МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

факультет КНТ

Кафедра ПМІ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до курсової роботи з дисципліни

"Системне програмування"

Тема роботи: «Пошук об'єктів, довжина яких не менше зазначеної кількості кластерів»

Керівники: Виконав: каф. КІ ст. гр. ІПО-13 Шевченко О. А. Лисенко А. С.

\_\_.\_\_. 2016р. \_\_.\_\_. 2016р.

Покровськ 2016

Затверджую

Зав. кафеди КІ\_\_\_\_\_\_\_

Святний В.А

\_\_\_\_\_\_\_ 20 року.

# ТЕХНІЧНЕ ЗАВДАННЯ

**до курсового проекту з дисципліни «Системне програмування»**

Тема роботи: Розробка програмного забезпечення файлового менеджера.

Студенту групи ІПЗ-13

Термін здачі роботи на перевірку: згідно графіку виконання.

Варіант - №4.

***Мета*** - дослідження схем сегментації пристроїв зовнішньої пам’яті та фізичних моделей файлових систем ОС Windows.

***Практичний результат*** - розробка програмного забезпечення файлового менеджера (ФМ), що реалізує наступні функції:

* пошук і найменування **всіх** логічних дисків у межах даної конфігурації технічних засобів;
* визначення характеристик логічних дисків;
* навігація по дереву файлової системи;
* Створення директорії

***Вимоги*** до виконання:

* 1. Програмне забезпечення ФМ розробляється як графічний WIN -додаток.
  2. Мова програмування – будь-яка мова класу С, інструментальне середовище – згідно обраної мови
  3. Кожна з зазначених вище функцій ФМ повинна ініціюватися тільки за вимогою користувача.
  4. Вихідний інтерфейс розробляється в традиційному стилі відомих файлових менеджерів (FAR, WC, провідник, тощо).
  5. Після запуску ФМ на екрані відображається «список» об'єктів файлової системи (ФС) кореневого каталогу завантажувального диску.
  6. ФМ підтримує логічні диски з файловою системою FAT32, NTFS.
  7. Доступ до об'єктів файлової системи FAT32 виконується тільки через аналіз службової інформації диску.
  8. Доступ до об'єктів файлової системи NTFS виконується з використанням засобів мови програмування. Індивідуальне. завдання для диску з NTFS- не реалізується.
  9. Імена логічних дисків повинні збігатися з іменами відповідних дисків у рамках ОС, де функціонує ФМ.
  10. Ідентифікація типів об'єктів ФС у вихідному інтерфейсі виконується або по графічному відображенню їхніх імен (колір, шрифт, іконки, тощо), або явною вказівкою типу (файл, директорія).
  11. Повнота інформації про об'єкт ФС визначається індивідуальним завданням і повинна бути достатньою для перевірки його виконання.
  12. Характеристики логічних дисків відображаються в одному з наступних форматів: діаграма.
  13. У вихідному інтерфейсі зобов'язаний відображатися поточний шлях дерева ФС.
  14. Обов'язкова підтримка довгих імен і кирилиці.
  15. Довге ім'я об'єкта повинно бути представлене по всій довжині.
  16. У вихідному інтерфейсі інформація індивідуального завдання і характеристики диску не повинна затирати список об'єктів ФС поточної директорії.
  17. У ФМ повинні бути передбачені індикатори виконання тривалих за часом операцій.
  18. Колірна палітра вхідного/вихідного інтерфейсу повинна бути витримана в спокійних тонах.

***Примітки***.

1. Робота ФМ повинна бути передбачувана.
2. Керуючі компоненти вихідного інтерфейсу повинні бути інформативні, повідомлення про можливі помилки - лаконічними

***Зміст пояснювальної записки:***

Титульний лист.

Технічне завдання.

Реферат.

Вступ.

1. Структура програмного забезпечення файлового менеджеру.
2. Структура даних.
3. Опис алгоритмів ПЗ ФМ.
4. Опис програмних модулів.
5. Методика роботи.
6. Дослідження результатів.

Висновок.

Список використаної літератури.

Додатки.

***Склад графічної частини***

1. Структура програмного забезпечення (А2).
2. Схеми-алгоритмів (A2).
3. Плакат з вихідними результатами.

***Графік виконання***

1. Розробка структури програмного забезпечення (1-тиждень)
2. Розробка алгоритмів інтерфейсу файлового менеджера. (1-тиждень)
3. Програмування та автономне налагодження вхідного та

вихідного інтерфейсу. (2-тижні )

1. Розробка алгоритмів доступу до об'єктів файлової системи. (1-тиждень)
2. Програмування та автономне налагодження модулів доступу (2-тижні)
3. Комплексне налагодження програмного забезпечення (3-тижні)
4. Оформлення пояснювальної записки та графічної частини (1-тиждень)
5. Захист курсової роботи.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оглавление

[ТЕХНІЧНЕ ЗАВДАННЯ 2](#_Toc474795388)

[ВСТУП 5](#_Toc474795389)

