## Міністерство освіти і науки України

## ФАКУЛЬТЕТ ІНФОРМАТИКИ

КАФЕДРА

#### Реєстраційний №\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## КУРСОВА РОБОТА

**Тема:**

**Програма для роботи з файловою системою.**

#### Рекомендована до захисту

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2008р.

Робота захищена

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2008р.

з оцінкою

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

Підписи членів комісії

**Зміст**

# Вступ

# Теорія

# Практична частина

# Висновки

# Література

**Вступ**

Поставимо перед собою задачу створення програми для роботи з файловою системою (створення закритих ділянок файловою системи). Для створення програми скористуємося мовою програмування С++. В теоретичній частині роботи коротко охарактеризуємо головні особливості роботи з файловими системами FAT 16 (MS-DOS).

Ієрархічна (багаторівнева) файлова система MS-DOS схожа з файловою системою ОС XENIX. Опис багаторівневої каталогової системи й роботи з нею дано в MS-DOS в теоретичній частині.

**ТЕОРІЯ**

Файлова система є однієї з найбільших частин операційної системи. Файлова система є надбудовою над носієм даних блокового пристрою (звичайно це дисковід гнучких магнітних дисків або вінчестер), що перетворює структуру каталогу й файли у одиницю пам'яті. Файлова система на диску містить, як мінімум, інформацію про розміщення файлів, каталог й обсяг файлів. Інформація про розміщення файлів може приймати різні форми в залежності від операційної системи, але всі форми, в основному, стежать простір, використовуваний файлами й простір доступне для нових даних. Каталог містить список файлів, збережених на пристрої, їхні розміри й інформацію про розміщення цих файлів.

Існує кілька різних підходів при розміщенні файлів й елементів каталогу. MS-DOS використає специфічний метод розміщення, називаний таблицею розміщення файлів (FAT), і ієрархічну структуру каталогу.

Ступінь деталізації файлу, наявна в розпорядженні операційної системи, також сильно коливається залежно від реалізації. Деякі системи, такі як MS-DOS, мають файли, які доступні на уровні байта; інші системи обмежують цей доступ фіксованим розміром запису.

Файлові системи іноді розширюються до таблиці символьних пристроїв, тому що б вони були файлами. Ці "файли" пристроїв можуть відкриватися, закриватися, читатися, у них можна записувати інформацію як в звичайні дискові файли, але всі транзакції відбуваються безпосередньо з пристроєм, специфікованим символом. Файли пристрою забезпечують корисну погодженість середовища для прикладних програм. MS-DOS підтримує такі файли шляхом призначення резервного логічного імені (такі як CON або PRN) кожному символьному пристрою.

**Структура файлової системи**

Доступ до блокових пристроїв здійснюється по секторах. Ядро MS DOS за допомогою драйвера прибудую розглядає блоковий пристрій як логічний кінцевий масив секторів і припускає, що цей масив містить припустиму в MS-DOS файлову організацію. Драйвер прибудую, у свою чергу, перетворити запитий MS DOS налогічний сектор у фізичну адресі блокового прибудую.

Первісна файлова система MS DOS записана на згадку з програми MS DOS FORMAT (див. главу USER COMMANDS: FORMAT).

Сектор початкового завантаження завжди розташований на качану роздягнула. Він містить ідентифікацію ОЕМ , програма-завантажник і блок параметрів BIOS (BPB) з інформацією про пристрій; потім треба необов'язкова область зарезервованих секторів (див. нижче розділ СЕКТОР ПОЧАТКОВОЇ ЗАВАНТАЖЕННЯ). Зарезервована область не має спеціального призначення, однак ОЕМ може зажадати більше складної програми-завантажника й помістити її в цю область. Таблиці розміщення файлів (FAT) указують на розташування області файлів даних; кореневий каталог містить фіксовані число входів, а область файлів даних включає файли даних, файли підкаталогів і вільні сектори даних.

