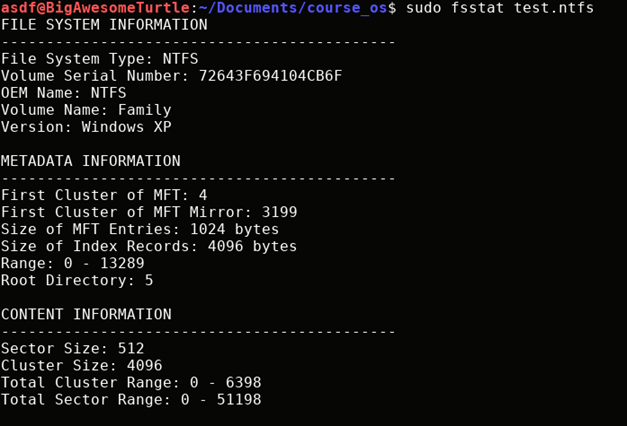


Рис.1.7. Информация об исследуемой файловой системе

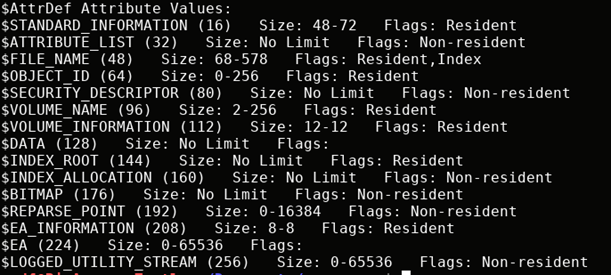


Рис.1.8. Информация об исследуемой файловой системе (Продолжение)

С помощью NTFS-3G и команды mount смонтируем образ, привязав его к loop-устройству:



Рис.1.9. Монтирование образа с привязкой к loop-устройству

Флаг -t команды mount позволяет указать тип файловой системы. В данном случае мы указываем ntfs-3g, сообщая, что мы хотим использовать NTFS-3G в качестве драйвера.

Флаг -o loop позволяет смонтировать образ, используя loop-устройство. Система найдёт свободное loop-устройство и привяжет к нему образ.

Loop-устройство – это псевдо-устройство, при использовании которого появляется возможность работать с файлом как с блочным устройством.

test.ntfs – образ с файловой системой NTFS, ntfs\_part -точка монтирования.

Информация о смонтированном диске:

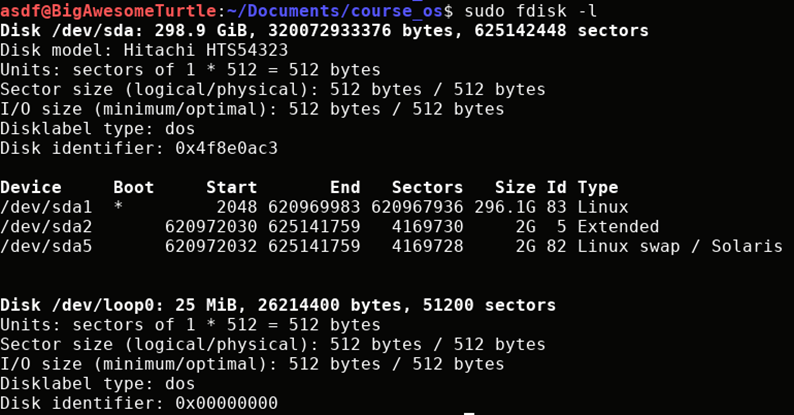
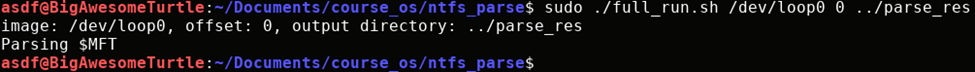


Рис.1.10. Информация о смонтированном диске

Получение доступа к метафайлам ($Mft, $Bitmap и т.д.) стандартными средствами является крайне проблематичным, поэтому информацию, содержащуюся в них, будем получать напрямую с диска и преобразовывать в удобочитаемый вид. Для этих целей используем программу, написанную на языке python (mft\_parse.py), а также скрипт full\_run.sh, который автоматизирует работу mft\_parse.py.

