НАВЧАЛЬНІ ВИДАННЯ

ПРАКТИКУМ З СИСТЕМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КИЇВ НУБіП 2020

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ

Савицька Я.А., Смолій В.В.,
Чичикало Н.І., Шкарупило В.В.

ПРАКТИКУМ З СИСТЕМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

для студентів спеціальності «123 Комп'ютерна інженерія»

КИЇВ, НУБіП
2020

УДК 004.45
С

Рекомендовано до видання рішенням вченої ради Національного університету біоресурсів і природокористування України (Протокол № 4 від 26 листопада 2019 року)
Перевидання – перероблене та доповнене.

Рецензенти:
Лахно В.А., доктор технічних наук, професор, завідувач кафедри комп'ютерних систем і мереж Національного університету біоресурсів і природокористування України
Чумаченко С.М., доктор технічних наук, професор, завідувач кафедри інформаційних систем Національного університету харчових технологій
Ясковець І.І., доктор фізико-математичних наук, провідний науковий співробітник Інституту фізики НАН України.

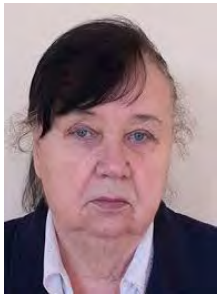
Практикум з Системного програмного забезпечення. [навчальний посібник] / В.В. Смолій В.В., Савицька Я.А., Шкарупило В.В., Чичикало Н.І. (Перевидання) // - К.:НУБіП України, 2020.- 265с.

Навчальний посібник призначений для студентів вищих навчальних закладів ОС «Бакалавр», які навчаються за спеціальністю 123 – Комп’ютерна інженерія. Розділи посібника охоплюють основні питання з обслуговування ОС Windows, зберіганням даних на дискових накопичувачах та архівації даних. Увесь матеріал подано у формі практичних задач, що доволі часто виникають як у адміністратора системи так і у звичайних користувачів.

Навчальний посібник відповідає програмі дисципліни "Системне програмне забезпечення".

УДК 004.45
С

ВІДОМОСТІ ПРО АВТОРІВ



Чичикало Ніна Іванівна

Професор кафедри ЗТтаРІ КПІ ім. Ігоря Сікорського, доктор технічних наук за спеціальністю 05.11.16 – Інформаційно-вимірювальні і клерувальні системи. Автор понад 600 наукових та методичних праць, 25 патентів. Ветеран праці, винахідник СРСР, академік міжнародної академії стандартизації, академік академії метрології України, член асоціації електронного приладо- та машино-будівництва, відмінник освіти. Наукові напрями роботи:

– розробка ІВС та Аудіовізуальних систем управління процесами для технічної, харчової та хімічної промисловості;

– медико-біологічні прилади та системи.

Савицька Яна Артурівна

Науковий співробітник, кандидат технічних наук за спеціальністю 05.13.06 – Інформаційні технології. Автор понад тридцяти наукових робіт у області інформаційних технологій, з яких 1 монографія, 10 методичних розробок, 3 авторських свідоцтва.

Наукові напрями роботи:

– інформаційні технології управління складними промисловими комплексами;

– технічні засоби захисту інформації.

Шкарупило Вадим Вікторович

Доцент, кандидат технічних наук за спеціальністю 05.13.05 – Комп'ютерні системи та компоненти. На кафедрі комп'ютерних систем і мереж НУБіП викладає наступні дисципліни: "Мікропроцесорні системи керування", "Системне програмування", "Протоколи передачі даних в IoT системах". Основні напрямки наукових досліджень:

– розробка й дослідження моделей і методів специфікації, верифікації і валідації композитних веб-сервісів;

– автоматизовані системи передачі даних в IoT системах;

– програмні засоби в комп'ютерних системах.

Смолій Віктор Вікторович

Доцент, кандидат технічних наук за спеціальністю 05.13.13 – Обчислювальні машини, системи та мережі. Є автором понад п'ятдесяти наукових та навчально-методичних праць, 2 з яких монографії. На кафедрі комп'ютерних систем і мереж НУБіП викладає наступні дисципліни:

–архітектура комп'ютерів;

–технічні засоби передачі інформації;

–системи візуалізації і розпізнавання образів.

Наукові напрями роботи:

– інтерактивні геоінформаційні комплекси реального часу;

– інтегровані геоінформаційні системи аграрного застосування;

– вбудовані мікроконтролерні засоби інтернету речей.

ЗМІСТ

ПЕРЕДМОВА

Системне програмне забезпечення – це програмне забезпечення для оптимізації роботи програмних та апаратних засобів, пошук системних вад та помилок, налаштування особливостей функціонування конкретної системи на різних рівнях застосувань, засоби розробки нового програмного та системного забезпечення. Кожен з цих напрямів відіграє важливу роль у ефективному використанні комп'ютерної або комп'ютеризованої системи в цілому, що обумовлює необхідність володіння цими питаннями для кваліфікованого фахівця та надбання відповідних фахових компетенцій.

Дисципліна «Системне програмне забезпечення» є однією з базових дисциплін за спеціальністю 123 «Комп'ютерна інженерія» та включена до переліку обов'язкових дисциплін у освітніх програмах бакалавріату.

Основні цілі видання полягають у:

- ознайомленні студентів з теоретичними матеріалами з принципів побудови систем обслуговування засобів операційних систем та супутнього програмного та апаратного забезпечення;
- оволодінні практичними навичками з аналізу та локалізації проблем, які виникають при роботі комп'ютерних систем;
- оволодінні практичними навичками усунення виявлених проблем та використанні відповідних спеціалізованих інструментальних засобів.

Відповідно до цього, наданий матеріал містить компоненти що створюють логічний ланцюг від «проблеми» до її вирішення, пропонується перелік контрольних питань та завдань

Чичикало Н.І. виклала передмову, матеріали за темою «Обчислення та перевірка контрольної суми» викладено Шкарупило В.В., Савицька Я.А. є розробником РНП з дисципліни, структури та складу викладеної тематики та сумісно з Смолій В.В. автором основної частини роботи.

РОЗДІЛ 1. ПРИНЦИПИ ОРГАНІЗАЦІЇ ДАНИХ НА ЖОРСТКИХ МАГНИТНИХ ДИСКАХ

Тема 1.1. Логічна структура жорсткого диска

Мета: Навчитися визначати види розмітки та характеристики накопичувачів на жорстких магнітних дисках і виконувати аналіз цілісності файлової системи.

1.1.1. Теоретичні відомості

Перед тим як почати використовувати жорсткий диск, його необхідно розбити на розділи – частини його довгострокової пам'яті, логічно виділені для зручності роботи, що складаються з суміжних блоків. Інформація про розділи диска містить такі дані:

– початок і кінець розділів;
– належність до розділу конкретного сектора диска;
– вид розділу (основний або завантажувальний).

На сьогодні існує два типи розмітки дисків:
– MBR (головний запис);
– GPT (таблиця розділів GUID).

Тип MBR з'явився досить давно, в 80-х роках минулого століття. Основне обмеження, яке можуть помітити володарі великих дисків – це те, що MBR працює з дисками, розмір яких не перевищує 2Тб (хоча, за певних умов можна використовувати диски і більшого розміру) і підтримує лише 4 основні розділи.

GPT – це відносно нова розмітка і у неї немає обмежень, як у MBR: диски можуть бути набагато більше 2 Тб. Крім цього GPT дозволяє створювати необмежену кількість розділів (обмеження в даному випадку накладає використовувана ОС).

У GPT є одна незаперечна перевага: якщо MBR пошкодиться – то виникне помилка і збій при завантаженні ОС, тому що дані MBR зберігає лише в одному місці. GPT ж зберігає кілька копій даних, завдяки чому, якщо пошкодитися одна з них – він відновить дані з іншого місця.

Так само варто відзначити, що GPT працює паралельно з UEFI, який прийшов на зміну BIOS, і має більш високу швидкість завантаження, підтримує безпечне завантаження, зашифровані диски та ін.

Перевірка розмітки диска (MBR або GPT) через меню управління дисками. Спочатку необхідно відкрити панель управління ОС Windows і перейти за наступним шляхом: Панель управления / Система и безопасность / Администрирование (рис. 1.1).