Всі описані вище області - сектор початкового завантаження, FAT, корневого каталог й область файлів даних - мають фіксований розмір; це означає, що смороду не змінюються після того, як програма FORMAT визначила середовище даних. Розмір кожної із цих областей залежить від різних факторів. Наприклад, розмір FAT пропорційний області файлів даних. Розмір кореневого каталогу звичайно залежить від типу пристроїв; однобічний гнучкий диск може мати 64 входу, двосторонній гнучкий диск - 1126, а фіксований диск - 256.(Драйвери віртуальних дисків такі, як RAMDRIVE.SYS і деякі реалізації програми FORMAT дозволяють специфицировать число входів каталогу). Область файлів даних описується в термінах кластерів. Кластер визначається як фіксоване число суміжних секторів. Розмір сектора і розмір кластера повинні бути рівні ступені 2. Розмір сектора звичайно становить 512 байт, а розмір кластера - 1, 2 або 4 Кбайт, однак можливі сектори й кластери більшого розміру. Нижче наведені типові розміри кластерів, використовуваних в MS DOS :

Загалом кажучи, кластери більшого розміру використаються для фіксованих дисків більшого розміру. Хоча кластери меншого розміру дають їхнє розташування просторово більше ефективним, кластери більшого розміру є звичайно більше ефективними для довільного й послідовного доступу, особливо якщо кластери одному файлу расположєні непослідовно.

Таблиця розміщення файлів містить по одному вході для шкірного кластера в області файлів даних. Дублювання секторів у кластері також зменшить удвічі число входів FAT для даного розділу

Сектор початкового завантаження (мал.3-6) містить блок параметрів BIOS (BPB), програма-завантажник і деякі інші дані корисні для драйверів пристроїв. BPB описує деяке число фізичних параметрів пристрою, а також розташування й розмір інших областей на пристрої. Драйвер пристрою, при відповідному запиті, передає MS DOS інформацію про BPB, так що MS DOS може визначити конфігурацію диска.

Інформація ВРВ, що втримується в байтах з ОВН по 17Н, указує, що:

сектор включає 512 байт,

у кластері два сектори;

є один зарезервований сектор (для сектора початкової

завантаження), 2 FAT'а, 112 входів у кореневий каталог;

диск містить 1440 секторів, F9H дескрипторів пристроїв;

FAT включає 3 сектора.

Наступна за BPB додаткова інформація вказує, що в доріжку втримується 9 секторів, є 2 головки читання/запису й 0 невидимих секторів.

Дескриптор носія даних, що вказує у ВРВ й у першому байті кожної FAT, використається для опису типу носія, що перебуває в даний момент на пристрої. Сумісні з IBM носії даних мають наступні дескриптори:

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

00H | E9 XX XX або EB XX 90 |

|------------------------------------------------------|

03H | Ім'я й версія ОЕМ (8 байт) |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_

0BH | Число байтів у секторі (2 байти) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

0DH | Число секторів на пристрої (1 байт) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

0EH | Число резервних секторів - нумерація з 0 (2 байти) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

10H | Число FAT (1 байт) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

11H | Число входів у кореневому каталозі (2 байти) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | BPB

13H | Загальне число секторів на логічному томі (2 байти) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

15H | Дескриптор носія даних (1 байт) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| | |

16H | Число секторів в FAT (2 байти) | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_|

| |

18H | Число секторів на доріжці (2 байти) |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| |

1AH | Число головок (2 байти) |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| |

1CH | Число невидимих секторів (2 байти) |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| |

1EH | |

| ПРОГРАМА-ЗАВАНТАЖНИК |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Структура сектора початкового завантаження MS DOS

Байти з ОВН по 17Н містять блок параметрів BIOS (BPB)

0 1 2 3 4 5 6 7 8 9 A B C D E F

0000 EB 2D 90 20 20 20 20 20-20 20 20 00 02 02 01 00 .................

0010 02 70 00 A0 05 F9 03 00-09 20 22 00 00 00 00 00 .................

0020 00 0A 00 00 DF 02 25 02-09 2A B8 50 F6 02 FA 00 .................

0030 BB C0 07 8E D8 BC 80 7C-33 C0 8E C0 F8 02 01 00 .................

. . .

. . .

. . .

0180 OA 44 69 73 5B 20 42 48-65 20 48 61 69 65 27 00 .Disk Boot.Failu.

0190 72 65 0D 0A 0D 0A 0A 4E-6F 20 53 79 73 74 65 60 re....Non-System.

01A0 20 64 69 73 6B 20 6F 72-20 64 69 73 6B 20 01 72 .disk or disk er

01B0 72 6F 72 0D 0A 52 65 70-60 61 63 65 20 61 62 64 .ror..Replace..and

01C0 20 70 72 65 73 73 20 61-6E 79 20 6B 65 79 20 77 .press.any key.w.

01D0 68 65 6E 20 72 65 61 64-79 79 0D 0A 00 00 00 00 .hen .ready......

01E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .................

01F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA .................