Запустим full\_run.sh:



В результате работы программы в директории parse\_res создалось 2 файла: mft.csv и mft.parsed. Первый представляет из себя csv таблицу, отображающую MFT, без учёта некоторых атрибутов (например, атрибута Data) для сохранения удобочитаемости таблицы. Второй содержит полную информацию обо всех файлах в текстовом виде.

Как уже было сказано, первые 24 записи в $Mft являются записями о метафайлах, одним из которых является сам $Mft. Рассмотрим запись о нём в файле mft.parsed:

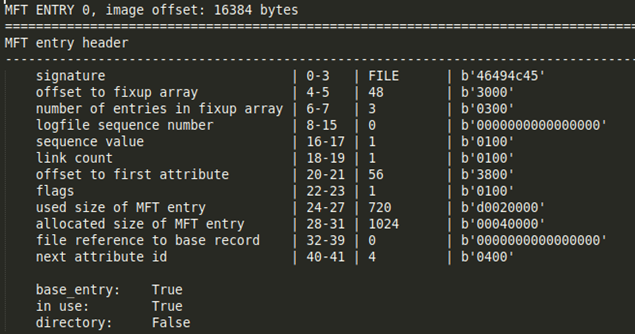


Рис.1.11. Информация о $Mft в самом $Mft

Из рисунка 1.11. можно увидеть, что файлы в NTFS начинаются со слова FILE. Также, можно обратить внимание на то, сколько места выделено под эту запись (1024 байта) и сколько используется (720 байт).

Далее идёт содержимое атрибутов файла:

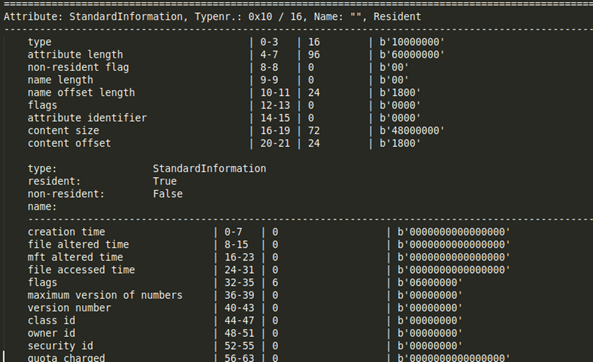


Рис.1.12. Атрибут StandardInformation

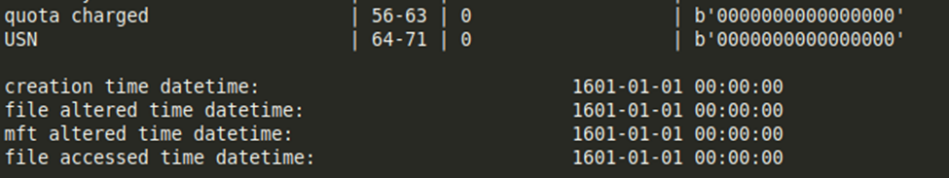


Рис.1.13. Атрибут StandardInformation(Продолжение)

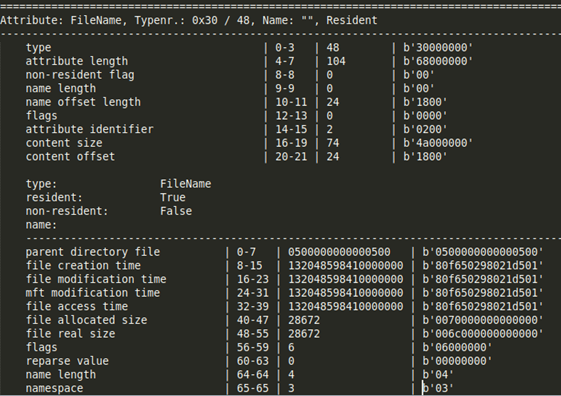


Рис.1.14. Атрибут FileName

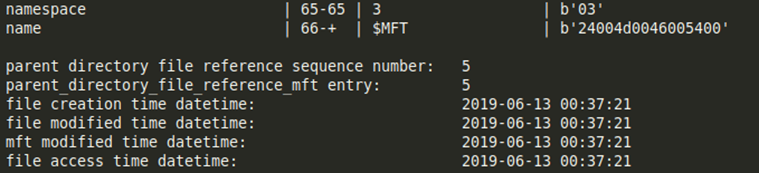
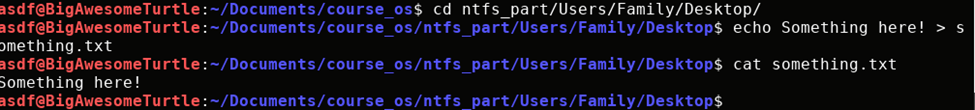


