ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Р.Е.Алексеева»

Институт Радиоэлектроники и Информационных Технологий

Кафедра "Вычислительные системы и технологии"

ОТЧЁТ

По курсовой работе

«Утилита восстановления удаленных файлов»

по дисциплине «Принципы и методы организации системных программных средств»

Выполнил:

Докукин Д.В.

Проверил:

Кочешков А.А.

Нижний Новгород

Содержание:

* Теория
* Ограничения
* Ход работы
* Результаты
* листинг

Задание:

"Утилита восстановления удаленных файлов"

Разработать программу восстановления удаленных файлов в файловой системе FAT. Анализировать возможность восстановления конкретного файла и выдать на экран сообщение о его свойствах.

Пояснение к работе:

При выполнении данного задания, я решил сосредоточиться на файловой системе FAT16, она наиболее проста для изучения, доступна к пониманию и разбору в условиях ограниченного времени. И хотя существуют также разновидности данной файловой системы - FAT12, FAT32, VFAT, exFAT - но, в широком понимании, под файловой системой FAT подразумевается именно FAT16.

Теория:

Файловая система (ФС) FAT16 имеет довольно простую структуру:

* Загрузочная область
* Таблицы FAT
* Корневой каталог
* Область данных

Файловая система делится на сектора, стандартом является размер сектора 512 байт. В области данных формируются кластеры, по 16Кб или 32 сектора, 32Кб или 64 сектора и 64Кб или 128 секторов – размер кластеров определяется ёмкостью накопителя.

Данные, содержащиеся в файловой системе, принято интерпретировать особым образом:

Каждый байт представлен в виде двух шестнадцатиричных символов, эти символы условно делятся рядами и разбиваются на сектора. Чтобы ориентироваться по разделам структуры данных, введено понимание смещения – количество байт относительно начала ФС. Все данные в FAT16 воспринимаются, как шестнадцатиричные элементы. При этом, если определённый элемент данных в структуре ФС занимает больше одного байта, его первый байт является младшим, а каждый последующий принимается за старший, по возрастанию слева направо. Так, запись:

«02 A0 3E» - будет читаться не 2A03E в шестнадцатеричной системе счисления, а как 3EA002.

Загрузочный сектор

В загрузочном секторе находится служебная информация. Ниже приведу таблицу наиболее исчерпывающей информации:

|  |  |  |
| --- | --- | --- |
| Смещение | размер | описание |
| 0x0D | 1 | количество секторов в кластере |
| 0x0E | 2 | количество зарезервированных секторов |
| 0x10 | 1 | количество таблиц FAT |
| 0x11 | 2 | размер корневой директории |
| 0x16 | 2 | количество секторов для одной таблицы FAT |
| 0x36 | 8 | Тип используемой ФС |
| 0x1FE | 2 | конец загрузочного сектора с кодом 0x55AA |

Таблица FAT

Чтобы начать пояснение таблицы FAT, вначале следует объяснить принцип хранения данных в файловой системе. Область данных разделена на кластеры с фиксированной величиной, но довольно часто пользовательские файлы и данные занимают намного больше места, чем мог бы вместить в себя один кластер. Тогда, данные файла будут записываться в другие свободные кластеры, следующие за первым. Информация о первом кластере и размере файла фиксируется в корневом каталоге, а цепочка кластеров формируется в таблице FAT.

При использовании накопителя с ФС FAT, таблица выгружается в оперативную память и любые изменения, добавление/удаление данных будут зафиксированы. В случае повреждения таблицы, в памяти устройства выделяется область под копию таблицы. Количество таблиц указывается в загрузочном секторе.

Корневой каталог

Внутри корневого каталога все элементы разбиты по 32 байта. Первый элемент – это метка тома. Каждый последующий – описывает хранящиеся файлы и папки. Первые 8 байт – имя файла, следующие 3 – тип данных. 2 байта по смещению 0x1A указывают на первый кластер, куда был записан файл. Последние 4 байта – размер файла.

Первый байт элемента также служит идентификатором – например, код 0xE5 говорит, что файл удалён, а код 0x00 говорит, что предыдущий элемент был последним.

Область данных

Формально, корневой каталог также расположен в области данных, хоть ему и выделена строгая область, он занимает один кластер – ввиду чего, имеет ограничение на 512 записей. Также ограничения касаются общего количества кластеров, которое не может превышать 65536, а размер одного файла не может быть больше 2-х гигабайт.

Удаление файлов

Для ФС FAT16 предусмотрен стандартный алгоритм удаления файлов – в корневом каталоге первый байт имени заменяется на код 0xE5, в таблице FAT зануляется цепочка кластеров.