Шестнадцатеричный дамп сектора початкового завантаження MS DOS

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

Дескриптор Тип носія Версії MS DOS |

-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і|

OF8H Фіксований диск 2.3 |

OF0H 3,5-дюймовий, 2-сторонній, 18-секторний 3.2 |

OF9H 3,5-дюймовий, 2-сторонній, 9-секторний 3.2 |

OF9H 5,25-дюймовий, 2-сторонній, 15-секторний 3.x |

OFCH 5,25-дюймовий, 1-сторонній, 9-секторний 2.x,3.x |

OFDH 5,25-дюймовий, 2-сторонній, 9-секторний 2.x,3.x |

OFEH 5,25-дюймовий, 1-сторонній, 8-секторний 1.x,2.x,3.x |

OFFH 5,25-дюймовий, 2-сторонній, 8-секторний 1.x(крім 1.0)|

OFEH 8-дюймової, 1-сторонній, одинарної щільності 2.3 |

OFDH 8-дюймової, 2-сторонній, одинарної щільності |

OFEH 8-дюймової, 1-сторонній, подвійної щільності |

OFDH 8-дюймової, 2-сторонній, подвійної щільності |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

**Таблиця розміщення файлів**

Таблиця розміщення файлів представляє схему розміщення файлів в дискової пам'яті за допомогою вказівки того, які кластери й у якому порядку включаються в кожен файл. Для того, щоб MS DOS могла опредилити місце розташування файлу, опис даного файлу в каталозі містить номер його початкового входу в FAT. Цей вхід FAT у свою чергу, включає номер входу наступного кластера, якщо файл складається більш ніж з одного кластера, або номер останнього кластера, якщо файл містить єдиний кластер. Файл, що складається з 10 кластерів, повинен мати 10 входів в FAT й 9 зв'язків між входами FAT(безліч зв'язків для одного файлу називається ланцюжком - (chain)).

Додаткові копії FAT є резервними й використаються в випадку ушкодження первинної FAT; типові гнучкі й жорсткі диски тримають звичайно 2 FAT. Ці таблиці розташовуються послідовно за сектором початкового завантаження, а між ними іноді є резервна копія. Як правило, MS DOS використає первинну FAT, однак, при появлені в ній змін коректуються всі таблиці. Крім того, MS DOS порівнює всі таблиці при першому звертанні до диска, щоб переконатися, що вони відповідають один одному.

MS DOS підтримує два типи FAT: один використає 12-бітові посилання; іншої застосовується у версії 3.0 для включення в більші фіксовані диски більше 4087 кластерів і використає 16-бітові посилання.

Два перших входи FAT завжди резервуються й містять копію дескриптора носія даних (1 байт) і два (для 12-бітової FAT) або три (для 16-бітової FAТ) байти OFFH, як показано на прикладі наступних дампів перших 16 байтів FAT:

12-бітова FAT:

F9 FF FF 03 40 00 FF 6F-00 07 F0 FF 00 00 00 00

16-бітова FAT:

F8 FF FF FF 03 00 40 00-FF FF 06 00 07 00 FF FF

Інші входи FAT перебувають у взаємо-однозначній відповідності с кластерами в області файлів даних. Стан кожного кластера відмічається відповідним значенням в FAT (програма FORMAT спочатку позначає вхід FAT для кожного кластера, як вільний). Стан кластера може приймати одне з наступних значень:

Якщо вхід FAT містить ненульове значення, що відповідає кластер вже зайнятий. Вільний кластер можна знайти, скануючи FAT з початку до виявлення першого нульового значення. Дефектні кластери звичайно идентифицируются в процесі форматування. На мал.3-8 зображена типова ланцюжок FAT.

Вільні входи FAT містять нульове значення посилання; одиниця, як значення посилання, ніколи не використається. Таким чином, першим номером посилання, пов'язаним з першим доступним кластером в області файлів даних, є 2; цей номер ставиться до першого фізичного кластеру в області файлів даних. На мал.3-9 показаний зв'язок між файлами, входи FAT і кластери в області файлів даних.

Не існує логічного розходження між обробкою 12-бітових й 16-бітових входів FAT; розходження ставляться тільки до пам'яті й методів доступу. Оскільки процесор 8086 особливо призначений для эфективний обробки 8- або 16-бітових величин, то процедура доступу для 12-бітової FAT є більше складної, чим для 16-бітової.