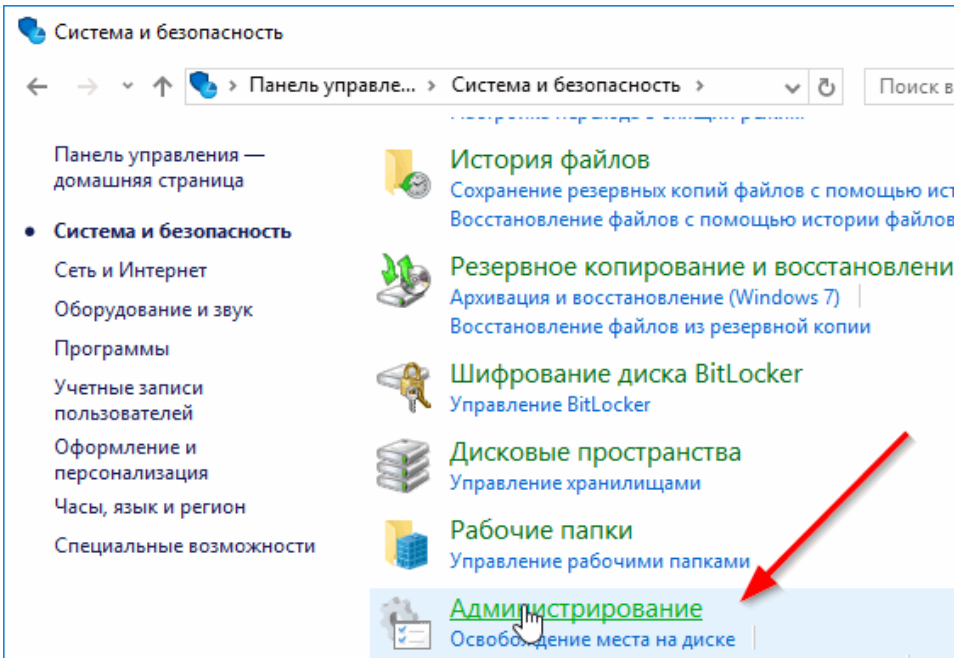


Рис. 1.1. Перехід на вкладку «Администрирование»

Далі необхідно перейти в оснастку «Управление компьютером» (рис. 1.2).

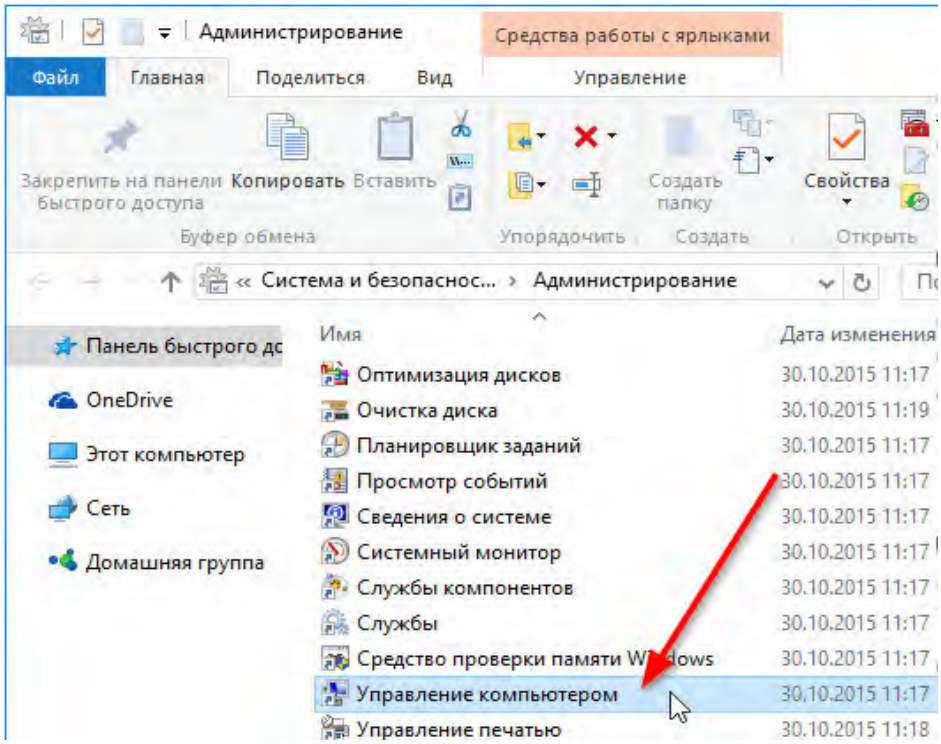


Рис. 1.2. Оснастка «Управление компьютером»

Після чого в меню зліва відкрити розділ «Управление дисками», а в списку дисків справа, вибрати потрібний диск і перейти в його властивості (рис. 1.3).

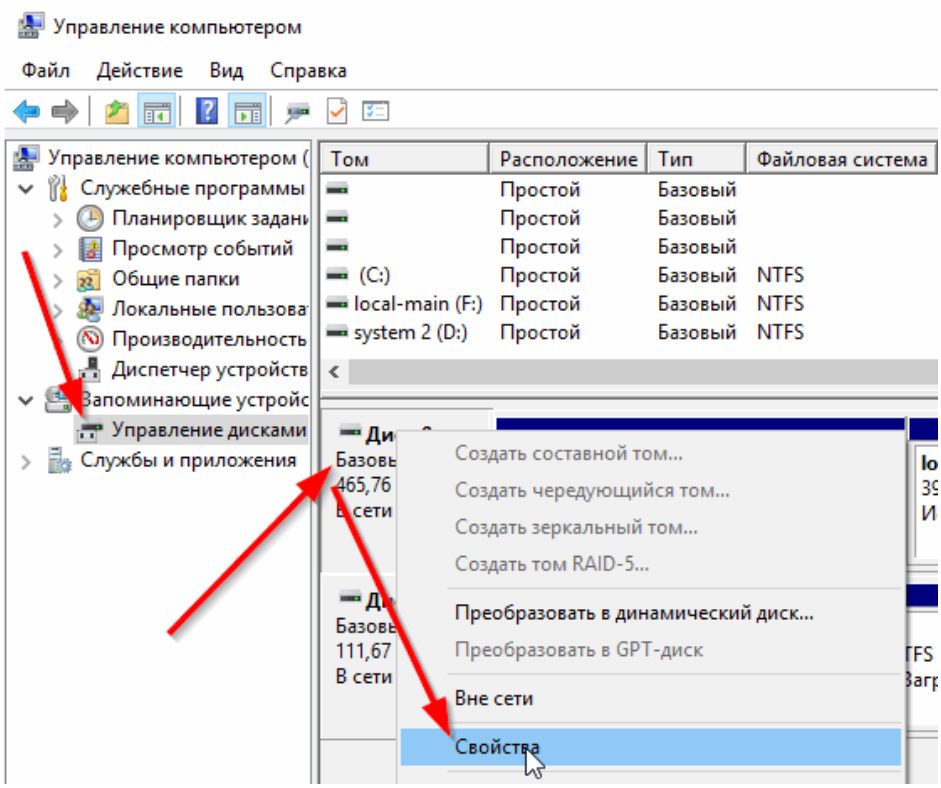


Рис. 1.3. Властивості диска

Далі в розділі «Тома», навпроти рядка «Стили раздела» зазначено вид розмітки диска. На рис. 1.4 представлений диск з розміткою MBR.

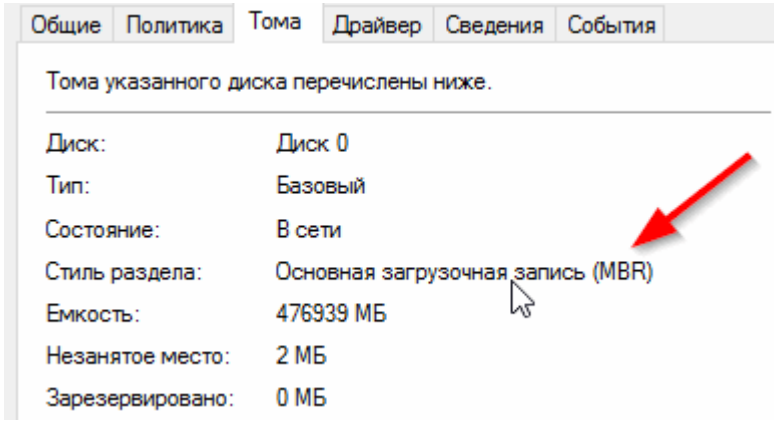


Рис. 1.4. Диск з розміткою MBR

На рис. 1.5 наведено приклад диска з розміткою GPT.

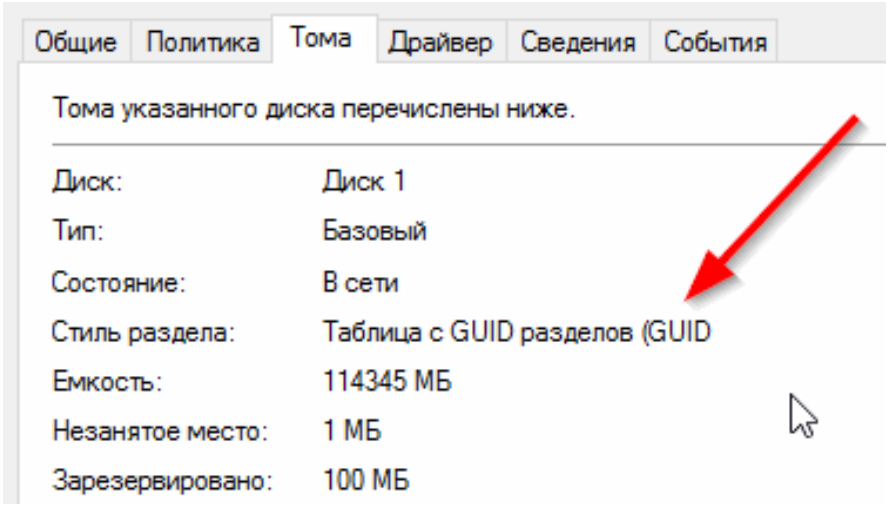


Рис. 1.5. Приклад диска з розміткою GPT

Визначення розмітки диска через командний рядок.