От теории к практике

Если система не фрагментирована, файлы довольно легко восстановить. Для этого, требуется перейти в корневой каталог, вести поиск по коду 0xE5 и фильтровать поиск по наличию первого кластера и размеру файла – если первый кластер не указан, а размер нулевой – значит, данный элемент нужно пропустить. При нахождении подходящих элементов, восстановление следует начинать с последнего элемента, поскольку шанс его успешного восстановления выше ввиду того, что вероятность перезаписи кластеров такого файла ниже.

По размеру файла определяется количество кластеров, необходимое для размещения файла. Далее переходим к таблице FAT. Находим соответствующие байты первому кластеру записи файла и восстанавливаем цепочку данных в свободных ячейках таблицы, опираясь на размер файла.

Такие же действия проделываем для других удалённых файлов.

Этот метод, однако, не гарантирует 100% успеха восстановления данных, но, при случайном их удалении, он позволяет их сразу весьма легко восстановить.

Ограничения

Для создания и испытания утилиты, было принято ограничиться некоторыми вводными:

- Используется только файловая система FAT16.

- Размер кластера ограничен 16 Килобайтами.

- все действия проводились на флеш карте объёмом 1Гб.

- Программа была написана средствами языка Си и его стандартных библиотек.

- Редактирование файловой системы происходит не непосредственно на накопителе, а на его образе.

Ход работы

Утилита восстановления данных мною создавалась в дистрибутиве GNU/Linux – Ubuntu - и опирается на возможности, реализованные в моей системе. Поскольку мне не удалось найти информации, как обратиться к байтам непосредственно на самом накопителе, я применял утилиту к образу накопителя. Образ накопителя создаётся одной строкой в терминале, а также большим разнообразием программ и утилит. Чтобы создать образ, я использовал команду:

$ dd if=<путь к системному файлу устройства> of=<путь для образа>

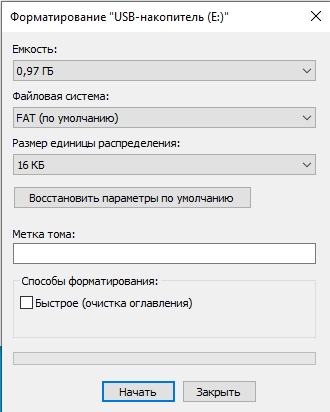
Для восстановления устройства из образа, нужно расположить наоборот в команде путь к файлу устройства и путь к образу.

Метод работы программы действует следующим образом:

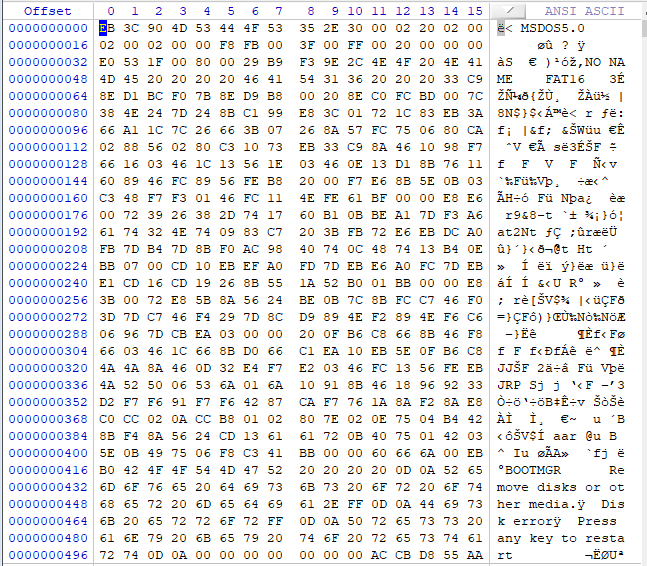
При открытии образа устройства с помощью функций fopen() и fread(), программа сверяет имя файловой системы с эталонным, затем берёт необходимые данные из загрузочного сектора, такие, как количество секторов на кластер, на ФАТ таблицу, зарезервированные секторы и пр. Далее, создаётся массив, в который загружаются байты данных из области ФАТ таблиц. Точно так же происходит с корневой директорией. Начинается проверка на наличие удалённых файлов – происходит сверка кода удалённого файла, ссылки на первый кластер и размера файла. Если программа выделяет удалённые файлы, то переходит к процессу восстановления. Восстановление происходит с конца – поскольку, чем ранее файл был удалён – тем ниже вероятность его восстановления, а также ниже вероятность пересечения кластеров при фрагментации. В функцию restoreFAT() передаются указатель на массив с байтами таблицы ФАТ, размер таблицы, адрес первого кластера удалённого файла и размер удалённого файла. Внутри функции, идёт проверка ячеек таблицы ФАТ, последовательно после начальной ячейки. Если ячейка свободна – она добавляется в последовательность кластеров, если же нет – проверяется следующий кластер. Чтобы избежать зацикленной проверки, введено ограничение – когда проверка достигает начального адреса, не восстановив полную цепочку – это будет значить, что восстановление прошло неудачно. Тогда функция завершит свою работу с не нулевым кодом, что станет сигналом не сохранять изменения в таблице ФАТ и продолжить восстановление других файлов. Если восстановление цепочки кластеров завершилось – далее следует проверка на размерность файла. Функция bytesAmountOf() проходится по восстановленной последовательности кластеров в таблице ФАТ, скопированной в байтовый массив, и проверяет данные из области данных – на выход функция отправляет вычисленное значение. Затем, программа сверяет, сходится ли вычисленный размер с указанным в элементе корневого каталога. Если информация сходится программа сообщит об успешном восстановлении файла и изменит код в первом байте элемента на код нижнего подчёркивания, если нет – сообщит о причине его неудачного восстановления. После восстановления всех файлов, с помощью функции fwrite() данные записываются из массивов в файл и программа завершает свою работу.