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

12-бітовий 16-бітовий Значення |

-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і-і|

000H 0000H Вільний кластер |

001H 0001H Код не використається |

FF0-FF6H FFF0-FFF6H Резерв |

FF7H FFF7H Дефектний кластер |

(не використається) |

FF8-FFFH FFF8-FFFFH Останній кластер в |

файлі |

Всі інші значення Посилання на наступний |

кластер у файлі |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Вхід FAT

0 1 2 3 4 5 6 7 8 9

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

| FFDH| FFFH| 003H| 005H| FF7H| 006H| FFFH| 000H| 000H| 000H|

| 4093| 4095| 3 | 5 | 4087| 6 | 4095| 0 | 0 | 0 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_

| | | |

| | | |

| | | |

| | | |\_\_ не використаний:

| | | доступний кластер

| | |

| | |\_\_\_\_\_ використати не можна

| |

| |\_\_\_\_\_\_ не використається: не доступний

|

|

|\_\_\_\_\_\_ диск двосторонній з подвійною щільністю

**Розподіл пам'яті в FAT для типового диска MS DOS**

FAT є високоефективною "бухгалтерською" системою, однак тут можуть виникнути різні проблеми й необхідність компромісних рішень. Одна із проблем пов'язана з наявністю частково заповненого кластера наприкінці файлу. При цьому виникає проблема ефективності, пов'язана з використанням кластера більшого розміру, коли відбувається виділення цілого кластера, незалежно від числа содержащихся в ньому байтів. Наприклад, десять 100-байтовых файлів на диску із кластерами розміром по 16 Кбайт, займає 160 Кбайт дискової пам'яті; ті ж файли на диску із кластерами розміром 1 Кбайт займають тільки 10 Кбайт – різниця становить 150 Кбайт, що означає зменшення обсягу пам'яті в 15 разів при використанні кластерів меншого розміру. З іншого боку, програма для 12-бітової FAT, показана на мал.3-10, демонструє складність (а отже, низьку швидкість) переміщення у великому файлі, які мають довгий список посилань між невеликими кластерами. Таким образом, необхідно враховувати характер даних: додатка більших баз даних працюють ефективніше всього із кластерами більшого розміру; використовують кластери меншого розміру, можна розмістити на диску безліч невеликих текстових файлів драйвер, що розробляє, для дискового програміст звичайно встановлює розмір кластера).

Внаслідок відключення електроенергії або безладного виконання програм виникають проблеми руйнування каталогів або таблиць розміщення файлів, які , не будучи виправлені, можуть привести до більше серйозним проблемам. Програма MS DOS CHKDSK може виявляти й фіксують деякі дефекти. (см. КОМАНДИ КОРИСТУВАЧА CHKDSK). Наприклад, однієї з типових проблем є наявність списків "повислих" посилань (dangling allocation lists), викликане тим, що в каталозі відсутній покажчик на початок списку. Подібна ситуація часто буває, коли не був обновлений вхід у каталог, тому що не відбулося закриття файлу перед вимиканням або перезавантаженням комп'ютера. Наслідки цього досить необразливі: дані недоступні, однак це обмеження не впливає на інші операції по розміщенню файлів. Команда CHKDSK може вирішити цю проблему, створивши новий вхід каталогу й зв'язавши його зі списком.

Інші труднощі виникають, якщо розмір файлу у вході каталогу не збігається з його довжиною, що обчислює за допомогою перегляду списку посилань в FAT. Це може викликати невірне виконання програми і повідомлення про помилки MS DOS.

Більше складна й рідше виникаюча проблема зустрічається, коли вхід каталогу заданий правильно, однак список посилань (весь або деяка його частина) використаний також іншим входом каталогу. Виникає важка ситуація, тому що запис або додавання в один файл змінює вміст іншого. Ця помилка звичайно викликає серйозне руйнування даних й/або каталогу й приводить до краху системи.

Подібна проблема зустрічається, коли список зв'язків завершується вільним кластером замість номера останнього кластера. Якщо вільний кластер розподілений до виправлення цієї помилки, ситуація наприкінці кінців перетворюється в описаний вище випадок. Пов'язана із цим труднощі виникає, коли зустрічається значення посилання 1 або значення, перевищує розмір FAT.

Крім CHKDSK, для обслуговування FAT можна використати безліч комерційно доступних сервісних програм. Наприклад, програми реорганізації диска (disk reorganizers) можна застосовувати для зміни структури FAT й упорядкування каталогу так, щоб всі файли на диску були розміщені послідовно в області файлів даних й, зрозуміло, в FAT.