Досить швидко визначити розмітку диска можна за допомогою командного рядка:

1. Спочатку натисніть поєднання клавіш Win + R щоб відкрити вкладку «Виконати» (або через меню Пуск). У вікні виконати введіть команду diskpart і виконайте її (рис. 1.6).

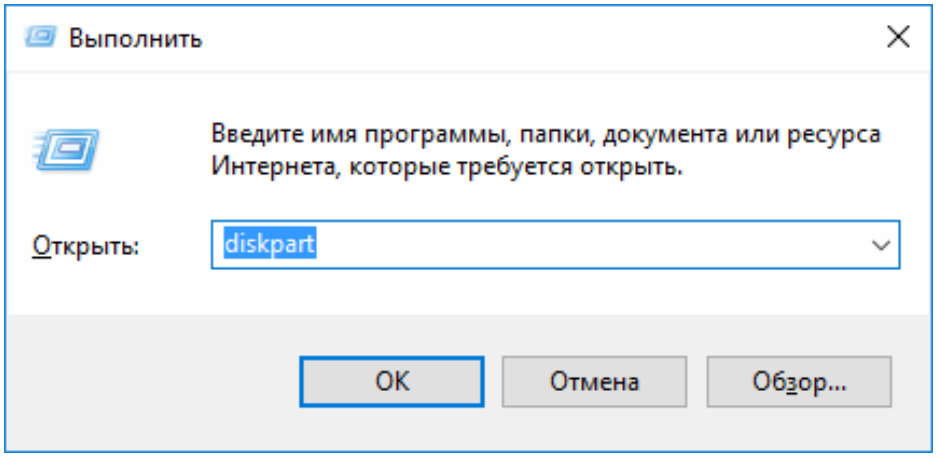


Рис. 1.6. Команда diskpart

Далі в командному рядку впишіть команду list disk і натисніть Enter. Після цього з'явиться список всіх дисків, підключених до системи.

Зверніть увагу серед цього списку на останній стовпчик GPT: якщо напроти конкретного диска стоїть знак "*" – це означає, що диск з розміткою GPT (рис. 1.7).

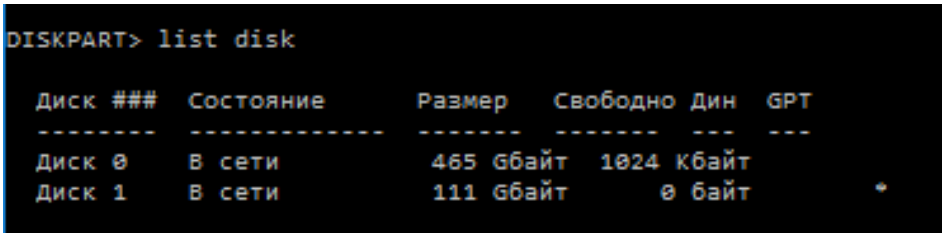


Рис. 1.7. Визначення розмітки диска через командний рядок

Визначення кількості вільних кластерів на диску. Логічною одиницею зберігання даних на жорсткому диску є кластер. Всі кластери конкретного вінчестера однакові за розміром і пронумеровані по порядку. Кожному кластеру відповідає службовий рядок, що має той же номер. Сукупність таких службових рядків складає FAT-таблицю (File Allocation Table). Номер кластера, з якого починається розміщення файлу є адресою його місцезнаходження на диску. Рядок FAT-таблиці містить наступні відомості про кластер:

– кластер може бути вільним;

– кластер може бути непридатним для зберігання даних;

– кластер може зберігати частину файлу, файл цілком або останню частину файлу.

Якщо файл не поміщається в один кластер, то він розміщується в декількох кластерах. Якщо диск перший раз заповнюється інформацією, то файл буде розміщений в суміжних кластерах, у вигляді одного цілісного блоку. Але, в процесі роботи одні файли видаляються, інші – додаються. Виникає ситуація, коли для розміщення одного файлу будуть потрібні кластера в різних областях дискового простору. У цьому випадку файл буде складатися з декількох блоків.

У будь-якому випадку зв'язок між кластерами і блоками здійснюється за допомогою FAT-таблиці, яка «склеює» файл із шматків. Коли файл пишеться на диск, в таблицю FAT

заноситься початкова адреса місцезнаходження файлу – номер першого кластера з числа тих, які він буде займати. Рядок FAT-таблиці, який містить запис про початковий кластер файлу, зберігає номер наступного кластера, який містить продовження файлу і т. д. За допомогою такого ланцюжка посилань фіксується розташування всього файлу на диску.

Записуючи файл на диск, система (якщо це можливо) розміщує його безперервно – в суміжних кластерах. Якщо цього зробити не можна, то система шукає необхідну кількість вільних кластерів в різних місцях дискового простору і розміщує файл в них. Таким чином, файл (особливо, якщо він великих розмірів) може бути «склеєний» з декількох десятків блоків. На ємність диска це ніяк не впливає, але швидкість доступу до такого фрагментованого файлу різко падає, оскільки для його зчитування система витрачає додаткові механічні (повільні) операції з переміщення голівок з доріжки на доріжку.

При видаленні файлу в FAT-таблиці проводяться відповідні записи – кластера, в яких розміщувався цей файл, позначаються як вільні, але при цьому сама інформація в кластері залишається до того моменту, поки не буде записаний новий файл в цей кластер. На цьому заснована робота деяких програм з відновлення видаленої інформації.

Розрядність рядків FAT-таблиці визначає розрядність адрес, тобто, фактично кількість кластерів на жорсткому диску. Якщо система оперує з 16-розрядними рядками (в цьому випадку, говорять про таблиці FAT16), то максимально можлива кількість кластерів дорівнює $2^{16} = 65535$ штук (64 Кб); для 32-розрядних рядків (FAT32) – число кластерів дорівнює $2^{32} = 4294967296$ штук (4 Гб).

Перевірка цілісності файлової системи. Рекомендовано періодично виконувати перевірку цілісності файлової системи за допомогою спеціалізованих утиліт. У Windows використовують вбудовану утиліту CHKDSK, яка має наступний синтаксис в командному рядку:

chkdsk.exe диск: [\шлях] [/ключ] [/ ключ]…

Основні ключі утиліти наведені в табл. 1.

Таблиця 1.1

Головні ключі утиліти chkdsk

Ключ	Назва
шлях	Визначає імена файлів для перевірки. Доступний тільки для логічних дисків FAT.
/F	Перевірка з автоматичним виправленням помилок.
/V	На логічному диску FAT видає ім’я кожного перевіреного файлу. На логічному диску NTFS повідомлення про очищення.
/R	Пошук збійних секторів і спроба відновлення даних. Використовується разом з ключем /F.

Повний перелік ключів утиліти chkdsk можна переглянути ввівши команду: chkdsk /?

При запуску утиліти без ключів отримують результати перевірки файлової системи (рис. 1.8).

З рис. 1.8 видно, що перевірка файлової системи виконується в три етапи – перевірка файлів, перевірка індексів, перевірка дескрипторів безпеки. Отримані результати свідчать про відсутність помилок на диску (0 кБ у пошкоджених секторах).

У разі виявлення пошкоджених секторів рекомендовано запустити утиліти з ключем /F, а після цього повторно виконати перевірку файлової системи без ключів.

Рис. 1.8. Результат перевірки файлової системи

1.1.2. Завдання до самостійного виконання

1. Визначити вид розмітки жорсткого диска за допомогою панелі управління і командного рядка. Привести відповідні скріншоти. Чи можуть логічні диски в межах одного розділу мати різні види розмітки?

2. У таблиці наведено приклад фрагмента файлової системи MS Windows FAT * – вміст каталогу і таблиці FAT. Проаналізуйте цілісність даної файлової системи, вважаючи, що eof – останній кластер файлу, bad – дефектний кластер і не заповнений елемент таблиці – вільний кластер.

Ім’я файла	Номер кластера
A	2
B	15
C	30
D	18

0		8		16	6	24	
1		9		17		25	26
2	3	10	16	18	19	26	27
3	4	11	bad	19	13	27	28
4	5	12	eof	20		28	eof
5	eof	13	12	21	bad	29	
6	eof	14	13	22		30	10
7		15	14	23		31	

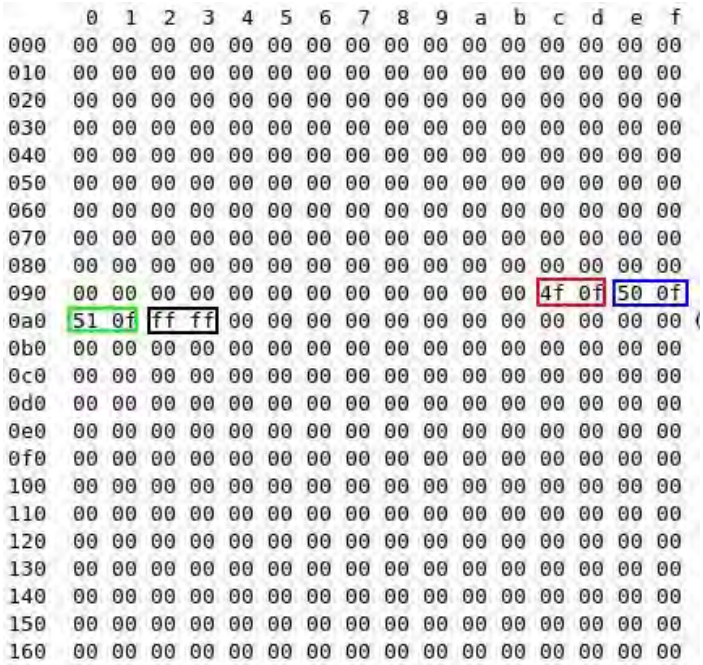
2. Завдання для непарних варіантів.