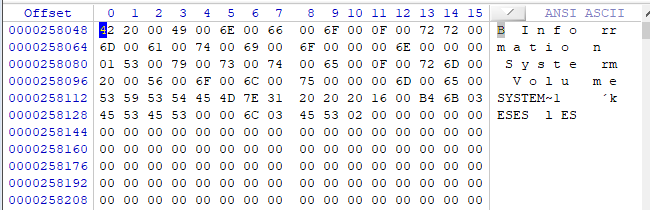
Результаты/Демонстрация

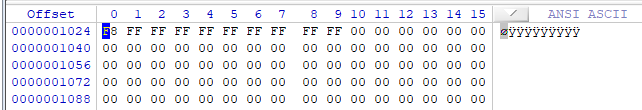
Вначале, я отформатировал флеш карту под требуемые характеристики.



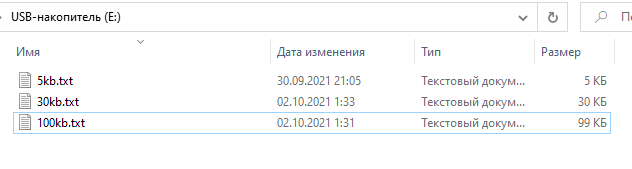
После форматирования, изучил структуру и данные – посмотрел загрузочный сектор и убедился, что установлена ФАТ16. В корневом каталоге и таблицах ФАТ отсутствуют данные о каких-либо установленных объектах.



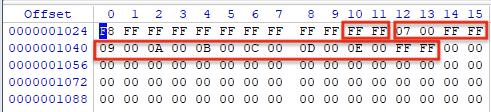


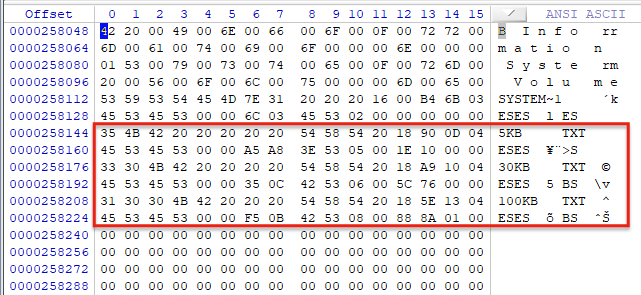


Далее, установлю файлы небольшого объёма, чтобы наглядно рассмотреть заполнение данными директорий файловой системы.



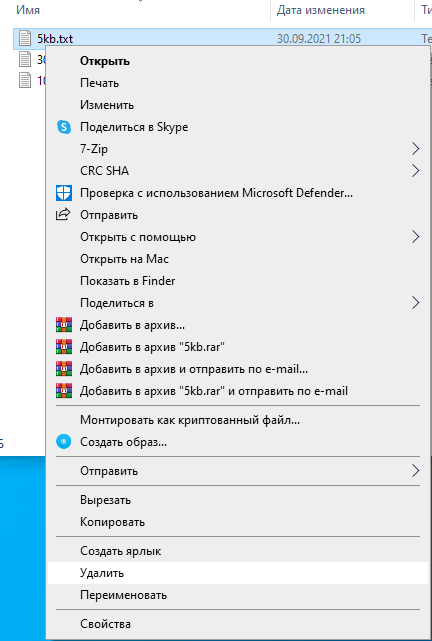
Теперь в таблице ФАТ и корневом каталоге можно наблюдать обновления информации.



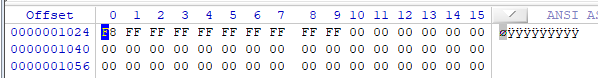


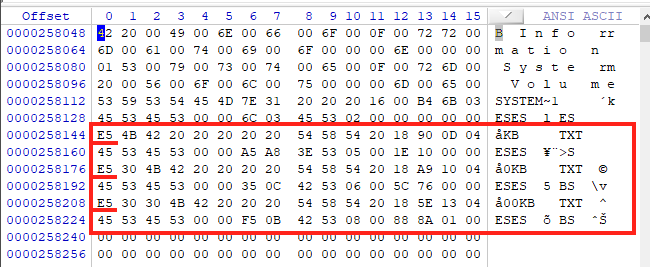
Можно увидеть, что первый файл занимает один кластер, второй файл – два, а третий - 7.