12-бітовий FAT FFFH 007H 000H

003H \_\_\_\_\_ \_\_\_\_ \_\_\_\_

Резерв \_\_\_\_\_ | | \_|\_ | \_|\_\_ |

\_\_\_\_|\_\_\_\_ \_|\_\_ | \_\_|\_ | 00 |07| | |00| |

| | | | | 00 | FF | 6F |\_\_\_\_| F0 FF 0

F9 FF FF 03 40 |\_\_\_| | | | |\_\_|

| | | | | |

| | | | | |

|\_\_\_\_\_\_| |\_\_\_\_\_| |\_\_\_|

004H

16-бітовий FAT

Резерв

| 0003H

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

| | | | | | | | | | | | | | | |

F8 FF FF FF 03 00 04 00 FF FF 06 00 07 00 FF FF 00 00

Вхід FAT

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

12-бітовий FAT |резерв| 003H| 004H| FFFH| 006H| 007H| FFFH| 000H|

16-бітовий FAT | |0003H|0004H|FFFFH|0006H|0007H|FFFFH|0000H|

|\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|

Вхід каталогу \_\_\_\_\_\_\_\_\_\_\_\_

(указує на |FILE1.TXT |

2 вхід FAT) |\_\_\_\_\_\_\_\_\_\_\_|

(указує на \_\_\_\_\_\_\_\_\_\_\_\_

5 вхід FAT) |FILE2.TXT |

|\_\_\_\_\_\_\_\_\_\_\_|

Область файлів даних Відповідний вхід FAT

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

| FILE1.TXT | 2

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| FILE1.TXT | 3

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| FILE1.TXT | 4

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| FILE2.TXT | 5

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| FILE2.TXT | 6

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| FILE2.TXT | 7

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

| не використається (доступний) | 8

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

**Відповідність між FAT й областю файлів даних**

;-і-і-і Одержати номер наступного посилання з 12-бітової FAT

;Параметри:

; ax = поточний номер ліворуч

; ds:bx = адреса FAT (повинна бути безперервної)

;

; Повертає:

; ax = номер наступного посилання

;

; Використає ax, bx, cx

next 12 proc near

add bx,ax /ds:bx - частковий індекс

shr ax,1/ax = зсув /2

/ зрушення не потрібний

pushf / зберегти зрушення

add bx,ax /ds:bx - індекс номера наступного кластера

mov ax,|bx| /ax = індекс номера наступного кластера

popf / зрушення не потрібний

jc shift / пропустити при використанні більше 12 біт

and ax,0fffh / менш 12 біт

ret

shift: mov cx,4/ cx = лічильник зрушень

shr ax,c1 /cx =

ret

next12 endp

**Ассемблерная програма для доступу до 12-бітового FAT**

**Корневий каталог**

Входи каталогу, як у кореневому, так й у підкаталогах, мають довжину 32 байта. Кожен вхід включає ім'я файлу й розширення, розмір файлу, початковий вхід в FAT , час і дату створення або останньої модифікації файлу і його атрибути. Ця структура нагадує формат блоків управління файлом (file control blocks - FCBs) для СР/М, використовуваний у файловій системі MS DOS, версії 1.х.

Угода про найменування файлів в MS DOS відбувається також з СР/М: восьмисимвольне ім'я файлу й наступний за ним трьохсимвольний тип файлу, вирівняні ліворуч і при необхідності доповнені пробілами. В межах обмежень символьного ряду ім'я й тип є абсолютно довільними. Час і дата задаються у форматі, використовуваному іншими функціями MS DOS і відбивають час останнього запису файлу.

Показаний дамп сектора каталогу обсягом 512 байт, що тримає 16 входів (кожен вхід у цьому прикладі займає два рядки). Байт зі зсувом ОАВН, що містить значення 10Н позначає, що вхід починаючи з адреси 0А0Н, ставиться до підкаталогу. Байт зі зсувом

;-і-і-і Одержати номер наступного посилання з 16-бітової FAT

;Параметри:

; ax = поточний номер ліворуч

; ds:bx = адреса FAT (повинна бути безперервної)

;

; Повертає:

; ax = номер наступного посилання

;

; Використає ax, bx, cx

next 16 proc near

add ax,ax /ax = зсув ліворуч

add bx,ax /ds:bx - індекс номера наступного посилання

mov ax,|bx| /ax = номер наступного посилання

ret

next16 endp

**Ассемблерная програма для доступу до 16-бітового FAT**

160Н, що містить 0Е5Н, означає, що файл вилучений. Байт зі зсувом

8ВН і значенням 08Н указує, що вхід каталогу, починаючи зі зсуву

80Н, є міткою тому. В остаточному підсумку, байт зі зсувом 1Е0Н

означає кінець каталогу, з огляду на, що наступні входи каталогу ніколи не використалися й, отже, їхній пошук не потрібно

Корнєвій каталог може мати спеціальний тип входу, що називається міткою тому, ідентифікується типом атрибута 08Н и використається для позначення імені диска. Корнєвій каталог може містити тільки одну мітку тому. Корнєвій каталог може також включати входи, указиває на підкаталоги; такі входи ідентифікуються типом атрибута 10Н і нульовим розміром файлу. Програми обробки підкаталогів повинні здійснює трасування ланцюжків кластерів в FAT.