Рис.1.15. Атрибут FileName(Продолжение)

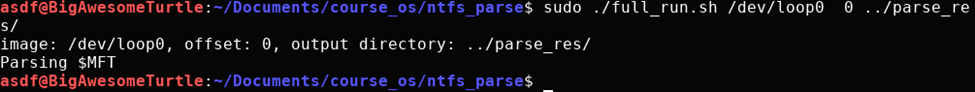
Из атрибута FileName можно узнать, что именем файла является $Mft.

Создадим файл something.txt в директории Users/Family/Desktop/ смонтированной системы:



При создании нового файла ему выделяется необходимое количество кластеров на диске. Система старается избежать фрагментации файлов и ищет пространство на диске с необходимым количеством кластеров пор ряд. Если найти такое не удаётся, система фрагментирует файл. Выбранные кластеры помечаются как занятые в файле $Bitmap (подробнее о нём в методах управления свободным пространством). В $Mft создаётся запись о файле или несколько записей, в том случае, если файл был фрагментирован. Маленькие файлы (размер которых меньше объёма памяти, выделенного на запись в MFT) хранятся прямо в $Mft, в то время как у остальных в атрибуте Data хранится список пар вида: номер первого кластера в порции, длина порции, т.е. используется **индексированный метод с порциями переменного размера с примесью цепочечного.**

Запустим full\_run.sh ещё раз для получение обновлённой $Mft:



В $Mft появилась новая запись:



Из атрибута FileName можно узнать, что имя файла, соответствующего данной записи - something.txt:



Атрибут Data содержит текст, хранящийся в файле:



Файл является весьма маленьким, следовательно, его данные хранятся прямо в $Mft (атрибут Data является резидентным).

При создании файла, не помещающегося целиком в $Mft, атрибут Data будет иметь следующий вид:

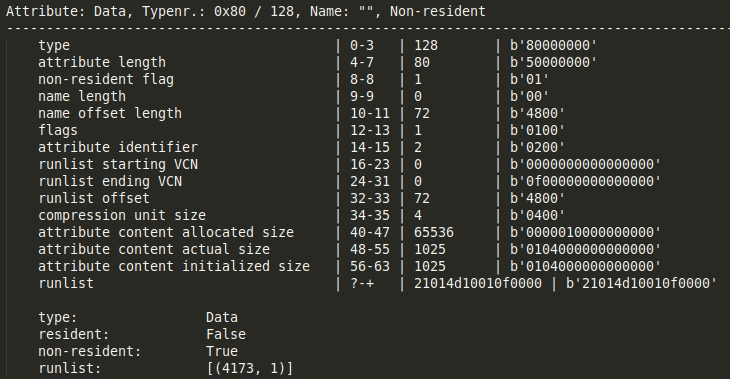
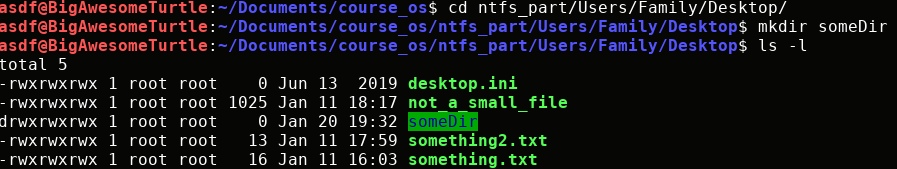


Рис.1.16. Атрибут Data файла, не поместившегося в $Mft

Легко заметить, что файлу соответствует порция с началом в 4173 кластере и длиной в 1 кластер.