[1 СТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ФАЙЛОВОГО МЕНЕДЖЕРУ 5](#_Toc474795390)

[2 СТРУКТУРА ДАНИХ 6](#_Toc474795391)

[2.1 Структури завантажувального запису 7](#_Toc474795392)

[2.2 Структура службової областіFAT 8](#_Toc474795393)

[2.3 Структура елемента каталогу 11](#_Toc474795394)

[3 ОПИС АЛГОРИТМІВ ПЗ ФМ 13](#_Toc474795395)

[3.1 Алгоритм пошуку дисків й іменування дисків 13](#_Toc474795396)

[3.2 Алгоритм доступу к об’єктам файлової системи 15](#_Toc474795397)

[3.3 Алгоритм пошук об'єктів, довжина яких не менше зазначеної кількості кластерів 17](#_Toc474795398)

[3.4 Алгоритм визначення зайнятого місця на розділі 18](#_Toc474795399)

[3.5 Алгоритм зрівняння директорій 20](#_Toc474795400)

[ВИСНОВОК 20](#_Toc474795401)

[ДОДАТОК А 21](#_Toc474795402)

[Програмна реалізація 21](#_Toc474795403)

[ДОДАТОК Б 40](#_Toc474795404)

[Скріншоти програми 40](#_Toc474795405)

Реферат

# ВСТУП

Метою даного проекту є практичне дослідження та засвоєння прийомів роботи з дисковими накопичувачами у середовищі ОС Windows та роботи на низькому рівні з файловими системами FAT16/FAT32. Також підтримується робота з NTFS. Програма розроблена як WINDOWS програма, написана мовою С++. Інтерфейс програми був розроблений схожий до існуючих файлових менеджерів середовища Microsoft Windows.

# 1 СТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ФАЙЛОВОГО МЕНЕДЖЕРУ

Файловий менеджер – програма, яка виконує візуалізацію вмісту каталогу, дозволяє виконувати різноманітні операції з об’єктами каталогу, наприклад, копіювання, видалення, правка та інші.

Файловий менеджер, який був розроблений згідно з завданням до проекту виконує наступні функції:

- пошук і найменування логічних дисків у межах даної конфігурації технічних засобів;

- визначення характеристик логічних дисків;

Також була реалізована підтримка довгих імен та кирилиці для об’єктів директорій.

Структура файлового менеджеру на рис. 1.1.

Згідно зі структурою файловий менеджер має інтерфейс, в якому можна обрати наступну дію.

Список дій показаний в структурі ФМ.

Програма має багатомодульну структуру. Кожний модуль виконує свої функції. Деякі дії, які можна виконати, реалізована в окремі функції. Виключення із загального правил являє собою модуль Source.cpp, який реалізує як інтерфейс так і роботу з NTFS та порівняння директорії. Тому на структурній схемі проекту він зустрічається двічі.

При старті програми спочатку створюється інтерфейс користувача, виконується пошук логічних дисків в межах даної конфігурації технічних засобів, відбувається найменування усіх знайдених дисків.

Після цього обирається завантажувальний диск та зчитується кореневий каталог цього диску. Вміст каталогу показується на екрані. Далі програма очікує наступних вказівок користувача щодо подальшої діяльності.



Рисунок 1.1 – Структура ФМ

# 2 СТРУКТУРА ДАНИХ

У програмі використовуються декілька структур даних. Структури завантажувального запису, службової частини ФС та елементу каталогу – це системні структури. Також програма містить і власні структури – інформація про логічні диски та інші.

## 2.1 Структури завантажувального запису

Інформація про розділи жорсткого диску зберігається у першому секторі пристрою. Це – головний завантажувальний запис MBR (Master Boot Record). Структура MBR наведена у табл. 2.1.

Таблиця 2.1 - Структура MBR

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 1BE h | Код завантажника MBR | reserved |
| 1BE h | 40 h | Масив з 4х елементів Partition Table | Partition Table |
| 1FE h | 2 | Сигнатура MBR (0х55АА) | sign |

Один елемент Partition Table може визначати логічний диск або розширений розділ. У межах одного жорсткого диска може бути лише один розширений розділ. Заповнення полів Partition Table виконується на етапі розбивання диска на розділи. В табл. 2.2 наведена структура елементу Partition Table.

Таблиця 2.2 Структура елемента Partition Table

|  |  |  |
| --- | --- | --- |
| Зсув | Довжина(байт) | Опис |
| 00h | 1 | Ознака активності розділу |
| 01h | 1 | Початок розділу – голівка |
| 02h | 1 | Початок розділу - сектор (біти 0-5), доріжка (біти 6,7) |
| 03h | 1 | Початок розділу - доріжка |
| 04h | 1 | Код типу розділу |
| 05h | 1 | Кінець розділу – голівка |
| 06h | 1 | Кінець розділу - сектор (біти 0-5), доріжка (біти 6,7) |
| 07h | 1 | Кінець розділу - доріжка |
| 08h | 4 | Відносний номер першого сектору розділу (startSector) |
| 0Ch | 4 | Кількість секторів розділу |

Поле код типу розділу містить інформацію про тип ФС (основний розділ) або про ознаку розширеного розділу (05h, 0Fh).