Перейдём к удалению.



Воспользуюсь стандартными средствами Windows и удалю файлы через выпадающее меню.

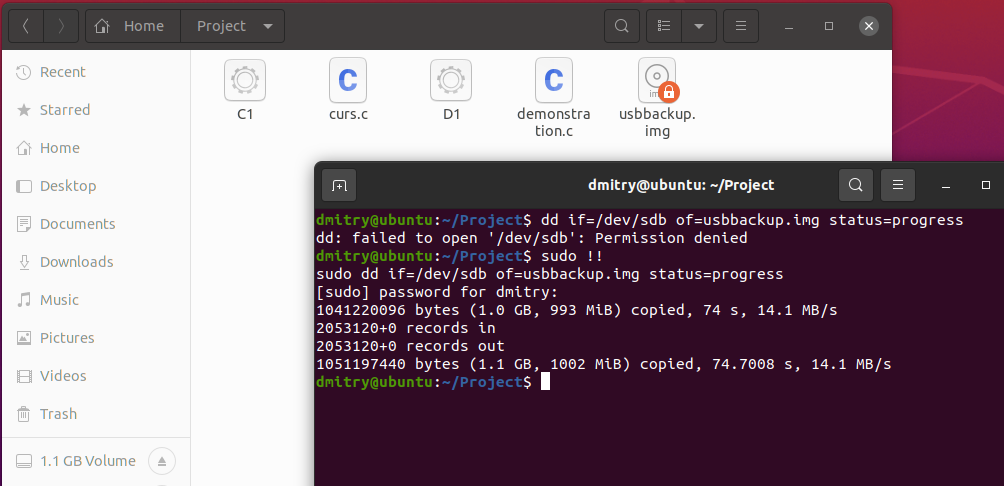




Наблюдаем в таблице ФАТ и корневом каталоге изменения – из таблицы пропали кластерные связи, а в корневом каталоге у элементов в первом байте имени появился код E5 – код, сообщающий, что файл был удалён.

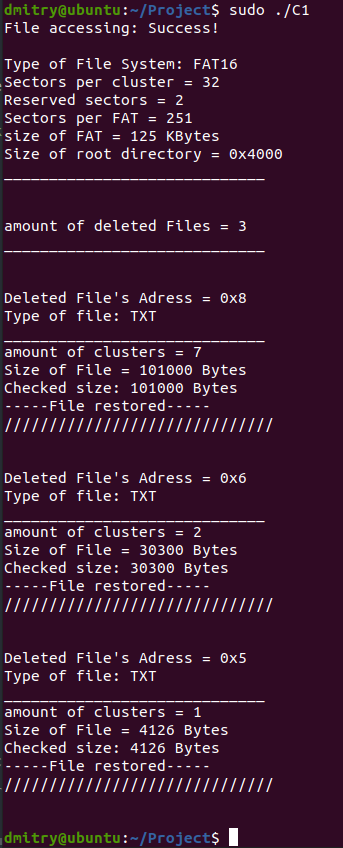
Теперь самое время перейти в Linux, чтобы попробовать восстановить эти данные.

Зайдём в терминал и воспользуемся командой “dd” для создания образа флеш карты.



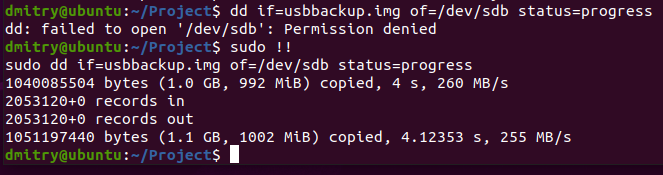
После создания образа, запускаю созданную мной утилиту.

В неё также была добавлена базовая отчётность, чтобы мониторить результаты.

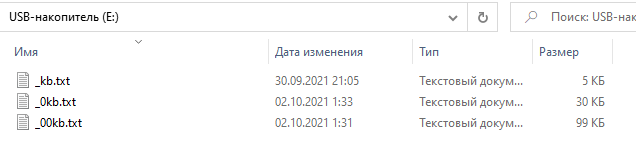


По выполнении, программа сообщает, что было обнаружено 3 удалённых файла, которые успешно удалось восстановить.