Создадим каталог и посмотрим на его запись в $Mft:



Содержимое атрибута FileName:

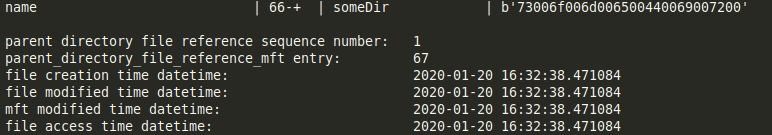
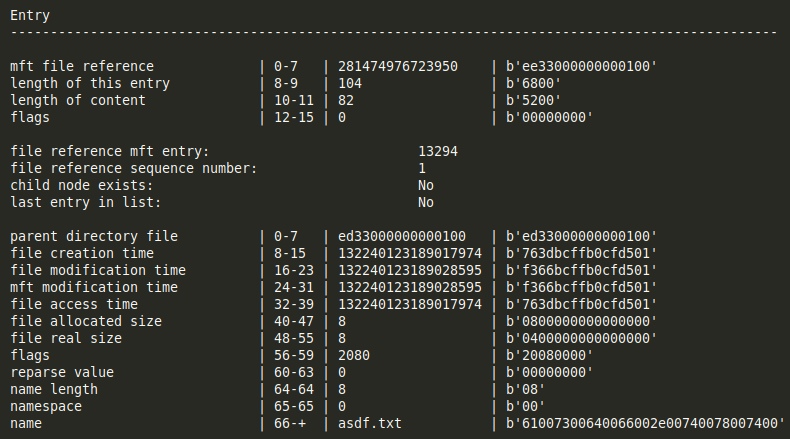


Рис.1.17. Содержимое атрибута FileName каталога

В атрибуте IndexRoot не содержится ни одного Entry, так как директория является пустой.

Содержимое атрибута IndexRoot после создания файла в директории:



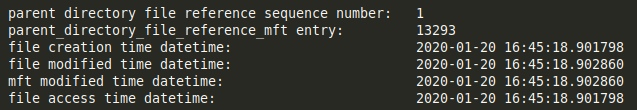


Рис.1.18. Содержимое атрибута IndexRoot после создания файла в директории

В атрибуте IndexRoot появилась информация о добавленном файле, включая его имя и индекс в MFT.

# **Методы доступа**

В общем случае данные, содержащиеся в файле, имеют логическую структуру. Эта структура является базой при разработке программы, предназначенной для обработки этих данных. Поддержание структуры данных может быть возложено на приложение, либо, в определенной степени, эту работу может выполнять файловая система. Если поддержание структуры возложено на приложение, то файл представляется файловой системе неструктурированной последовательностью байт. Приложение формулирует запросы к файловой системе на ввод-вывод, используя общие для всех приложений системные средства, например, указывая смещение от начала файла и количество байт, которые необходимо считать или записать. Поступивший к приложению поток байт интерпретируется в соответствии с заложенной в программе логикой.

Другая модель файла – структурированный файл применялась в ранних операционных системах (OS/360, DEC RSX и VMS), в настоящее время используется редко. В этом случае поддержание структуры файла поручается файловой системе. Файловая система видит файл как упорядоченную последовательность логических записей. Приложение может обращаться к файловой системе с запросами на ввод-вывод на уровне записей. Файловая система должна обладать информацией о структуре файла, достаточной для того, чтобы выделить любую запись. Файловая система предоставляет приложению доступ к записи, а вся остальная обработка данных, содержащаяся в этой записи, выполняется приложением. Развитием этого подхода стали системы управления базами данных (СУБД), которые поддерживают сложную структуру данных и взаимосвязи между ними.

В случае структурированных файлов, логическая запись представляет собой наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. Файловая система может использовать два способа доступа к логическим записям: читать или записывать логические записи последовательно (последовательный доступ) или позиционировать файл на запись с указанным номером (прямой доступ).

Файловая система может поддерживать несколько схем логической организации файла (Рис.2.1.).



Рис.2.1. Cхемы логической организации файла

1) Представление данных в виде записей, длина которых фиксирована в пределах файла (Рис.2.1.(1)). В этом случае доступ к n-ой записи осуществляется либо путем последовательного чтения предшествующих записей, либо прямо по адресу, вычисленному по ее порядковому номеру.

2) Представление данных в виде последовательности записей, размер которых изменяется в пределах одного файла (Рис.2.1.(2)). В этом случае для поиска нужной записи система должна последовательно считать все предшествующие записи. Вычислить адрес нужной записи по ее номеру при такой логической организации файла невозможно, поэтому не может быть применен более эффективный метод прямого доступа. Файлы, доступ к записям которых осуществляется последовательно, по номерам позиций, называются неиндексированными или последовательными.