Основний розділ описує логічний диск, адресу початку якого можна взяти зі структури елементу Partition Table – поле startSector. Це ж поле у елементі Partition Table при ознаці розширеного розділу вказує на вторинну MBR. Ця MBR може містити максимум два елементи Partition Table з чотирьох. Перший елемент буде вказувати на черговий логічний диск, а другий – на наступну вторинну MBR. Для отримання абсолютної адреси початку логічного диска необхідно до значення поля startSector додати адресу MBR, у якій описується даний диск.

## 2.2 Структура службової областіFAT

Після отримання абсолютної адреси початку логічного диска в програмі виконується зчитування першого сектора диска. В системі FAT це – завантажувальна область (BOOT – область). BOOT – область містить параметри та характеристики логічного диска. Її структура для ФС FAT12 та FAT16 наведена у табл. 2.3, а для системи FAT32 – у табл. 2.4.

Таблиця 2.3 – Структура BOOT – сектору для FAT12 та FAT16

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 3 | Команда JMP на код завантажника | jmpcode |
| 3 | 8 | Назва операційної системи, у якій виконано форматування диску | os |
| 11 | 2 | Кількість байт у секторі | BytePerSector |
| 13 | 1 | Кількість секторів у кластері | SectorPerCluster |
| 14 | 2 | Кількість резервних секторів | SizeReserv |
| 16 | 1 | Кількість копій FAT | NumberCopiesFAT |
| 17 | 2 | Кількість елементів кореневого каталогу | MaxDirElem |
| 19 | 2 | Розмір диску в секторах для дисків <32MB, інакше 0 | Smallsize |
| 21 | 1 | Описувач середовища | MediaDescriptor |
| 22 | 2 | Кількість секторів таблиці FAT | SizeFAT16inSectors |
| 24 | 2 | Секторів на доріжці | SectorPerTrack |
| 26 | 2 | Кількість голівок | Heads |
| 28 | 4 | Кількість схованих секторів | NumberHiddenSectors |
| 32 | 4 | Розмір в секторах для дисків > 32MB | BigSize |
| 36 | 1 | Тип пристрою(для першого диску в системі 80h, для інших 0) | -- |
| 37 | 1 | Резерв | -- |
| 38 | 1 | Сигнатура 29h. | Code |
| 39 | 4 | Серійний номер | SerialNumber |
| 43 | 11 | Метка диску | Label |
| 54 | 8 | Ідентифікатор FAT (‘FAT12’ або ‘FAT16’) | FATID |
| 62 | 2 | Код завантажника | -- |

Таблиця 2.4 – Структура BOOT – сектору для FAT32

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 3 | Команда JMP на код завантажника | jmpcode |
| 3 | 8 | Назва операційної системи, у якій виконано форматування диску | os |
| 11 | 2 | Кількість байт у секторі | BytePerSector |
| 13 | 1 | Кількість секторів у кластері | SectorPerCluster |
| 14 | 2 | Кількість резервних секторів | SizeReserv |
| 16 | 1 | Кількість копій FAT | NumberCopiesFAT |
| 17 | 4 | Резерв | --- |
| 21 | 1 | Описувач середовища | MediaDescriptor |
| 22 | 2 | Резерв | --- |
| 24 | 2 | Секторів на доріжці | SectorPerTrack |
| 26 | 2 | Кількість голівок | Heads |
| 28 | 4 | Кількість схованих секторів | NumberHiddenSectors |
| 32 | 4 | Резерв | --- |
| 38 | 6 | Резерв | reserv1 |
| 44 | 4 | Початковий кластер кореневого каталогу | StartCluster |
| 48 | 2 | Початковий сектор структури FS INFO | BegFS |
| 50 | 2 | Номер сектору з копією BOOT-розділу | BootCopy |
| 52 | 12 | Резерв | reserv2 |
| 64 | 1 | Фізичний номер пристрою | PhysNum |
| 65 | 1 | Резерв | reserv3 |
| 66 | 1 | Розширена сігнатура | ExtSign |
| 67 | 4 | Серійний номер пристрою | SerialNumber |
| 71 | 11 | Метка диску | Label |
| 82 | 8 | Ідентифікатор (‘FAT32’) | FATID |
| 90 | 2 | 55AA | --- |

Дана структура дозволяє отримати доступ до інформаційних полів BOOT-сектора необхідного логічного диска.