На рис. 1.8 наведено початок FAT таблиці диска з файловою системою FAT32. Проаналізуйте даний фрагмент. Скільки вільних кластерів він містить? Відповідь поясніть.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	28	00	00	00	FF	FF	FF	0F
FF	FF	FF	0F	FF	FF	FF	0F	00	00	00	00	00	00	00	00	0D	00	00	00
FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	15	00	00	00
19	00	00	00	FF	FF	FF	0F	FF	FF	FF	0F	1C	00	00	00	1D	00	00	00
FF	FF	FF	0F	60	00	00	00	FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F
FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	7D	08	00	00	2D	00	00	00
31	00	00	00	32	00	00	00	33	00	00	00	A6	00	00	00	35	00	00	00
39	00	00	00	3A	00	00	00	FF	FF	FF	0F	3C	00	00	00	3D	00	00	00

Рис. 1.8. Завдання для парних варіантів

3. На рис 1.9 наведено початок FAT таблиці диска. Визначте тип файлової системи (FAT16 або FAT32) і кількість вільних кластерів. Відповідь поясніть.



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
090	00	00	00	00	00	00	00	00	00	00	00	00	4f	0f	50	0f
0a0	51	0f	ff	ff	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Рис. 1.9. Завдання для непарних варіантів

4. На жорсткому диску використовується розміщення файлів у вигляді зв'язного списку кластерів. Посилання на кластер займає 4 байта. Запис каталогу разом з атрибутами файлу займає 32 байта. Витрата ємності диска на завантажувальний запис не враховується. Вважаючи, що:

• ємність диска – 20 Гбайт,

• розмір кластера – 8 Кбайт.

Знайти частку адресної інформації на диску.

5. У якій файловій системі ефективніше зберігати дані про іконку розміром 32x32 точок, в якій використано 128 кольорів?

a) 1 сектор = 1 кластер;

b) 2 сектора = 1 кластер;

c) в) ефективність однакова.

1.1.3. Запитання для самоперевірки

1. Що називається розміткою диска? Які види розмітки існують?

2. Які види розділів існують в ОС Windows? Коротко охарактеризуйте їх.

3. Поясніть різницю між швидким і повним форматуванням жорсткого диска. Які процеси відбуваються в тому і в іншому випадку?

4. Визначити мінімальний розмір кластера на флешці об'ємом 2 Гб і файловою системою FAT32.

5. Файл Lab1.doc має інформаційний розмір 16248 байт. Розмір кластера на логічному диску – 8 секторів. Скільки кластерів буде виділено файлу?

6. Скільки вільного місця залишиться в останніх секторі і кластері виділених файлу Lab1.doc?

7. У чому полягає перевірка на цілісність файлової системи? Що таке bad сектор? Чи має він логічну адресу? Відповідь поясніть.

8. На диск записані файли різних форматів. Чи залишаться вони на диску в разі:

а) пошкодження FAT1?

б) пошкодження FAT1 і FAT2?

в) пошкодження MBR?

9. Чому дорівнює у байтах розмір запису в таблиці FAT для файлової системи FAT16?

10. Чому дорівнює у байтах розмір запису в таблиці FAT для файлової системи FAT32?

11. Який сектор називають пошкодженим? За яким критерієм визначається справність секторів?

12. Назвіть процеси, які відбуваються під час перевірки сектора на справність.

13. Яка мінімальна кількість вільних кластерів має бути на диску для здійснення процесу дефрагментації?

Тема 1.2. Перевірка працездатності жорсткого диску

Мета: перевірити працездатність жорсткого диску за допомогою системного програмного забезпечення Victoria і HDDScan.

1.2.1. Теоретичні відомості

Перевірка жорсткого диску на працездатність і її відновлення у програмі Victoria.

У випадку нестабільної роботи комп'ютера, наприклад, зависанні при зверненні до файлів, довгому копіюванні інформації з одного розділу жорсткого диска на інший, пропажі файлів і директорій і т.д. рекомендовано виконати перевірку жорсткого диску за допомогою системного програмного забезпечення Victoria.

Перевагою Victoria є робота з обладнанням на низькому рівні, тобто програма звертається до жорсткого диску безпосередньо, а не зчитує визначені системою або іншим системним програмним забезпеченням дані.

Victoria має велику базу для роботи з жорсткими дисками і зовнішніми пристроями і є безкоштовним системним програмним забезпеченням як для 32-, так і для 64-бітних ОС сімейства Windows. Підтримує версії від XP до Windows10. Останню версію програми можна скачати за посиланням: https://victoria.en.lo4d.com/windows.

Після скачування програми Victoria, вміст архіву необхідно розпакувати і запустити виконуваний файл від імені адміністратора. Для цього необхідно в контекстному меню файлу обрати команду «Запуск від імені адміністратора» (рис. 1.10). Оскільки програма не потребує встановлення на комп'ютер, то запуск здійснюється лише таким чином.

Після запуску відкриється вікно програми, у якому необхідно перейти на вкладку «Standard». У верхній правій частині показані жорсткі диски і CD-Rom, встановлені в системі. Серед них необхідно обрати жорсткий диск для

тестування і натиснути кнопку «Passport» (рис. 1.11). Далі програма має визначити і показати модель жорсткого диска.

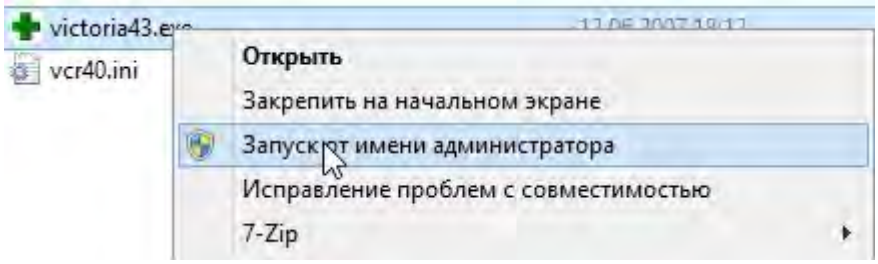


Рис. 1.10. Запуск виконуваного файлу програми Victoria від імені адміністратора

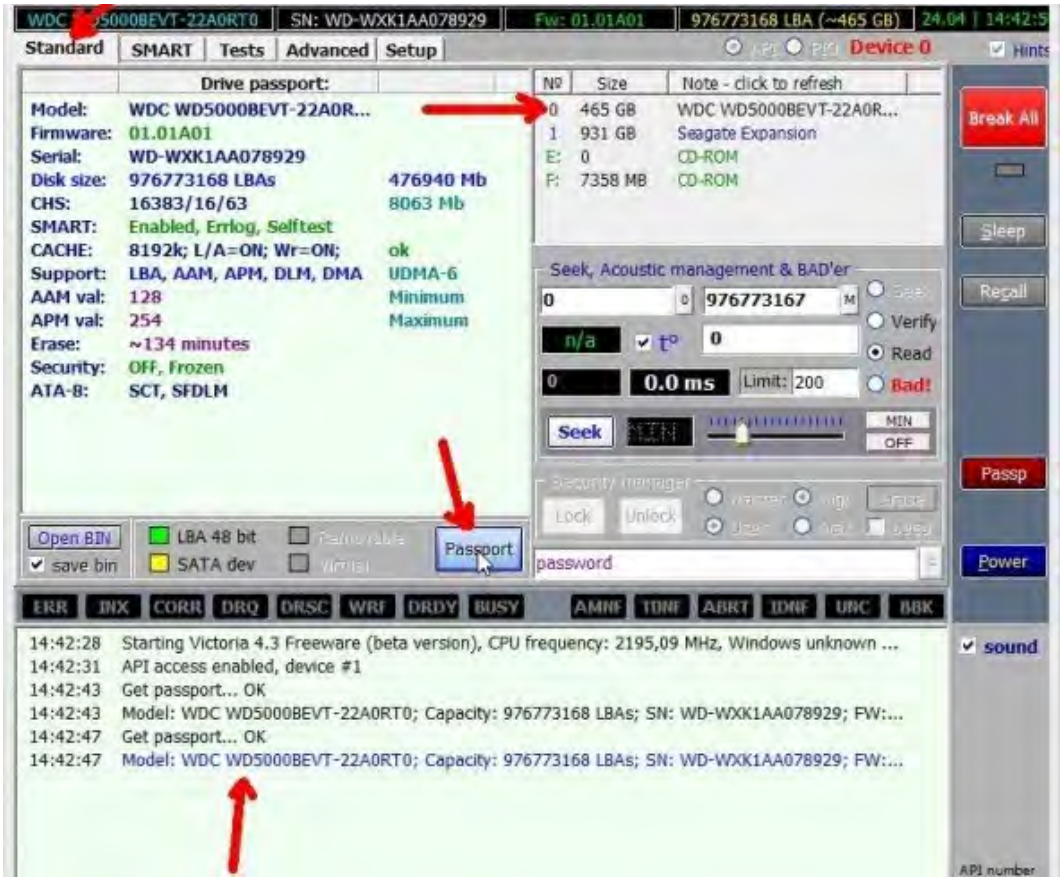


Рис. 1.11. Інтерфейс програми Victoria

Далі необхідно перейти на вкладку «S.M.A.R.T.», на якій натиснути на кнопку «Get SMART». У самому низу вікна

з’явиться повідомлення «SMART Status = GOOD» (рис. 1.12). Абревіатура «S.M.A.R.T.» у перекладі означає «Засоби самодіагностики пристрою для визначення залишкових ресурсів і загального стану».