3) Индексированные файлы, которые допускают прямой доступ к отдельной логической записи (Рис.2.1.(3)). В индексированном файле записи имеют одно или более ключевых полей и могут адресоваться путем указания этих значений. Для быстрого поиска данных в индексированном файле предусматривается индексная таблица, в которой значениям ключевых полей ставится в соответствие адрес внешней памяти. Этот адрес может указывать на искомую запись (прямой доступ) либо на область внешней памяти, занимаемой несколькими записями, в число которых входит и искомая запись (прямой и последовательный доступ). В последнем случае файл имеет индексно-последовательную организацию. Ведением индексных таблиц занимается файловая система.

В NTFS реализован индексированный метод размещения файлов, поэтому поддерживаются как прямой, так и последовательный методы доступа к файлам.

# **Методы учёта свободного пространства**

## **Теоретические сведения**

Любой файловой системе необходимо осуществлять управление свободным пространством диска.

Применяя любую из описанных технологий размещения файлов, ОС должна владеть оперативной информацией о том, какие блоки на диске доступны. Для этого, наряду с таблицей размещения файлов, необходима общая таблица (карта) дискового пространства.

Существуют четыре основных модели реализации такой информации, которые наиболее эффективно работают в условиях того или иного варианта размещения:

- Битовые таблицы;

- Цепочки свободных порций;

- Индексирование свободного пространства;

- Список свободных блоков.

1. Использование битовых таблиц

Все пространство диска, изначально разбитое на блоки, соотносится с одномерным массивом, элементами которого являются биты: «0» - соответствует свободному блоку, «1» - занятому.

Преимущества:

- Простота поиска как отдельного свободного блока, так и непрерывной группы свободных блоков (порции);

- Наименьший размер самой таблицы из всех возможных ее представлений.

Объём битовой карты (в байтах) =

Часто при использовании битовых карт диск делят на поддиапазоны, каждый из которых хранит число своих свободных блоков. Данный способ позволяет существенно увеличить скорость работы с битовой картой.

2. Цепочки свободных порций

Свободные порции (непрерывные участки из нескольких блоков) связываются ФС в цепочки. Для каждой порции запоминается ее длина в блоках и указатель на следующую свободную порцию. При такой организации необходимость в битовой таблице отпадает, но повышается фрагментация диска и исключается возможность доступа сразу же к нескольким свободным порциям, что может привести к существенному замедлению операций размещения файлов (особенно большого размера) и освобождения пространства. Несмотря на то, что данный способ работает для всех методов размещения файлов, наиболее эффективно он применяется при методе размещения непрерывными порциями переменной длины.

3. Индексирование

При индексировании свободное пространство диска рассматривается как индексированный файл. Каждая свободная порция переменной длины имеет уникальный элемент (индекс) в индексной таблице, по которому осуществляется доступ. Данный подход обеспечивает эффективную поддержку всех методов размещения файлов.

4. Список свободных блоков (UNIX)

Каждому блоку диска присваивается порядковый номер, а список номеров всех свободных блоков содержится в зарезервированной области на диске.

Преимуществом данного способа является однозначная идентификация (по уникальному номеру) всех свободных блоков, что позволяет эффективно поддерживать цепочечный метод размещения файлов.

Однако, при таком подходе, в зависимости от размера диска, для хранения порядкового номера одного блока необходимо 24 или 32 бита. Т.е. размер списка свободных блоков получается значительно большим, чем размер битовой таблицы.

Существуют модели достаточно эффективного использования подхода «список номеров свободных блоков», которые базируются на помещении в основную память только некоторой части списка и ее динамического изменения по мере необходимости.

В NTFS используется битовая таблица (файл $Bitmap).

## **Практика**

Файл $Bitmap представляет из себя битовое поле, в котором каждому биту поставлен в соответствие кластер диска. При размещении файла выбранные кластеры помечаются как занятые в $Bitmap. При удалении файла занятые им кластеры просто помечаются как свободные, поэтому в свободном кластере также могут находиться данные. Если такой кластер понадобится в дальнейшем системе, данные в нём просто перезапишутся.