BOOT-область в файлових системах FAT12,16 займає 1 сектор, а в ФС FAT32 – 3 сектори. Другий сектор містить додаткові параметри та сигнатури, а третій – продовження програми завантаження. За завантажувальною областю розташовані таблиці FAT – таблиці кластерів. Їх кількість визначається у BOOT-секторі. У файлових системах FAT12,16 за таблицями кластерів знаходиться кореневий каталог. Його розмір обмежений кількістю елементів, вказаних в BOOT-секторі. Кореневий каталог FAT32 може не розміщатися відразу ж за таблицями кластерів та не має меж щодо свого розміру. За всіма цими службовими областями знаходиться область даних. Таблиця FAT містить інформацію про розподілення дискового простору під об’єкти ФС. Ця таблиця – масив елементів із розмірністю 12, 16 або 32 біти в залежності від версії ФС. Номер елементу таблиці FAT відповідає номеру кластера в області даних. У таблиці наведені можливі значення одного елементу FAT.

Таблиця 2.5 – Значення елементу FAT

|  |  |  |  |
| --- | --- | --- | --- |
| FAT12 | FAT16 | FAT32 | Пояснення |
| 0 | 0 | 0 | Вільний кластер |
| FF0-FF6 | FFF0-FFF6 | 0FFFFFF0-0FFFFFF6 | Зарезервований кластер |
| FF7 | FFF7 | 0FFFFFF7 | BAD-кластер |
| FF8-FFF | FFF8-FFFF | 0FFFFFF8-0FFFFFFF | Останній кластер об’єкту |

Усі інші значення вказують на наступний кластер.

Послідовність кластерів, яка може належати одному об’єкту в таблиці FАТ, представляє собою односпрямований список, голова якого в явному виді відсутня, а кінець визначається ознакою кінця ланцюжка.

## 2.3 Структура елемента каталогу

Кожен каталог представляє собою послідовність дескрипторів. Структура дескриптора об’єкта з коротким ім’ям наведена у табл. 2.6 для FAT12/FAT16 та у табл. 2.8 для FAT32. Структура байту атрибуту у табл. 2.7.

Таблиця 2.6 – Структура дескриптора для FAT12 / FAT16

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 1 | Ознака дескриптору: 0 – вільний; E5h - видалений; інше-перший символ імені об’єкту | fn |
| 1 | 7 | 7 символів імені об’єкту | name |
| 8 | 3 | Розширення об’єкту | ext |
| 11 | 1 | Байт атрибутів | attr |
| 12 | 10 | Резерв | reserv |
| 22 | 2 | Час створення або останньої модифікації | TimeMade |
| 24 | 2 | Дата створення або останньої модифікації | DateMade |
| 26 | 2 | Молодша частина початкового кластеру об’єкта | FirstCluster |
| 28 | 4 | Розмір об’єкта в байтах | SizeFileInBytes |

Таблиця 2.7 – Байт атрибутів об’єкта

|  |  |  |
| --- | --- | --- |
| Номер біту | Значення біту | Опис |
| 0 | 1 | Об’єкт тільки для читання |
| 1 | 1 | Об’єкт схованого типу |
| 2 | 1 | Об’єкт системного типу |
| 3 | 1 | Мітка тому |
| 4 | 1 | Директорія |
| 5 | 1 | Архівний файл |
| 6 | Не використовується | |

Таблиця 2.8 – Структура дескриптора для FAT32

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 1 | Ознака дескриптору: 0 – вільний; E5h - видалений; інше-перший символ імені об’єкту | fn |
| 1 | 7 | 7 символів імені об’єкту | name |
| 8 | 3 | Розширення об’єкту | ext |
| 11 | 1 | Байт атрибутів | attr |
| 12 | 1 | Резерв | reserv |
| 13 | 2 | Час створення (0.1 секунд) | TimeMadeSec |
| 14 | 2 | Час створення | TimeMade |
| 16 | 2 | Дата створення або останньої модифікації | DateMade |
| 18 | 2 | Дата останнього звертання | DateLast |
| 20 | 2 | Старший байт номеру першого кластеру, який був виділений об’єкту | FirstClusterHigh |
| 22 | 2 | Час останньої модифікації об’єкту | TimeLast |
| 24 | 2 | Дата останнього запису об’єкту | DateLastWrite |
| 26 | 2 | Молодший байт номеру першого кластеру, який був виділений об’єкту | FirstClusterLow |
| 28 | 4 | Розмір файлу в байтах | SizeFileInBytes |

Якщо об’єкт іменується довгим ім’ям, то під нього виділяється декілька дескрипторів стандартного розміру (32б). Кількість дескрипторів визначається довжиною імені об’єкта. Максимальна довжина імені об’єкта – 255 символів, які зберігаються в форматі UNICODE (по два байти на один символ). У кожному дескриптору може зберігатися 13 символів імені об’єкту. Структура дескриптора для довгого імені наведена у табл. 2.9.