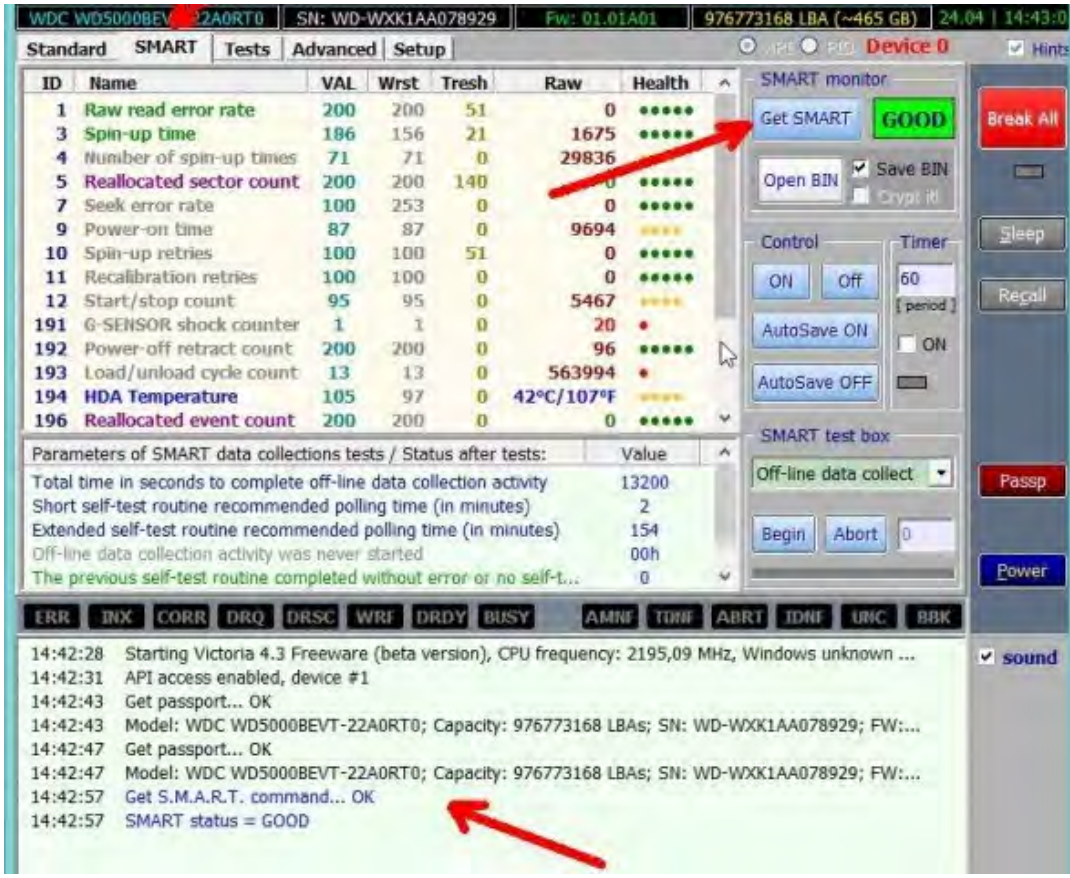


Рис. 1.12. Отримання S.M.A.R.T. параметрів жорсткого диску

Якщо контролер жорсткого диска працює в режимі AHCI (Native SATA), SMART атрибути можна не отримати, з видачею в лог повідомлення «Get S.M.A.R.T. command ... Error reading S.M.A.R.T!». Про неможливість отримання даних S.M.A.R.T також говорить підсвічений червоним напис «Non ATA» при ініціалізації носія, контролер якого не дозволяє використовувати команди ATA-інтерфейсу, в тому числі запит атрибутів S.M.A.R.T.

У цьому випадку потрібно зайти в BIOS і на вкладці Config → Serial ATA (SATA) → SATA Controller Mode Option → змінити з AHCI на Compatibility. Після закінчення тестування в Victoria, налаштування необхідно повернути.

Потім переходять на вкладку «Test» і натискають кнопку «Start». У головному вікні, ліворуч, почнуть показуватися різнокольорові прямокутники (рис. 1.13). Найкращий випадок, якщо вони всі будуть сірими. Увагу потрібно загострювати на червоних і синіх прямокутниках (так звані бед-сектори). Якщо на диску багато синіх прямокутників, рекомендується ще раз пройти перевірку диска, тільки з включеною галочкою «Remap». У цьому випадку Victoria буде приховувати знайдені збійні сектора. Таким чином виконують відновлення жорстких дисків, які почали вести себе не стабільно.

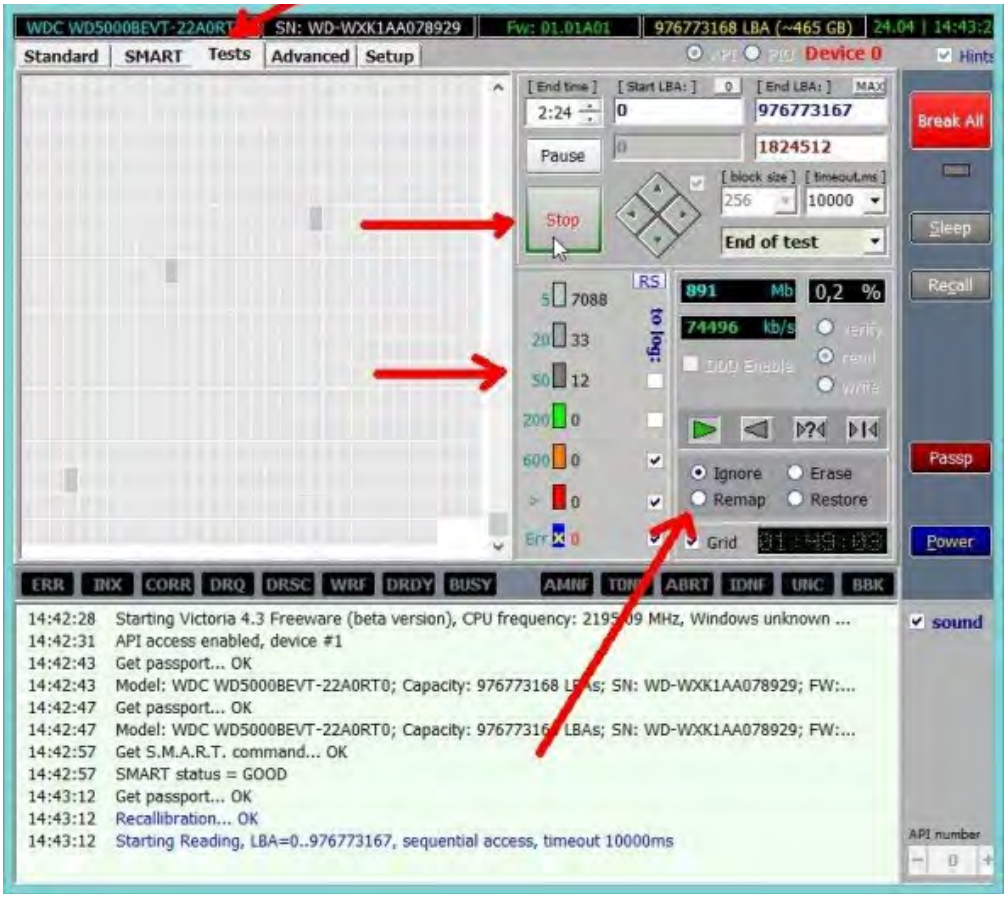


Рис. 1.13. Сканування жорсткого диску

Але після такого відновлення, не завжди жорсткий диск пропрацює довго. Якщо він вже почав «сипатися» то сподіватися на програму не варто. При наявності великої кількості синіх і червоних прямокутників необхідно задуматися про новий жорсткий диск, на якому сині блоки не припустимі.

Сканування жорсткого диску об'ємом 500-750 Гб триває близько 1-2 годин, для перевірки 2-3 Тб знадобиться в 3рази більше часу, що необхідно враховувати при запуску.

Переривати процес сканування, особливо сканування з виправленням вкрай не рекомендується. Це може привести до повного виходу пристрою з ладу.

Перевірка жорсткого диску на працездатність у програмі HDDScan.