Два інших особливих типи входів каталогу зустрічаються тільки в підкаталогах. Ці входи містять <імена\_файлів>. и.., що означає відповідно поточний каталог і батьківський каталог для поточні. Ці спеціальні входи, називані іноді альтернативними іменами каталогу (directory aliases), використаються для швидкого переміщення в структурі каталогу.

0 1 2 3 4 5 6 7 8 9 A B C D E F

0000 49 4F 20 20 20 20 20 20-53 59 53 27 00 00 00 00 10......SYS'.....

0010 00 00 00 00 00 00 59 53-89 0B 02 00 D1 12 00 00 ......YS....O....

0020 4F 53 44 4F 53 20 20 20-53 59 53 27 00 00 00 00 MSDOS...SYS'....

0030 00 00 00 00 00 00 41 49-52 0A 07 00 C9 43 00 00 ......AIR...IC..

0040 41 4E 53 49 20 20 20 20-53 59 53 20 00 00 00 00 ANSI.....SYS....

0050 00 00 00 00 00 00 41 49-52 0A 18 00 76 07 00 00 ......AIR...V...

0060 58 54 41 4C 4B 20 20 20-45 58 45 20 00 00 00 00 XTALK....EXE....

0070 00 00 00 00 00 00 F7 7D-38 09 23 02 84 0B 01 00 ......W18.......

0080 4C 41 42 45 4C 20 20 20-20 20 20 08 00 00 00 00 LABEL...........

0090 00 00 00 00 00 00 8C 20-2A 09 00 00 00 00 00 00 .........\*.D..R.

00A0 4C 4F 54 55 53 20 20 20-20 20 20 10 00 00 00 00 LOTUS...........

00B0 00 00 00 00 00 00 E0 OA-E1 C6 A6 01 00 00 00 00 ......'.a.&.a...

00C0 4C 54 53 4C 4F 41 44 20-43 4F 4D 20 00 00 00 00 LTSLOAD.COM.....

00D0 00 00 00 00 00 00 E0 0A-E1 06 A7 01 A0 27 00 00 ......'.a.......

00E0 4D 43 49 2D 53 46 20 20-58 54 4B 20 00 00 00 00 MCI-SE...XTK....

00F0 00 00 00 00 00 00 46 19-32 0D B1 01 79 04 00 00 .......F.2.1.y..

0100 58 54 41 4C 4B 20 20 20-48 4C 50 20 00 00 00 00 XTALK...HLP.....

0110 00 00 00 00 00 00 C5 6D-73 07 A3 02 AF 88 00 00 ......Ems.#.....

0120 54 58 20 20 20 20 20 20-43 4F 4D 20 00 00 00 00 TX COM....

0130 00 00 00 00 00 00 05 61-65 0C 39 01 E8 20 00 00 .................

0140 43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00 COMMAND COM .....

0150 00 00 00 00 00 00 41 49-52 0A 27 00 55 3F 00 00 ......AIR.'.UP...

0160 E5 32 33 20 20 20 20 20-45 58 45 20 00 00 00 00 e23 EXE.....

0170 00 00 00 00 00 00 9C B2-85 0B 42 01 80 5F 01 00 ........2..B.....

0180 47 44 20 20 20 20 20 20-44 52 56 20 00 00 00 00 GD.......DRV.....

0190 00 00 00 00 00 00 E0 0A-E1 06 9A 01 5B 08 00 00 .......'a...'....

01A0 4B 42 20 20 20 20 20 20-44 52 56 20 00 00 00 00 KB.......DRV.....

01B0 00 00 00 00 00 00 E0 0A-E1 06 9D 01 60 01 00 00 .......'a...'....

01C0 50 52 20 20 20 20 20 20-44 52 56 20 00 00 00 00 PR.......DRV.....

01D0 00 00 00 00 00 00 E0 0A-E1 06 9E 01 49 01 00 00 .......'a..'.....

01E0 00 F6 F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 .................

01F0 F6 F6 F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 .................