Таблиця 2.9 – Структура дескриптора для довгого імені

|  |  |  |  |
| --- | --- | --- | --- |
| Зсув | Розмір, байт | Опис | Ім’я |
| 0 | 1 | Номер порції імені | fn |
| 1 | 10 | 5 символів імені об’єкту | FiveSymb |
| 11 | 1 | Байт атрибутів, дорівнює 0Fh | attr |
| 12 | 1 | Завжди дорівнює 0 | reserv |
| 13 | 1 | Контрольна сума короткого ім’я | CRC |
| 14 | 12 | 6 символів імені об’єкту | SixSymb |
| 26 | 2 | Резерв | reserv2 |
| 28 | 4 | 2 символи імені об’єкту | TwoSymb |

Ім’я в останній порції довгого імені може бути меншим за 13 символів. У такому випадку значима частина імені завершується нулем. усі інші поля імені заповнюються FFFF.

# 3 ОПИС АЛГОРИТМІВ ПЗ ФМ

У цьому пункті розглядаються послідовно алгоритми пошуку та іменування дисків, доступу до об’єктів файлової системи, визначення зайнятого місця для файлової системи FAT32, FAT16.

## 3.1 Алгоритм пошуку дисків й іменування дисків

Алгоритм іменування логічних дисків засновано на звіренні серійного номера, отриманого логічного диска із серійним номером, збереженим системою.



Рисунок 3.1 – Пошук та найменування дисків

## 3.2 Алгоритм доступу к об’єктам файлової системи

Основна концепція файлової системи FAT полягає в тім, що кожному файлу й каталогу виділяється структура даних, називана дескриптором. У цій структурі зберігається ім'я файлу, його розмір, початкова адреса вмісту файлу й інші метадані. Данні файлів і каталогів зберігається в блоках даних, називаних кластерами. Якщо файлу або каталогу виділяється більш одного кластера, інші кластери знаходять за допомогою структури даних, називаної FAT(File Allocation Table). Структура FAT використовується як для ідентифікації наступних кластерів у файлах, так і для визначення стану кластерів. Існує три версії FAT: FAT12, FAT16 і FAT32. Вони відрізняються друг від друга насамперед розміром запису у структурі FAT. Зв'язки між структурами даних показано на рис. 3.4.



Рисунок 3.4 – Зв’язки між структурами даних

Файлова система FAT ділиться на три фізичні області дляFAT32, та на чотири для FAT12/16.Перша область називається зарезервованою; в FAT12 і FAT16 зарезервована область займає всього 1 сектор, але формально її розмір визначається в завантажувальному секторі. Друга область FAT - містить основні й резервні структури FAT. Вона починається в секторі, котрий розташовано за зарезервованою областю, а її розмір визначається кількістю й розміром структур FAT. Третя – кореневий каталог, для FAT12/16 починається за областю FAT, а у FAT32 має повільне положення у області даних. Область даних - містить кластери, виділені для зберігання файлів і вмісту каталогів.

Доступ до файлових об’єктів виконується з припущення, що відома адреса першого кластеру об’єкту.

У даній реалізації алгоритм доступу до об’єктів містить дві частини – алгоритм пошуку шляху до поточної директорії та алгоритм пошуку об’єктів у завантаженій директорії.

Алгоритм пошук об'єктів, довжина яких не менше зазначеної кількості кластерів на рис. 3.2 Алгоритм пошуку поточного шляху- рис.3.3

## 3.3 Алгоритм пошук об'єктів, довжина яких не менше зазначеної кількості кластерів

Алгоритм пошук об'єктів, довжина яких не менше зазначеної кількості кластерів на рис. 3.2



Алгоритм пошуку поточного шляху- рис.3.3

## 3.4 Алгоритм визначення зайнятого місця на розділі

Визначення зайнятого місця на розділі реалізується шляхом аналізу таблиці FAT. Виконується перевірка усіх елементів таблиці FAT. Рахується кількість елементів, що містять 0. Ці елементи в файловій системі ідентифікують незайняте місце.

Отже, після повного перегляду FAT таблиці відома кількість елементів FAT таблиці та кількість елементів незайнятого місця. Знаходиться відсоткове співвідношення. Через нього обчислюється зайняте місце в байтах.

Алгоритм визначення на рис. 3.4.



Рисунок 3.4 – Алгоритм визначення зайнятого місця

## 3.5 Алгоритм зрівняння директорій

 Рисунок 3.5 – Алгоритм зрівняння директорій

# ВИСНОВОК

У ході виконання курсового проекту була створена програма для ОС Windows. Також були покращені навички роботи з накопичувачем на жорсткому магнітному диску. Був розібраний низький рівень існування інформації на жорсткому диску.

Так як основна увага приділялася роботі з ФС FAT, були здобуті вичерпні знання про структуру цієї ФС та навички роботи з нею на низькому рівні.