HDDScan – утиліта для тестування накопичувачів інформації (HDD, RAID, Flash), призначена для їх діагностики на наявність BAD-блоків, перегляду S.M.A.R.T-атрибутів, зміни спеціальних налаштувань. Основна група пристроїв, які обслуговуються – жорсткі диски з такими інтерфейсами:

- IDE;
- ATA/SATA;
- FireWire или IEEE1394;
- SCSI;
- USB (для роботи існують обмеження).

Для флеш-накопичувачів можливо тільки проведення тестових робіт. Також тести є єдиним видом обстеження RAID-масивів з інтерфейсами ATA / SATA / SCSI.

Фактично програма HDDScan здатна працювати з будь-якими знімними пристроями, що підключаються до комп'ютера, якщо вони мають власне сховище інформації. Необхідно враховувати, що в завдання утиліти HDDScan не входить ремонтно-відновлювальний процес, вона розрахована тільки на діагностику, аналіз і визначення проблемних ділянок жорсткого диска.

Серед можливостей HDDScan виділяють наступні:

- надання детальної інформація про диск;

– тестування поверхні з використанням різних методик;

– перегляд атрибутів S.M.A.R.T.;

– регулювання або зміна параметрів AAM (рівень шуму) або значень APM і PM (розширене управління живленням);

– виведення в панель задач показників температури жорстких дисків для отримання можливості постійного контролю.

HDDScan підтримується сімейством операційних систем Windows від Windows XP до Windows10. Версія програми portable є безкоштовною. Для того, щоб установити HDDScan на своєму комп'ютері необхідно скачати файл HDDScan.exe з офіційного сайту – http://hddscan.com/ і двічі нажати по ньому лівою кнопкою миші для запуску. Далі з'явиться вікно з ліцензією, у якому необхідно погодитися з умовами використання програми, натиснувши «I Agree». При повторному запуску практично відразу ж відкривається головне вікно програми. Весь процес полягає у визначенні пристроїв, з якими програма буде працювати. Користувач має можливість запускати HDDScan на будь-яких пристроях або зі змінних носіїв без прав адміністратора.

Головне вікно HDDScan представлено на рис. 1.14. У верхній частині знаходиться поле з найменуванням носія інформації. У ньому можна переглянути список всіх носіїв, підключених до материнської плати. Нижче знаходяться три кнопки виклику базових функцій:

Рис. 1.14. Головне вікно HDDScan

S.M.A.R.T. General Health Information. Натискання на цю кнопку викликає вікно самодіагностики, в якому відображені всі параметри роботи жорсткого диска або іншого носія;

TESTS Read and Wright Tests. Запуск процедури тестування поверхні жорсткого диска. Пропонується 4 режими тестування, Verify, Read, Butterfly, Erase. Вони виконують різні види перевірки – від перевірки швидкості читання до визначення збійних секторів. Вибір того чи іншого варіанту викличе появу діалогового вікна і запуску процесу тестування;

TOOLS Information and Features. Виклик елементів управління або призначення потрібної функції. Доступні 5 інструментів, DRIVE ID (ідентифікаційні дані диску), FEATURES (вікно управління ATA або SCSI), SMART TESTS (вибір одного з трьох варіантів тесту), TEMP MON (відображення поточної температури носія), COMMAND (командний рядок).

Перевірку носія починають з вивчення звіту S.M.A.R.T (рис. 1.15).

	Num	Attribute Name	Value	Worst	Raw(hex)	Threshold
	001	Raw Read Error Rate	200	200	0000000000-0000	051
	003	Spin Up Time	142	139	0000000000-0F33	021
	004	Start/Stop Count	098	098	0000000000-08DE	000
	005	Reallocation Sector Count	200	200	0000000000-0000	140
	007	Seek Error Rate	200	200	0000000000-0000	000
	009	Power-On Hours Count	061	061	0000000000-6FB1	000
	010	Spin Retry Count	100	100	0000000000-0000	000
	011	Recalibration Retries	100	100	0000000000-0000	000
	012	Device Power Cycle Count	098	098	0000000000-08C9	000
	192	Emergency Retract Count	200	200	0000000000-007B	000
	193	Load/unload Cycle Count	200	200	0000000000-0862	000
	194	HDA Temperature	105	094	38 C	000

Рис. 1.15. Фрагмент звіту S.M.A.R.T. Report

Усі позиції, що не викликають проблем, відзначені зеленим кольоровим індикатором. Можливі неполадки або

невеликі вади відзначаються жовтим трикутником зі знаком оклику. Серйозні проблеми відзначаються червоним кольором.

Далі переходять до вибору тесту (рис. 1.16).

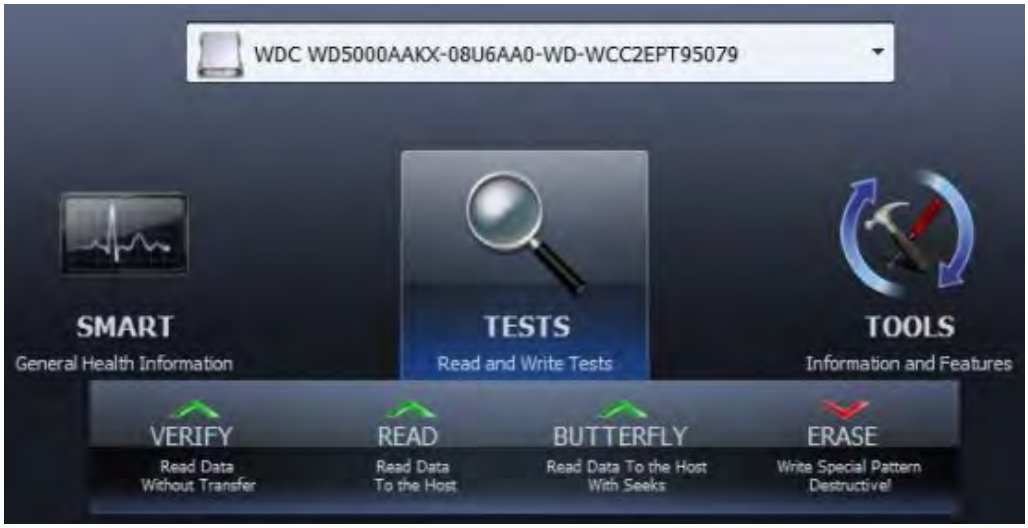


Рис. 1.16. Вибір типу тесту

Тестування в HDDScan, як і в Victoria – тривалий процес, що вимагає певного часу. Теоретично можна проводити одночасно кілька тестів, але на практиці цього робити не рекомендується. Програма не дає стійкого і якісного результату в таких умовах, тому, при необхідності виконати кілька типів тестування, краще витратити час і виконати їх по черзі. Доступні такі типи тестів:

Verify. Проводиться перевірка чистої швидкості читання інформації, без передачі даних по інтерфейсу.

Read. Перевірка швидкості читання з передачею даних по інтерфейсу.

Butterfly. Перевірка швидкості читання з передачею по інтерфейсу, виконувана в специфічній послідовності: перший блок-останній-другий-передостанній-третій ... і т. д.

Erase. Проводиться запис на диск спеціального тестового інформаційного блоку. Перевіряється якість запису, читання, визначається швидкість обробки даних. Інформація на цій ділянці диска буде загублена.

При виборі типу тесту з’являється вікно (рис. 1.17), в якому вказується:

- номер першого сектора для перевірки;
- кількість блоків для тестування;
- розмір одного блоку (кількість секторів LBA, що містяться в одному блоці).

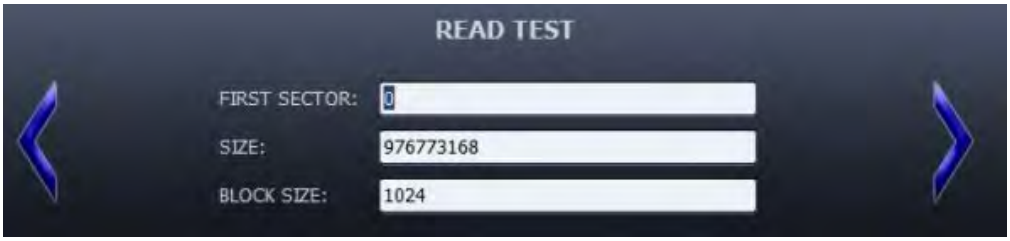


Рис. 1.17. Вікно параметрів тесту

При натисканні кнопки «Вправо» тест додається в чергу завдань. У вікні диспетчера задач з’являється рядок з поточною інформацією про проходження тесту. Натискання по ньому викликає меню з інформацією про деталі процесу, можливості поставити на паузу, зупинити або повністю видалити завдання. Подвійне натискання по рядку викличе вікно з докладною інформацією про проходження тесту в реальному часі з візуальним відображенням процесу.

Вікно має три варіанти візуалізації: у вигляді графіка, карти або блоку числових даних. Така кількість варіантів дозволяє отримати максимально детальну і зрозумілу користувачеві інформацію про хід процесу.

Приклад відображення працездатності секторів накопичувача у вигляді карти наведений на рис. 1.18. На рис. 1.19 наведена карта непрацездатного накопичувача.