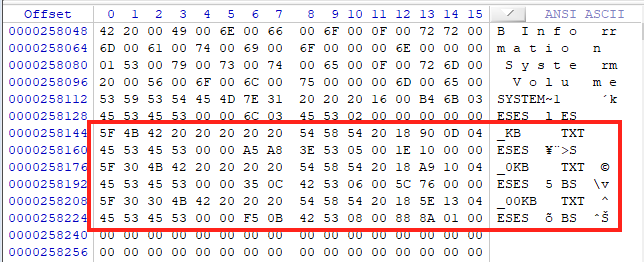
Восстановим флеш карту из отредактированного образа.

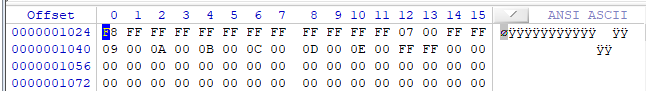


Зайдём в Windows, чтобы внимательнее разглядеть итоги.

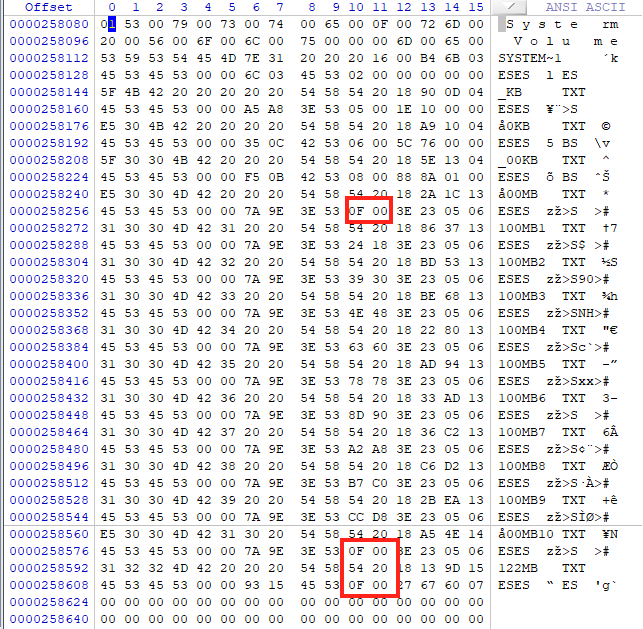


Как можно увидеть, файлы восстановились, но с учётом потерянной первой буквы имени – это связано с тем, что код E5 занимает первый байт имени файлов, вследствие чего восстановить первую букву имени в FAT16 на базовом уровне не представляется возможным.





Последовательности кластеров были идеально восстановлены.



Затем, были попытки создать искусственную фрагментацию, устанавливая и удаляя однотипные, но объёмные по размерам файлы. Увы, в процессе восстановления данных, было расхождение веса удалённых файлов и вычисленными значениями, а также наблюдались трудности в работе программы с тяжеловесными файлами, как следствие, восстановление завершалось неудачно.

Итог: в процессе изучения Файловой системы FAT и разработки методов восстановления информации, я узнал, насколько проста и дружелюбна к сторонним вмешательствам, но в то же время, насколько система FAT16 отстаёт в плане возможностей, реализуемых современными файловыми системами. Также, было крайне трудно понять, как получить доступ к скрытым разделам. Российские ресурсы не способны предложить решения по данному вопросу. Лишь путём проб и ошибок, мне удалось самому понять и разобраться, как можно получить доступ с скрытым байтам собственной флешки. Язык Си и среда Линукса оказались крайне полезны в решении этого вопроса. И совсем не лишней оказалась третья лабораторная работа по Операционным Системам, где мне приходилось разбирать свойства Unix-подобных систем на примере Ubuntu, именно в ней я выполнял свою курсовую работу.

Ниже представлен листинг программы с краткими комментариями к функциям и определённым частям кода.

Листинг:

#include "stdio.h"

#include "stdlib.h"

#include "limits.h"

#include <string.h>

// функция отображения данных секторами, по смещению - (file, amount of sectors, offset)

**void** showBytesOf(FILE \*file, **unsigned** **int** sectors, **unsigned** **int** offset) {

**unsigned** **char** a = 0; // буфер для байта

**short** **int** flag = -7; // флаг для красивого вывода

**int** i = 0, j = 0; // счётчики

fseek(file, offset, SEEK\_SET); // возвращаем указатель в файле на начало

// здесь реализован красивый вывод

**for** (i = 0; i < sectors \* 0x20; i++) { // смещение по оси Y

**for** (j = 0, flag = -7; j < 0x10; j++, flag++) { // смещение по оси X

fread(&a, 1, 1, file); // Читаем байты из файла

**if** (a >= 0x0 && a <= 0xF) { // Рисуем нолик для красоты, если байт меньше 16

printf ("0%X ", a);

} **else** {

printf("%X ", a);