Шестнадцатеричный дамп сектора каталогу обсягом 512 байт

**Область файлу**

Область файлу містить підкаталоги, даних файлу й нерозподілені кластери. Область розділяється на кластери фіксованого розміру й використання кожного кластера регламентується відповідним входом FAT.

**Практична частина**

**Лістинг модулів програми**

Модуль, який блокує запуск COM та EXE файлів

#include <fcntl.h>

#include <io.h>

#include <dos.h>

#include <stdio.h>

#include <string.h>

#include <process.h>

unsigned char buffer[3];

FILE \*fd;

int handle;

fail(char \*);

void pass(char \*);

main(int argc,char \*argv[])

{

int n;

if(argc!=3)

{

printf("\nFailpass Utility v1.0 Email: chingpongin@yahoo.co.in");

printf("\nEnable/Disable .EXE or .COM execution");

printf("\nsyntex failpass fail <filename.extension>");

printf("\n or");

printf("\n failpass pass <filename.extension>");

exit(2);

}

strupr(argv[1]);

n=strcmp(argv[1],"FAIL");

if(n==0)

{

fail(argv[2]);

exit(2);

}

n=strcmp(argv[1],"PASS");

if(n==0)

{

pass(argv[2]);

exit(2);

}

return 0;

}

fail(char \*fname)

{

\_chmod(fname,1,FA\_ARCH);

fd=fopen(fname,"rb+");

if(fd==NULL)

{

printf("\nCann't open %s\n\n",fname);

exit(2);

}

fseek(fd,0,SEEK\_SET);

fread(buffer,2,1,fd);

if(buffer[0]==0xcd)

if(buffer[1]==0x20)

{

fclose(fd);

printf("\n%s already protected...\n\n",fname);

exit(2);

}

fseek(fd,0,SEEK\_END);

fwrite(buffer,2,1,fd);

fseek(fd,0,SEEK\_SET);

buffer[0]=0xcd; /\* don't take any PANGA with this line \*/

buffer[1]=0x20; /\* don't take any PANGA with this line \*/

fwrite(buffer,2,1,fd);

fclose(fd);

return 0;

}

void pass(char \*fname)

{

int xyz;

fd=fopen(fname,"rb+");

if(fd==NULL)

{

fclose(fd);

printf("\nCannot open file %s\n\n",fname);

exit(2);

}

fseek(fd,0,SEEK\_SET);

fread(buffer,2,1,fd);

if(buffer[0]!=0xcd)

if(buffer[1]!=0x20)

{

fclose(fd);

printf("\n%s is NOT protected....\n\n",fname);

exit(2);

}

fseek(fd,-2,SEEK\_END);

fread(buffer,2,1,fd);

fseek(fd,0,SEEK\_SET);

fwrite(buffer,2,1,fd);

fclose(fd);

handle=open(fname,O\_RDWR);

chsize(handle,(filelength(handle))-2);

close(handle);

}

Модуль, який пише та читає інформацію напряму в ділянку диску без звертання до FAT системи

include common.mac

text segment use16

assume CS:text, DS:text

org 256

myproc proc

mov AH,09h ;printing function

printf g ;print g

printf g1 ;print g1

;parse the command line...

getf guff ;guff will have the command line

;is the command line empty?

cmp guff[0], 0

je expl

;is the command line starting with ?

cmp guff[0], '?'

je expl

;we are reading the disk

cmp guff[0], '^'

je readdisk

;nothing else but to write...

jmp writedisk

writedisk:

printf g2

waitenter

mov AH, 09h ;writing out

printf g3 ;what are we doing

printf guff ;file name

call disk\_write ;write to disk

jmp rc

readdisk:

printf g2

waitenter

call disk\_read ;write from disk

jmp rc

expl:

printf gline

jmp rc

rc: ;return

mov AX, 4C00h

int 21h

myproc endp

;====================================================================

;====================================================================

;==================== Procedure: disk\_read =======================

;====================================================================

;====================================================================

disk\_read proc

mov AH, 02h ;read raw input

mov AL, 1 ;number of sectors

mov CH, 0 ;cylinder

mov CL, 1 ;sector

mov DH, 1 ;head

mov DL, 0 ;disk drive - floppy

lea BX, guff

int 13h ;call BIOS

jnae ocr

;now guff stores file name

mov AH, 02h

mov AL, 1

mov CH, 1

mov CL, 1

mov DH, 0

mov DL, 0

lea BX, file

int 13h ;call BIOS

jnae ocr

;decrypt data

call chiper

;write output to the file saved

mov AH, 3Ch ;create file

mov CX, 0 ;no attributes

lea DX, guff ;name

int 21h ;call DOS

mov handle, AX ;get file handle

mov AH, 40h ;write to file

mov BX, handle ;handle

;get len

mov CX, 0 ;dump CX and SI

mov SI, 0

clen:

cmp file[SI], 0 ;is it zero yet?