У розділі TOOLS стає доступним меню інструментів. Також можна отримати інформацію про фізичні або логічні параметри диска, при натисканні на DRIVE ID.

Розділ FEATURES дає можливість змінити наступні параметри носія (крім пристроїв USB):

– знизити рівень шуму (функція AAM, доступна не на всіх видах дисків);

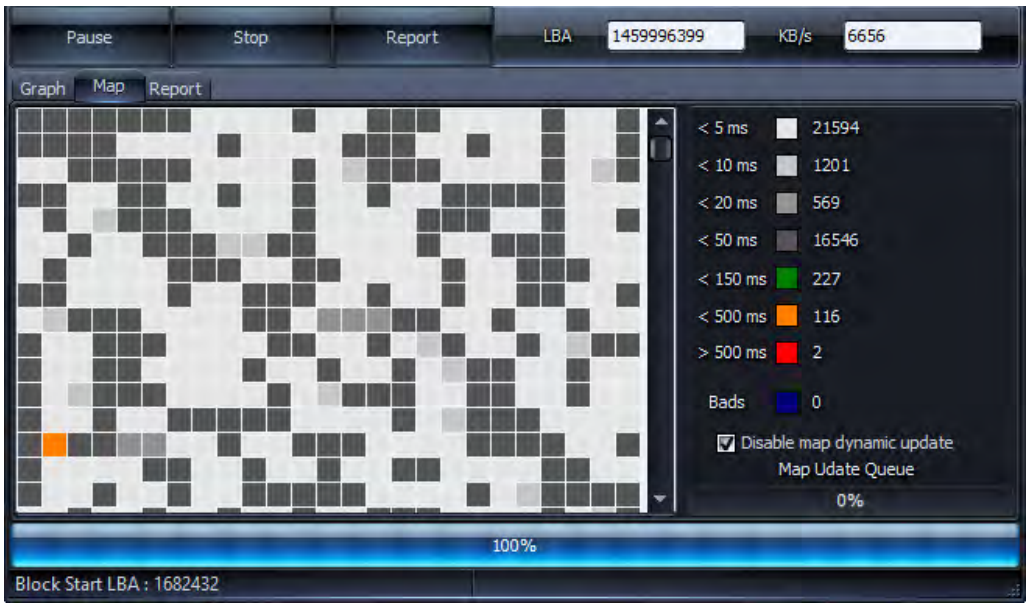


Рис. 1.18. Тестування на bad-сектори за допомогою HDD-scan(нормальний стан накопичувача)

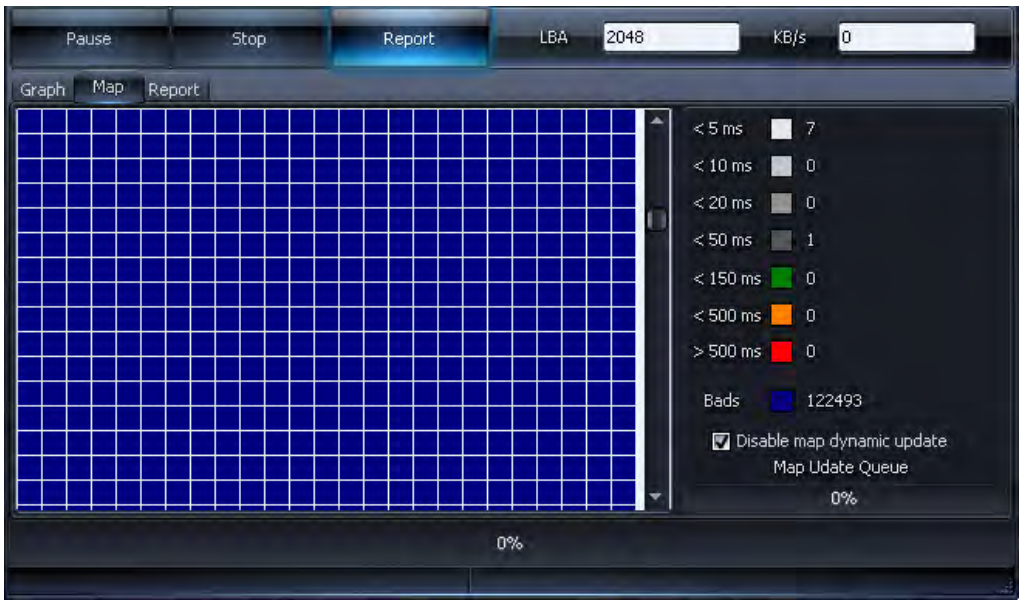


Рис. 1.19. Представлення накопичувача з пошкодженою магнітною поверхнею

– відрегулювати режими обертання шпинделя, що забезпечують економію енергії і ресурсу. Налаштовується

швидкість обертання, аж до повної зупинки під час простою (функція APM);

– задіяти таймер відстрочки зупинки шпинделя (функція PM). Шпиндель буде автоматично зупинятися після закінчення заданого часу, якщо диск не використовується в даний момент;

– можливість моментального запуску шпинделя на вимогу виконуваної програми.

Екран зміни налаштувань наведений на рис. 1.20. Для дисків з інтерфейсом SCSI/SAS/FC доступна опція відображення виявлених дефектів логіки або фізичних вад, а також запуск і зупинка шпинделя.

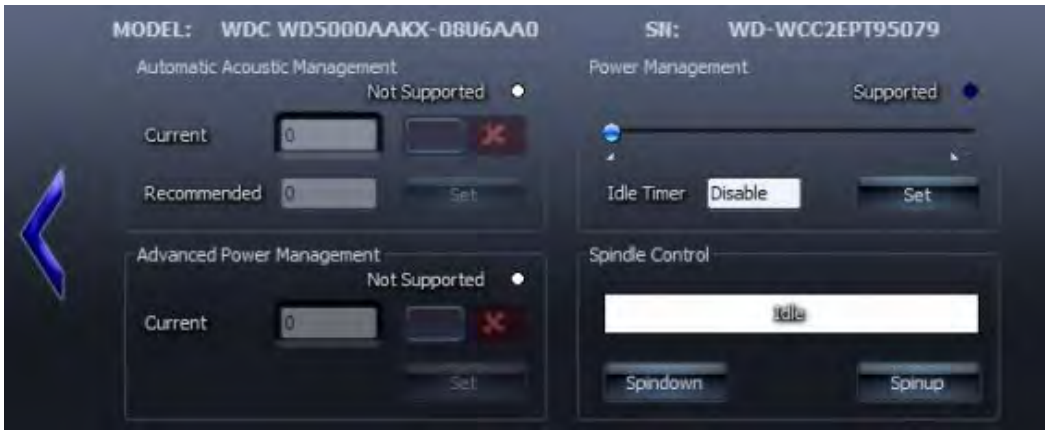


Рис. 1.20. Налаштування у розділі FEATURES

У розділі SMART TESTS доступні 3 варіанти тестів (рис. 1.21):

– Короткий. Триває 1-2 хвилини, здійснюється перевірка поверхні диска і проводиться швидкий тест проблемних секторів.

– Розширений. Триває близько 2 годин. Обстежуються вузли носія, виконується перевірка поверхні.

– Транспортування. Триває кілька хвилин, проводиться обстеження електроніки приводу і виявлення проблемних ділянок.

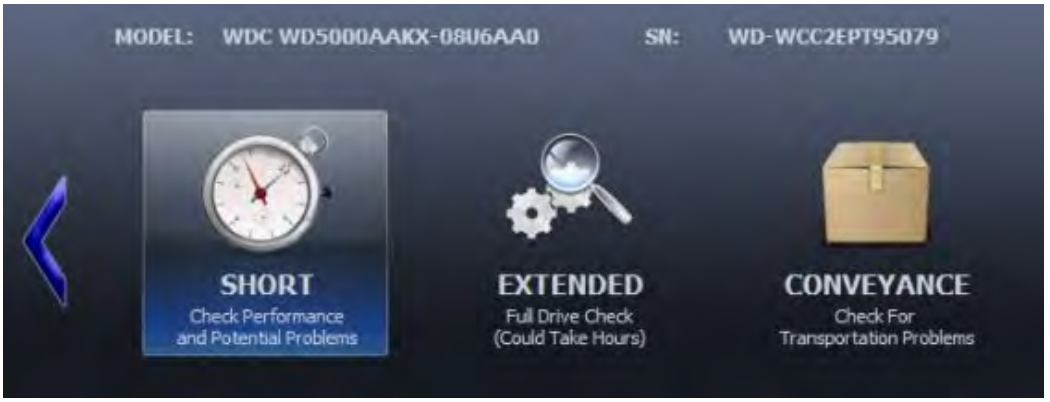


Рис. 1.21. Варіанти SMART TESTS

Функція TEMP MON дає можливість визначити ступінь нагріву диска в поточний момент часу. Функція TEMP MON дає можливість визначити ступінь нагріву диска в поточний момент часу.

Проведення повної перевірки за всіма позиціями не є завданням користувача. Зазвичай обстежуються певні параметри або функції диска, що потребують постійного спостереження.

Найбільш важливими показниками можна вважати загальний діагностичний звіт, що дає докладну інформацію про існування і кількість проблемних секторів, а також тестові перевірки, що демонструють стан поверхні під час роботи пристрою.