}

**if** (flag == 0) printf("%s", " "); // пропуск после каждого восьмого байта

}

printf("%s", "\n"); // переход на новую строку каждые 16 байт

**if** ((i+1) % 0x20 == 0) printf("%s", "\n"); // делаем пропуск после конца каждого сектора

}

fseek(file, 0, SEEK\_SET); // возвращаем указатель в файле на начало

}

// функция рисует таблицу байт в 16-ричной записи из массива сырых данных, взятых из файла

// развер таблицы задается байтами

**void** paintTable(**unsigned** **char** \*array, **unsigned** **int** size) {

**unsigned** **int** i = 0;

**for** (i = 0; i < size; i++) {

**if** (array[i] < 0x10) {

printf("0%X ", array[i]);

} **else** {

printf("%X ", array[i]);

}

**if** ((i+1) % 8 == 0) printf(" ");

**if** ((i+1) % 16 == 0) printf("\n");

**if** (i > 510) {

printf("\n");

**break**;

}

}

}

// функция поиска и вывода числа длиной от 1 до 4 байт из файла

// в действительнсоти, не актуальна

**unsigned** **int** get4BytesFromF(FILE \*file, **unsigned** **int** offset, **int** bytes) {

**unsigned** **int** buf = 0, num = 0; // буфер для байта

**switch** (bytes) {

**case** 4:

fseek(file, offset+3, SEEK\_SET);

fread(&buf, 1, 1, file);

num += buf \* 0x1000000;

**case** 3:

fseek(file, offset+2, SEEK\_SET);

fread(&buf, 1, 1, file);

num += buf \* 0x10000;

**case** 2:

fseek(file, offset+1, SEEK\_SET);

fread(&buf, 1, 1, file);

num += buf \* 0x100;

**case** 1:

fseek(file, offset, SEEK\_SET);

fread(&buf, 1, 1, file);

fseek(file, 0, SEEK\_SET); // возвращаем указатель в файле на начало

num += buf;

**break**;

**default**:

**break**;

}

**return** num;

}

// функция переводит запись байтов из массива от 1 до 4 байт в числовое выражение

// актуальна при сохранении сырых данных из файла в массив

**unsigned** **int** get4BytesFromAr(**unsigned** **char** \* arr, **unsigned** **int** ofst, **int** bytes) {

**unsigned** **int** num = 0;

**switch** (bytes) {

**case** 4:

num += arr[ofst + 3] \* 0x1000000;

**case** 3:

num += arr[ofst + 2] \* 0x10000;

**case** 2:

num += arr[ofst + 1] \* 0x100;

**case** 1:

num += arr[ofst];

**break**;

**default**:

**break**;

}

**return** num;

}

// функция заносит определенный отрезок данных из файла в массив

**void** putInArray(FILE \*file, **char** \*arr, **unsigned** **int** size, **unsigned** **int** offset) {

**unsigned** **int** i = 0;

**unsigned** **char** a;

fseek(file, offset, SEEK\_SET); // возвращаем указатель в файле на начало

**for** (i = 0; i < size; i++) {

fread(&a, 1, 1, file);

\*(arr + i) = a;

}

fseek(file, 0, SEEK\_SET); // возвращаем указатель в файле на начало

}

// makes string with type of file system info

**void** getTypeofFS(FILE \*file, **char** \*typeOfFS, **int** size, **unsigned** **int** offset) {

**unsigned** **char** a = 0;

**int** i = 0;

**for** (i = 0; i < size-1; i++) {

fseek(file, offset + i, SEEK\_SET);

fread(&a, 1, 1, file);

fseek(file, 0, SEEK\_SET); // возвращаем указатель в файле на начало

\*(typeOfFS + i) = a;

}

\*(typeOfFS + size-1) = '\0';

}

**void** getTypeofFile(**unsigned** **char** \*arr, **unsigned** **char** \*typeOfFile, **int** size, **unsigned** **int** offset) {

**unsigned** **char** a = 0;

**int** i = 0;

**for** (i = 0; i < size-1; i++) {

a = \*(arr + offset + i);

\*(typeOfFile + i) = a;

}

\*(typeOfFile + size-1) = '\0';

}

**void** copyFAT(**unsigned** **char** \*from, **unsigned** **char** \*to, **unsigned** **int** size) {

**unsigned** **int** i = 0;

**for** (i = 0; i < size; i++) {

\*(to + i) = \*(from + i);

}

}

// Основная функция, восстанавливающая цепочку кластеров в таблице ФАТ

**int** restoreFAT(**unsigned** **char** \* fat, **unsigned** **int** sizeOfFAT, **unsigned** **int** adress, **unsigned** **int** fileSize) {

// Если по адресу уже что-то находится - завершить работу функции и вернуть 1

**if** (fat[adress\*2] != 0 || fat[adress\*2+1] != 0) **return** 1;

// Вернуть 2, если файл ничего не весит

**if** (fileSize == 0) **return** 2;

// получаем расчетное количество кластеров, занимаемых файлов

**unsigned** **int** amount\_of\_clusters = fileSize / 16384;

**if** (fileSize % 16384 != 0) amount\_of\_clusters++;

printf("amount of clusters = %d\n", amount\_of\_clusters);