# ДОДАТОК А

## Програмна реалізація

Source.cpp

#include<stdio.h>

#include<iostream>

#include<conio.h>

#include<Windows.h>

using namespace std;

BYTE mbr[512];

BYTE ebr[512];

BYTE mbr1[512];

BYTE mbr2[512];

DWORD dwRead;

DISK\_GEOMETRY dg;

HANDLE hDevice;

DWORD ReturnSize;

DWORD DiskSize;

HANDLE hDevice1;

\_\_int64 z = 0x0FFFFFFFF;

int i = 0;

int d = 0;

int a = 0;

int a1 = 0;

int a2 = 0;

\_\_int64 ladd = 0;

\_\_int64 badd = 0;

\_\_int64 madd = 0;

int z1 = 0x048;

//int l = \*(int\*)&mbr[0] >> z1;

\_\_int64 l = 0;

int y = 0;

int y2 = 0;

int b = 0;

int e = 0;

int f = 0;

int k = 0;

DWORD serC = 0;

DWORD serD = 0;

DWORD serH = 0;

DWORD serG = 0;

DWORD serF = 0;

DWORD serI = 0;

void showEbr(int a)

{

i = 0;

ladd = a; //

a1 = a;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

//printf("%d\n", a);

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &ebr, 512, &dwRead, NULL);

int d = 0;

//int a = 0;

int b = 0;

while (d < 2)

{

//cout << hex << (int)ebr[446 + i] << endl;

e = (unsigned char)ebr[446 + i];

/\*if (e == 128)

cout << "Загрузочный" << endl;

else

cout << "Не загрузочный" << endl;\*/

//cout << hex << (int)ebr[446 + 4 + i] << endl;

f = (unsigned char)ebr[446 + 4 + i];

if (f == 7 || f == 17)

{

//cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

ladd = y; // add func

y += a;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[z1];

//serH = l;

cout << "Серийный номер: " << dec << l << endl;

}

else if (f == 11 || f == 12)

{

y = \*(int\*)&ebr[446 + 8 + i];

//cout << "Основной раздел" << endl;

cout << "ФС : FAT32\n";

ladd = y; // add func

y += a;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[67];

serH = l;

cout << "Серийный номер: " << dec << l << endl;

}

else if (f == 1 || f == 4 || f == 13 || f == 14 || f == 20 || f == 22 || f == 23 || f == 27)

{

cout << "Основной раздел" << endl;

cout << "ФС : ?\n";

}

else if (f == 5 || f == 15)

cout << "Расширенный раздел" << endl;

else if (f == 6)

{

cout << "ФС : FAT16" << endl;

y = \*(int\*)&ebr[446 + 8 + i];

ladd = y; // add func

y += a;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[39];

//serH = l;

cout << "Серийный номер: " << dec << l << endl;

}

else

cout << "ФС не опознана" << endl;

y = \*(int\*)&ebr[446 + 8 + i];

//a = a / 2 / 1024;

printf("Адрес начала раздела: %d в секторах\n", y);

b = \*(int\*)&ebr[446 + 12 + i];

b = b / 2 / 1024;

printf("Размер диска: %d Мб\n", b);

cout << endl;

i += 16;

d++;

}

cout << endl;

a += y;

ladd = a; //

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &ebr, 512, &dwRead, NULL);

i = 0;

d = 0;

while (d < 2)

{

//cout << hex << (int)ebr[446 + i] << endl;

e = (int)ebr[446 + i];

if (e == 128)

cout << "Загрузочный" << endl;

else

cout << "Не загрузочный" << endl;

//cout << hex << (int)ebr[446 + 4 + i] << endl;

f = (int)ebr[446 + 4 + i];

if (f == 7 || f == 17)

{

//cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

}

else if (f == 11 || f == 12)

{

//cout << "Основной раздел" << endl;

cout << "ФС : FAT32\n";

ladd = y; // add func

y += a;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

//serG = l;

l = \*(DWORD\*)&mbr1[67];

}

else if (f == 1 || f == 4 || f == 13 || f == 14 || f == 20 || f == 22 || f == 23 || f == 27)

{

cout << "Основной раздел" << endl;

cout << "ФС : ?\n";

}

else if (f == 5 || f == 15)

cout << "Расширенный раздел" << endl;

else if (f == 6)

{

y = \*(int\*)&ebr[446 + 8 + i];

cout << "ФС : FAT16" << endl;

y += a;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[39];

serG = l;

cout << "Серийный номер: " << dec << l << endl;

}

else

cout << "ФС не опознана" << endl;

y = \*(int\*)&ebr[446 + 8 + i];

//a = a / 2 / 1024;

printf("Адрес начала раздела: %d в секторах\n", y);

b = \*(int\*)&ebr[446 + 12 + i];

b = b / 2 / 1024;

printf("Размер диска: %d Мб\n", b);

cout << endl;

i += 16;

d++;

}

cout << endl;

a1 += y;

ladd = a1; //

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &ebr, 512, &dwRead, NULL);

i = 0;

d = 0;

while (d < 2)

{

//cout << hex << (int)ebr[446 + i] << endl;

e = (int)ebr[446 + i];

if (e == 128)

cout << "Загрузочный" << endl;

else

cout << "Не загрузочный" << endl;

//cout << hex << (int)ebr[446 + 4 + i] << endl;

f = (int)ebr[446 + 4 + i];

if (f == 7 || f == 17)

{

y = \*(int\*)&ebr[446 + 8 + i];

//cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

ladd = y; // add func

y += a1;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[z1];

serF = l;

cout << "Серийный номер: " << dec << l << endl;

}

else if (f == 11 || f == 12)

{

//cout << "Основной раздел" << endl;

cout << "ФС : FAT32\n";

ladd = y; // add func

y += a1;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[67];

}

else if (f == 1 || f == 4 || f == 13 || f == 14 || f == 20 || f == 22 || f == 23 || f == 27)

{

cout << "Основной раздел" << endl;

cout << "ФС : ?\n";

}

else if (f == 5 || f == 15)

cout << "Расширенный раздел" << endl;

else if (f == 6)

{

y = \*(int\*)&ebr[446 + 8 + i];

cout << "ФС : FAT16" << endl;

y += a1;

ladd = y;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[39];

cout << "Серийный номер: " << dec << l << endl;

}

else

cout << "ФС не опознана" << endl;

y = \*(int\*)&ebr[446 + 8 + i];

//a = a / 2 / 1024;

printf("Адрес начала раздела: %d в секторах\n", y);

b = \*(int\*)&ebr[446 + 12 + i];

b = b / 2 / 1024;

printf("Размер диска: %d Мб\n", b);

cout << endl;

i += 16;

d++;

}

cout << endl;

k++;

}

void showDisk()

{

int i = 0;

int d = 0;

int b = 0;

int e = 0;

int f = 0;

int a = 0;

char s[20] = "\\\\.\\PhysicalDrive0";

hDevice = CreateFile(s, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_WRITE | FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, 0, NULL);

if (hDevice == INVALID\_HANDLE\_VALUE)

printf("Невозможно получить хэндл - ого жесткого диска.\n");

else

{

dg.MediaType = FixedMedia;

DeviceIoControl(hDevice, IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY, NULL, 0,

&dg, sizeof(DISK\_GEOMETRY), &ReturnSize, (LPOVERLAPPED)NULL);

printf("Цилиндров : %d\n", dg.Cylinders);

printf("Дорожек на цилиндр : %d\n", dg.TracksPerCylinder);

printf("Секторов на дорожку : %d\n", dg.SectorsPerTrack);

printf("Размер сектора : %d байт\n", dg.BytesPerSector);

DiskSize = (dg.Cylinders.LowPart \* dg.TracksPerCylinder \*

dg.SectorsPerTrack) / 1024 \* dg.BytesPerSector;

printf("Размер HDD : %d Mb\n", DiskSize / 1024);

cout << endl;

if (!ReadFile(hDevice, &mbr, 512, &dwRead, NULL))

{

cout << "Ошибка не может прочитать MBR" << endl;

CloseHandle(hDevice);

}

while (d < 4)

{

cout << endl;

e = (unsigned char)mbr[446 + i];

//printf("Код загрузочного раздела: %X\n", e);

if (e == 128)

cout << "Загрузочный" << endl;

else

cout << "Не загрузочный" << endl;

f = (unsigned char)mbr[446 + 4 + i];

//printf("Код типа раздела: %X\n", f);

if (f == 7 || f == 17)

{

//unsigned char

cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

a = \*(int\*)&mbr[446 + 8 + i];

printf("Адрес начала раздела: %d в секторах\n", a);

b = \*(int\*)&mbr[446 + 12 + i];

b = b / 2 / 1024;

printf("Размер диска: %d Мб\n", b);

ladd = a; //

//if(k1 == 0)

a1 = a;

//else if (k1 > 0)

a1 = +a;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice, &mbr, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr[z1];

serC = l;

cout << "Серийный номер: " << dec << l << endl;

a = \*(int\*)&mbr[446 + 8 + i];

}

else if (f == 11 || f == 12)

{

cout << "Основной раздел" << endl;

cout << "ФС : FAT32\n";

}

else if (f == 1 || f == 4 || f == 6 || f == 13 || f == 14 || f == 20 || f == 22 || f == 23 || f == 27)

{

cout << "Основной раздел" << endl;

cout << "ФС : ?\n";

}

else if (f == 5 || f == 15)

cout << "Расширенный раздел" << endl;

else

{

cout << "ФС не опознана" << endl;

a = \*(int\*)&mbr[446 + 8 + i];

printf("Адрес начала раздела: %d в секторах\n", a);

b = \*(int\*)&mbr[446 + 12 + i];

b = b / 2 / 1024;

printf("Размер диска: %d Мб\n", b);

}

a1 += a;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice, &mbr, 512, &dwRead, NULL);

i += 16;

d++;

cout << endl;

}

cout << endl;

//}

i = 0;

d = 0;

b = 0;

e = 0;

f = 0;

char s1[20] = "\\\\.\\PhysicalDrive1";

hDevice1 = CreateFile(s1, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_WRITE | FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, 0, NULL);

if (hDevice1 == INVALID\_HANDLE\_VALUE)

printf("Невозможно получить хэндл - ого жесткого диска.\n");

else

{

dg.MediaType = FixedMedia;

DeviceIoControl(hDevice1, IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY, NULL, 0,