Посмотрим на изменения в файле $Bitmap при создании нового файла. Для этого воспользуемся всё той-же программой mftparse.py:



Рис.3.1. Извлечение файла $Bitmap

При таком вызове программа извлечёт данные 6-ой по индексу записи из MFT (файл $Bitmap) и запишет их в файл bitmap\_extracted.

Создадим ещё один маленький файл, something2.txt и сравним содержимое $Bitmap до и после создания файла:

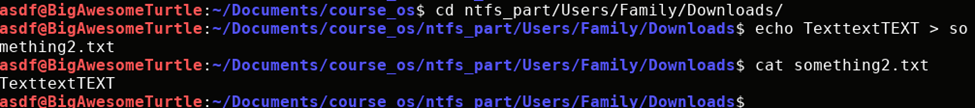




Рис.3.2. Извлечение изменённого $Bitmap

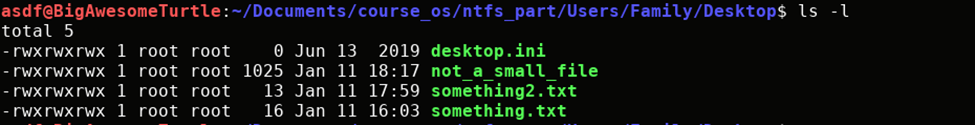
Для сравнения двух файлов используем утилиту diff:



Рис.3.3. Сравнение $Bitmap до и после создания файла

Утилита diff ничего не вывела, из чего следует, что файлы идентичны. Всё дело в том, что размер созданного файла весьма мал и он хранится прямо в MFT, поэтому новых кластеров резервировать не понадобилось.

Создадим файл not\_a\_small\_file, размером 1025 байт:



Заново извлечём $Bitmap и сравним результаты, используя утилиту vbindiff:

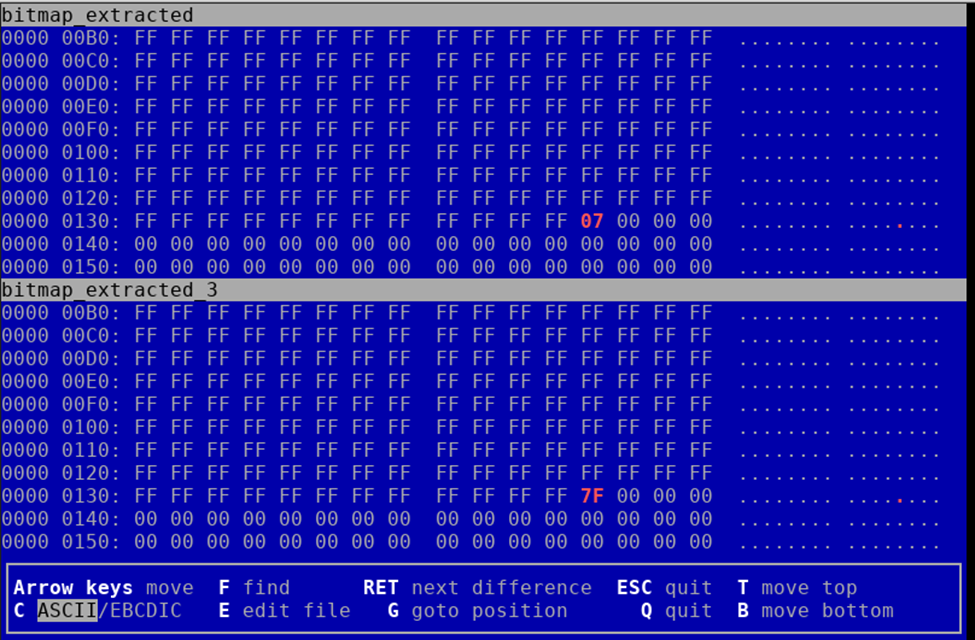


Рис.3.4. Сравнение $Bitmap до и после создания файла размером больше 1024 байт

Файл не поместился в $Mft, и ему было выделено несколько кластеров, которые стали занятыми в $Bitmap.