**unsigned** **int** flag = adress;

**unsigned** **int** j = adress;

**unsigned** **int** i = 0;

// Если файл занимает больше одного кластера - выстраиваем цепочку

**for** (i = 1; i < amount\_of\_clusters; i++) {

j++;

**if** (j == sizeOfFAT) j = 0;

**if** (j == adress) **return** 3;

**if** (fat[j\*2] == 0 && fat[j\*2 + 1] == 0) {

fat[flag\*2] = j%0x100;

fat[flag\*2 + 1] = j/0x100;

flag = j;

}

}

printf("marker2\n");

// Последнюю ячейку помечаем конечным кодом

fat[j\*2] = 0xFF;

fat[j\*2+1] = 0xFF;

**return** 0;

}

// Функция проверяет кластеры в области данных в соответствии с цепочкой адресов в таблице ФАТ

// чтобы определить количество байт данных

**unsigned** **int** bytesAmountOf(**unsigned** **int** start\_adress, **unsigned** **char** \*fat, **unsigned** **int** offset, FILE \*file) {

**unsigned** **int** i = 0;

**unsigned** **int** amount = 0;

**unsigned** **int** adress = start\_adress;

**unsigned** **char** a = 0;

**do** {

// перемещаем указатель в конец первого кластера файла

// если байт не нулевой - засчитываем 16 килобайт к весу файла и движемся к след кластеру

fseek(file, offset + 0x4000 \* (adress - 1) + 0x3FFF, SEEK\_SET);

fread(&a, 1, 1, file);

**if** (a != 0 && adress != 0xFFFF) {

amount += 0x4000;

adress = get4BytesFromAr(fat, adress \* 2, 2);

**continue**;

}

// Если последний байт кластера не нулевой - считаем ненулевые байты

**for** (i = 0; i < 0x4000; i++) {

fseek(file, offset + i + 0x4000 \* (adress - 1), SEEK\_SET);

fread(&a, 1, 1, file);

**if** (a == 0) **break**;

amount++;

}

adress = get4BytesFromAr(fat, adress \* 2, 2);

} **while** (adress != 0xFFFF);

fseek(file, 0, SEEK\_SET);

**return** amount;

}

// MAIN FUNCTION

**int** main(**void**) {

FILE \*file;

printf("Enter file's name: ");

**unsigned** **char** nameOfFile[20];

scanf("%s", nameOfFile);

file = fopen(nameOfFile, "rb+");

// проверяем, открылся ли файл

printf("File accessing: ");

**if** (file == **NULL**) {

printf("Error.\n");

exit(2);

} **else** {

printf("%s\n", "Success!\n");

}

**int** i = 0, j = 0;

// 4 байта на переменную - достаточно для работы с фат16

**unsigned** **int** size\_of\_sector = 512; // задаем по умолчанию bytes

**unsigned** **int** bootSector\_Offset = 0x4000; // ЗС имеет смещение в 16кб

**unsigned** **int** typeOfFS\_offset = 0x36; // 8 bytes

**unsigned** **char** typeOfFS[9]; // 8 bytes

getTypeofFS(file, typeOfFS, 9, bootSector\_Offset + typeOfFS\_offset);

**if** (typeOfFS[0] != 'F' || typeOfFS[1] != 'A' || typeOfFS[2] != 'T' || typeOfFS[3] != '1' || typeOfFS[4] != '6') {

printf("Wrong File System or size of Cluster. Should be FAT16 with 16Kb cluster size.\n");

**return** 0;

}

**unsigned** **int** secPerClustr\_offset = 0x0D; // 1 byte

**unsigned** **int** reservedSec\_offset = 0x0E; // 2 bytes

**unsigned** **int** secPerFAT\_offset = 0x16; // 2 bytes

**unsigned** **int** sizeofRootDir\_offset = 0x11; // 2 bytes

**unsigned** **int** sectorsPerCluster = get4BytesFromF(file, bootSector\_Offset + secPerClustr\_offset, 1);

**unsigned** **int** reservedSectors = 0;

fseek(file, bootSector\_Offset + reservedSec\_offset, SEEK\_SET);

fread(&reservedSectors, 1, 2, file);