je clenf ;this is the len

inc SI ;increase counting

inc CX ;increase counting

jmp clen ;jump back

clenf:

lea DX, file ;file data

int 21h ;call DOS

mov AH, 3Eh ;close file

mov BX, handle ;handle

int 21h ;call DOS

;open file for writing....

mov AH, 09h

printf success

jmp finr

ocr:

mov AH, 09h

printf error

finr: ret

disk\_read endp

;====================================================================

;====================================================================

;====================================================================

;====================================================================

;==================== Procedure: disk\_write ======================

;====================================================================

;====================================================================

disk\_write proc

;write file name

mov AH, 03h ;write raw output

mov AL, 1 ;write 1 sector

mov CH, 0 ;cylinder zero

mov CL, 1 ;sector no 1

mov DH, 1 ;head 1

mov DL, 0 ;disk drive A:

lea BX, guff ;set buffer

int 13h ;call BIOS

;read file...

mov AH, 3Dh ;open file

mov AL, 2 ;read-write

lea DX, guff ;file name

int 21h ;call DOS

mov handle, AX ;get handle

mov AH, 3Fh ;reading file

mov BX, handle ;set handle

mov CX, 1024 ;reading

lea DX, file ;buffer

int 21h ;call DOS

;encrypt data

call chiper

;looping through each

mov AH, 03h

mov AL, 1

mov CH, 1

mov CL, 1

mov DH, 0

mov DL, 0

lea BX, file

int 13h

;show message

jnb okw

occ:

mov AH, 09h

printf error

jmp fin

okw:

mov AH, 09h

printf success

fin: ret ;return backwards

disk\_write endp

;====================================================================

;====================================================================

;====================================================================

;====================================================================

;==================== Procedure: chiper ==========================

;====================================================================

;====================================================================

chiper proc

mov SI, 0

;xor values

chipe:

cmp file[SI], 0

je chipf ;jump if zero...

xor file[SI], 3

inc SI ;increase counter

jmp chipe

chipf:

ret ;return

chiper endp

;====================================================================

;====================================================================

g db 'Welcome to Disk Encryptor 1.1...', 10, 13, 'This program allows you to save encrypted data onto the disk....', 10, 13, 'Features: Data Chipering, Maximum of 1024 bytes of raw data', 10, 13,'Software Written in Assembler', 10, 13, '$'

gline db 10, 13, '? - help', 10, 13, '^ - reads a file from the floppy', 10, 13, '[Filename] - Saves the file onto a floppy', 10, 13, '$'

g1 db 'Copyright (C) Daniel Moisyeyev 2002. All Rights Reserved.', 10, 13, 10, 13, '$'

g2 db 10, 13, 'Please insert a floppy and press any key...', 10, 13, '$'

g3 db 10, 13, 'About to write $'

error db 10, 13, 'Error occured$'

success db 10, 13, 'Operation Completed$'

num db 4 dup(0), '$'

guff db 15 dup(0)

;file storage

handle dw 0

file db 1024 dup(0)

text ends

end myproc

**Робота з програмою**

Розроблена програма являє собою дві утіліти командного рядку. Перша дозволяє блокувати запуск виконавчіх файлів COM та EXE:

failpass fail <filename.extension>

failpass pass <filename.extension>

друга – тестує роботу з ділянкою диску в обхід FAT системи

Please insert a floppy and press any key...

About to write

Operation Completed

**Висновки**

Наприкінці роботи відмітимо лише, що доступ до блокових пристроїв здійснюється по секторах. Ядро MS-DOS за допомогою драйвера пристрою розглядає блоковий пристрій як логічний кінцевий масив секторів і припускає, що цей масив містить припустиму в MS-DOS файлову організацію.

Драйвер пристрою, у свою чергу, перетворить запит MS-DOS на логічний сектор у фізичну адресу блокового пристрою. Первісна файлова система MS DOS записана за допомогою програми MS-DOS FORMAT (див. теоретичну частину роботи).

**Література**

1. Нейбауэр А. Моя первая программа на С/С++. П., 1995. 368 С.

2. Бруно Бабэ. Просто и ясно о Borland C++. М., 1996. 400 С.

3. ObjectWindows для C++. К., 1993., 208 С.

4. Н. Барканати. Программирование игр для Windows на Borland C++. М., 1994.