Удалим файл not\_a\_small\_file и снова сравним $Bitmap:

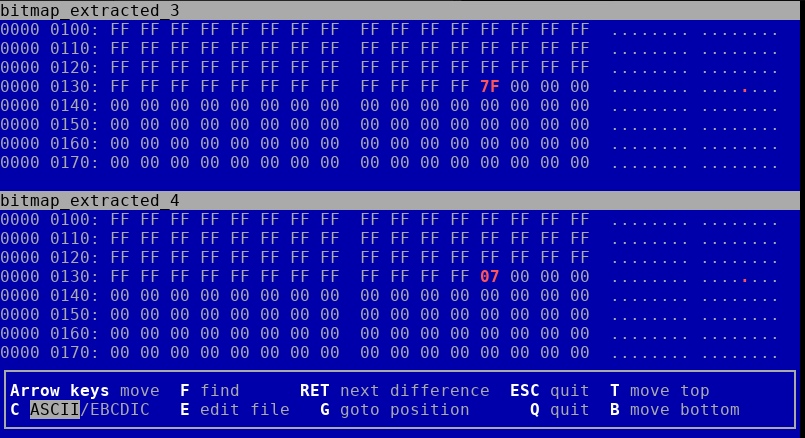


Рис.3.5. Сравнение $Bitmap до и после удаления файла

При удалении файла кластеры, занятые им, помечаются как свободные.

Работа с каталогами осуществляется аналогичным образом. Кластеры выделяются, если информация каталога не помещается в $Mft.

# **Заключение**

В любой файловой системе решаются задачи размещения файлов, доступа к файлам и управления свободным пространством. Для решения данных задач существуют различные подходы и механизмы.

В ходе выполнения данной курсовой работы были изучены различные методы решения приведённых задач, показаны плюсы и минусы тех или иных подходов, а также подробно рассмотрены методы, используемые в файловой системе NTFS и их реализация в рамках данной файловой системы. Кроме того, работа методов была продемонстрирована на практике с использованием драйвера NTFS-3G.

# **Список использованных источников**

1. Richard Russon, Yuval Fledel, " NTFS Documentation", 2004
2. Sibsankar Haldar, " Operating Systems", 2016
3. Jesse Russel, " NTFS-3G", 2012
4. Jesse Russel, " NTFS", 2012
5. Дайлип Наик, " Серверные технологии хранения данных в среде Windows 2000 Windows Server 2003", 2005
6. Dr. Philip Polstra, " Windows Forensics", 2016.
7. Brian Carrier, " File System Forensic Analysis", 2005
8. Dominic Giampaolo, " Practical File System Design", 1998
9. Jason Medeiros, " NTFS Forensics: A Programmers View of Raw Filesystem Data Extraction", 2008
10. Microsoft Press, "Ресурсы Microsoft Windows NT Server 4.0.", 2000
11. Rajeev Nagar, "Windows NT File System Internals: A Developer's Guide", 1997
12. Ayman Shaaban, Konstantin Sapronov, " Practical Windows Forensics", 2016
13. Файловая система NTFS https://www.ixbt.com/storage/ntfs.html

[Электронный ресурс] Дата обращения: 21.01.2020

1. NTFS Documentation. https://flatcap.org/linux-ntfs/ntfs/index.html

[Электронный ресурс] Дата обращения: 21.01.2020

1. Специальные файлы на разделе с NTFS. http://hex.pp.ua/ntfs-special-files.php, [Электронный ресурс] Дата обращения: 21.01.2020
2. Логическая и физическая организация файла. https://studopedia.su/1\_5766\_logicheskaya-i-fizicheskaya-organizatsiya-fayla.html, [Электронный ресурс] Дата обращения: 21.01.2020
3. Системы файлов.

https://www.intuit.ru/studies/courses/641/497/lecture/11302?page=2

[Электронный ресурс] Дата обращения: 21.01.2020

1. Open source NTFS-3G driver software and community,

https://www.tuxera.com/community/open-source-ntfs-3g/

[Электронный ресурс] Дата обращения: 21.01.2020

1. NTFS parser, plus linking capabilites between MFT LogFile and UsnJrnl. https://github.com/NTFSparse/ntfs\_parse

[Электронный ресурс] Дата обращения: 21.01.2020

1. Логическая организация ФС. https://studfile.net/preview/1175395/page:13/

[Электронный ресурс] Дата обращения: 21.01.2020