**unsigned** **int** sectors\_per\_FAT = get4BytesFromF(file, bootSector\_Offset + secPerFAT\_offset, 2);

**unsigned** **int** size\_of\_FAT = sectors\_per\_FAT \* size\_of\_sector;

**unsigned** **int** size\_of\_RootDirectory = get4BytesFromF(file, bootSector\_Offset + sizeofRootDir\_offset, 2) \* 32;

printf("Type of File System: %s\n", typeOfFS);

printf("Sectors per cluster = %d\n", sectorsPerCluster); // выводим кол-во секторов в кластере

printf("Reserved sectors = %d\n", reservedSectors);

printf("Sectors per FAT = %d\n", sectors\_per\_FAT); // выводим кол-во секторов в FAT1/2

printf("size of FAT = %d KBytes\n", size\_of\_FAT/1024);

printf("Size of root directory = 0x%X\n", size\_of\_RootDirectory);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n");

// помещаем таблицу ФАТ1 в массив для последующий взаимодействий

**unsigned** **int** FAT1\_offset = bootSector\_Offset + reservedSectors \* size\_of\_sector;

**unsigned** **int** FAT2\_offset = FAT1\_offset + size\_of\_FAT;

**unsigned** **char** FAT1[size\_of\_FAT];

putInArray(file, FAT1, size\_of\_FAT, FAT1\_offset);

**unsigned** **char** FAT2[size\_of\_FAT];

copyFAT(FAT1, FAT2, size\_of\_FAT);

// помещаем кластер с корневой директорией целиком в массив - поскольку на него выделена область,

// следующая сразу после таблиц ФАТ, то не нужно выяснять, в каких кластерах директория располагается

**unsigned** **char** RootDir[size\_of\_RootDirectory];

**unsigned** **int** dataSegment\_offset = FAT2\_offset + size\_of\_FAT;

**unsigned** **int** rD\_offset = dataSegment\_offset;

putInArray(file, RootDir, size\_of\_RootDirectory, rD\_offset);

// Восстановление данных происходит ниже

**unsigned** **int** amount\_of\_delFiles = 0;

**unsigned** **int** flag = 0; // флаг конца "списка элементов" корневого каталога

**for** (i = 0; i < size\_of\_RootDirectory; i = i + 32) {

**if** (RootDir[i] == 0xE5 && (RootDir[i + 0x1A] != 0 || RootDir[i + 0x1B] != 0)) {

amount\_of\_delFiles++;

}

**if** (RootDir[i] == 0) {

flag = i; // получает адрес нулевого байта (код последнего эл) в корневом каталоге

**break**;

}

}

// Завершить программу, если нету удалённых элементов

**if** (amount\_of\_delFiles == 0) {

printf("nothing to restore.\n");

**return** 0;

}

printf("amount of deleted Files = %d\n", amount\_of\_delFiles);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n");

**unsigned** **int** size\_test = 0;

**unsigned** **int** delFile\_Adress;

**unsigned** **int** size\_of\_delFile;

**unsigned** **char** type\_of\_file[4];

**for** (i = flag - 32; i >= 0 ; i = i - 32) { // -32

**if** (RootDir[i] == 0xE5 && get4BytesFromAr(RootDir, i + 0x1A, 2) != 0) {

delFile\_Adress = get4BytesFromAr(RootDir, i + 0x1A, 2);

size\_of\_delFile = get4BytesFromAr(RootDir, i + 0x1C, 4);

printf("Deleted File's Adress = 0x%X\n", delFile\_Adress);

getTypeofFile(RootDir, type\_of\_file, 4, i + 0x8);

printf("Type of file: %s\n", type\_of\_file);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

**if** (restoreFAT(FAT2, size\_of\_FAT, delFile\_Adress, size\_of\_delFile) != 0) {

printf("Error. Impossible to restore the file.\n");

printf("The sequense of clusters cannot be restored.\n");

printf("//////////////////////////////\n\n\n");

copyFAT(FAT1, FAT2, size\_of\_FAT);

**continue**;

}

size\_test = bytesAmountOf(delFile\_Adress, FAT2, dataSegment\_offset, file);

**if** (size\_test != size\_of\_delFile) {

printf("Error. different sizes: must be - %d Bytes, checked - %d Bytes\n", size\_of\_delFile, size\_test);

printf("//////////////////////////////\n\n\n");

copyFAT(FAT1, FAT2, size\_of\_FAT);

**continue**;

}

printf("Size of File = %d Bytes\n", size\_of\_delFile);

printf("Checked size: %d Bytes\n", size\_test);

copyFAT(FAT2, FAT1, size\_of\_FAT);

RootDir[i] = 0x5F;

printf("-----File restored-----\n");

printf("//////////////////////////////\n\n\n");

}

}

// запись результатов в файл

fseek(file, FAT1\_offset, SEEK\_SET);

fwrite(FAT1, 1, size\_of\_FAT, file);

fseek(file, FAT2\_offset, SEEK\_SET);

fwrite(FAT1, 1, size\_of\_FAT, file);

fseek(file, rD\_offset, SEEK\_SET);

fwrite(RootDir, 1, size\_of\_RootDirectory, file);

fclose(file);

**return** 0;

}