&dg, sizeof(DISK\_GEOMETRY), &ReturnSize, (LPOVERLAPPED)NULL);

//CloseHandle(hDevice);

printf("Цилиндров : %d\n", dg.Cylinders);

printf("Дорожек на цилиндр : %d\n", dg.TracksPerCylinder);

printf("Секторов на дорожку : %d\n", dg.SectorsPerTrack);

printf("Размер сектора : %d байт\n", dg.BytesPerSector);

DiskSize = (dg.Cylinders.LowPart \* dg.TracksPerCylinder \*

dg.SectorsPerTrack) / 1024 \* dg.BytesPerSector;

printf("Размер HDD : %d Mb\n", DiskSize / 1024);

cout << endl;

if (!ReadFile(hDevice1, &mbr, 512, &dwRead, NULL))

{

cout << "Ошибка не может прочитать MBR" << endl;

CloseHandle(hDevice1);

}

}

int p = 0;

a1 = 0;

//int k = 0;

while (d < 4)

{

a = \*(\_\_int64\*)&mbr[446 + 8 + i];

e = (unsigned char)mbr[446 + i];

//printf("Код загрузочного раздела: %X\n", e);

if (e == 128)

cout << "Загрузочный" << endl;

else

cout << "Не загрузочный" << endl;

f = (unsigned char)mbr[446 + 4 + i];

b = \*(int\*)&mbr[446 + 12 + i];

b = b / 2 / 1024;

if (f == 7 || f == 17)

{

if (k == 0)

{

cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

printf("Размер диска: %d Мб\n", b);

//if (k1 > 0)

printf("Адрес начала раздела: %d в секторах\n", a);

ladd = a; //

//a1 = a;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[z1];

serD = l;

cout << "Серийный номер: " << dec << l << endl;

}

else

{

cout << "Основной раздел" << endl;

cout << "ФС : NTFS\n";

printf("Размер диска: %d Мб\n", b);

//if (k1 > 0)

printf("Адрес начала раздела: %d в секторах\n", a);

ladd = a; //

//a1 = a;

ladd \*= 512;

badd = ladd & z;

ladd = ladd >> 32;

ladd = ladd & z;

SetFilePointer(hDevice1, (int)badd, (PLONG)&ladd, 0);

ReadFile(hDevice1, &mbr1, 512, &dwRead, NULL);

l = \*(DWORD\*)&mbr1[z1];

serI = l;

cout << "Серийный номер: " << dec << l << endl;

}

}

else if (f == 11 || f == 12)

{

cout << "Основной раздел" << endl;

cout << "ФС : FAT32\n";

}

else if (f == 1 || f == 4 || f == 13 || f == 14 || f == 20 || f == 22 || f == 23 || f == 27)

{

cout << "Основной раздел" << endl;

cout << "ФС : ?\n";

}

else if (f == 5 || f == 15)

{

cout << "Расширенный раздел" << endl;

showEbr(a);

}

else

{

cout << "Расширенный раздел" << endl;

}

//serI = l;

i += 16;

d++;

}

cout << endl;

}

}

int main()

{

setlocale(LC\_CTYPE, "Rus");

showDisk();

char VolumeNameBuffer[100];

char FileSystemNameBuffer[100];

DWORD VolumeSerialNumber;

BOOL GetVolumeInformationFlag = GetVolumeInformation(

"c:\\", (LPSTR)&VolumeNameBuffer,

100,

(LPDWORD)&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR) FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "c:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serC)

{

cout << "c:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

GetVolumeInformationFlag = GetVolumeInformation(

"d:\\", VolumeNameBuffer,

100,

(LPDWORD)&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR)FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "d:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serD)

{

cout << "d:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

GetVolumeInformationFlag = GetVolumeInformation(

"f:\\", VolumeNameBuffer,

100,

&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR)FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "f:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serF)

{

cout << "f:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

GetVolumeInformationFlag = GetVolumeInformation(

"g:\\", VolumeNameBuffer,

100,

(LPDWORD)&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR)FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "g:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serG)

{

cout << "g:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

GetVolumeInformationFlag = GetVolumeInformation(

"h:\\", VolumeNameBuffer,

100,

(LPDWORD)&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR)FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "h:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serH)

{

cout << "h:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

GetVolumeInformationFlag = GetVolumeInformation(

"i:\\", VolumeNameBuffer,

100,

(LPDWORD)&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

NULL, //&FileSystemFlags,

(LPTSTR)FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

//cout << "i:\\" << endl;

//cout << "Volume Name is " << &VolumeNameBuffer << endl;

cout << "Серийный номер: " << dec << VolumeSerialNumber << endl;

if (VolumeSerialNumber == serI)

{

cout << "i:\\" << endl;

}

cout << "Файловая система: " << FileSystemNameBuffer << endl;

cout << endl;

}

else cout << "Не представляет (GetVolumeInformation)" << endl;

\_getch();

}

# ДОДАТОК Б

## Скріншоти програми