1.2.2. Завдання для самостійного виконання

1. Завантажити і встановити програму Victoria на ПК.
2. Виконати тестування флеш-накопичувача і жорсткого диска на bad-сектори в програмі Victoria. При необхідності виконати оновлення диску і/або накопичувача.
3. Задокументувати результати перевірки у формі скріншота мапи секторів накопичувача і дати оцінку ступеня його зношеності.
4. Завантажити і встановити програму HDD-scan на ПК. Виконати тестування флеш-накопичувача і жорсткого диска на bad-сектори.

5. Задокументувати результати перевірки у формі скріншота мапи секторів накопичувача.
6. Порівняти отримані у програмах результати і зробити висновки.

1.2.3. Запитання для самоперевірки

1. Поясніть у чому полягає процес перевірки секторів накопичувача на працездатність? Як визначаються робочі і неробочі сектори?
2. Назвіть причини, з яких сектори накопичувачів стають непрацездатними.
3. Як виконується відновлення жорстких дисків, які почали вести себе не стабільно в програмі Victoria?
4. Поясніть необхідність прав адміністратора для запуску програми Victoria.
5. Що таке атрибути S.M.A.R.T. і як їх визначити в розглянутих програмах?
6. Назвіть причини через які інколи неможливо отримати дані S.M.A.R.T. Чи можна їх усунути?
7. Чому не можна переривати процес перевірки накопичувача на працездатність?
8. Яка з розглянутих програм буде більш точно виконувати процедуру діагностики? Відповідь обґрунтуйте.
9. Назвіть типи тестів накопичувачів, які можна провести в HDD-scan. Поясніть їх особливості.
10. Назвіть додаткові корисні функції, які підтримують програми Victoria і HDD-scan.

Тема 1.3. Основи роботи з Active Disk Editor

Мета: Познайомитися з інтерфейсом системного програмного забезпечення Active@Disk Editor і вивчити особливості його роботи.

1.3.1. Теоретичні відомості

Active Disk Editor – це системне програмне забезпечення від фірми LSoft Technologies Inc., призначене для перегляду і редагування вихідних даних фізичних дисків і томів, а також вмісту файлів будь-якого типу.

Завантажити Active Disk Editor можна безкоштовно з офіційного сайту www.disk-editor.org, на якому присутні версії для операційних систем сімейства Windows і Linux. Процес встановлення Active Disk Editor на комп'ютер користувача відбувається за допомогою покрокового менеджера, і є простим і інтуїтивно зрозумілим.

Після встановлення і запуску програми з'явиться екран привітання з наступними варіантами подальших дій:

- відкрити диск або том;
- відкрити файл;
- відкрити образ диска;
- продовжити перегляд локальних дисків та сховищ даних.

Почати перегляд локальних пристроїв, томів та файлів можна після закриття екрану привітання, натиснувши на кнопку «Перегляд файлів». Найпростіший спосіб відкрити об'єкти для редагування – це вибрати диск, том чи файл у браузері файлів та використовувати команду «Відкрити в редакторі дисків» на панелі інструментів або в контекстному меню (рис. 1.22).

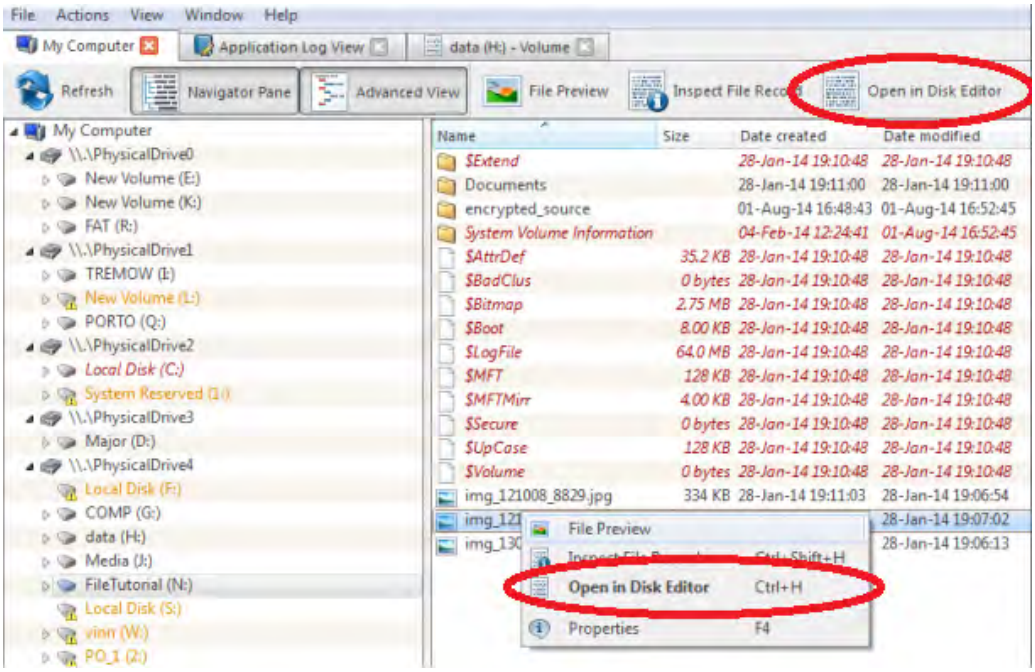


Рис. 1.22. Відкриття файлу за допомогою файлового браузеру

Active@Disk Editor може автоматично виявляти підключені пристрої та показувати їх у файловому браузері. Однак якщо підключений пристрій не з'являється при натисканні кнопки «Оновити» («Refresh») на панелі інструментів, необхідно натиснути і утримувати кнопку Ctrl на клавіатурі та одночасно натиснути кнопку «Оновити»» («Refresh») на панелі інструментів. Це дозволить повністю переглянути і оновити всі підключені локальні сховища даних.

Для того, щоб відкрити диск або том для редагування, використовують команду головного меню: Файл → Відкрити диск (File → Open Disk) або натискають кнопку «Відкрити диск» («Open Disk») на екрані привітання після запуску програми. Діалогове вікно відкриття диску наведено на рис. 1.23.

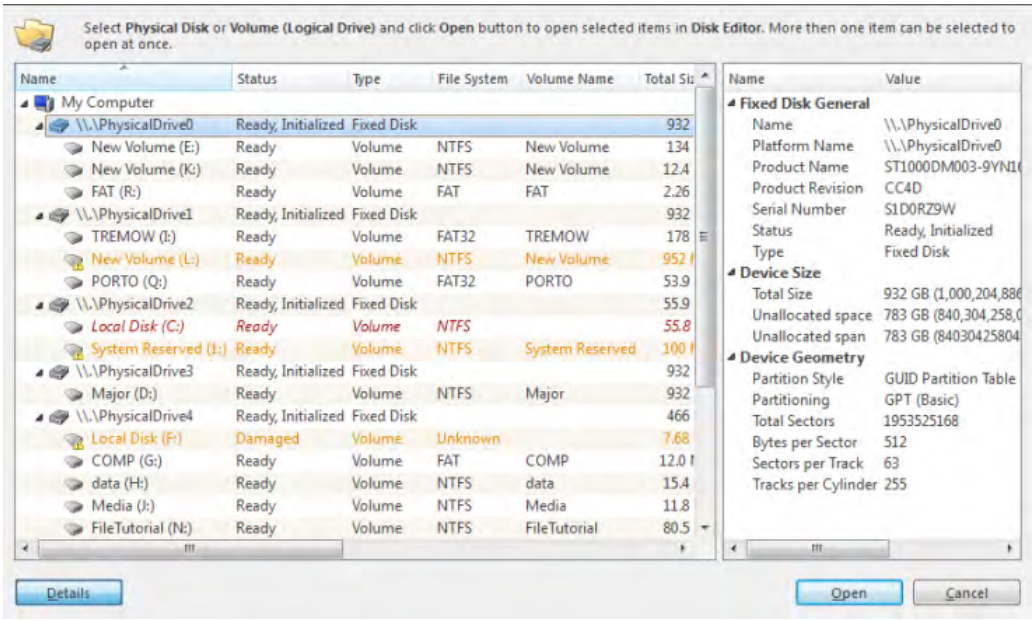


Рис. 1.23. Діалогове вікно відкриття диску в Active@Disk Editor

Для того, щоб відкрити файл для редагування, використовують команду головного меню: Файл → Відкрити файл або натискають кнопку «Відкрити файл» на екрані привітання після запуску програми. У діалоговому вікні «Відкрити файл» («Open File») необхідно знайти файл та натиснути кнопку «Відкрити» («Open»), щоб відкрити файл у дисковому редакторі. Діалогове вікно відкриття файлу наведено на рис. 1.24.

Active@Disk Editor дозволяє редагувати вміст вибраної частини відкритого об’єкта. За замовчуванням Active@Disk Editor показує вміст об’єкта в режимі «Читання», що запобігає випадковим змінам. У режимі «Редагування» можна змінити вміст відкритого файлу чи диска з подальшим збереженням усіх модифікацій в пам'яті. Збереження відбувається після натискання на кнопку «Зберегти» («Save»).

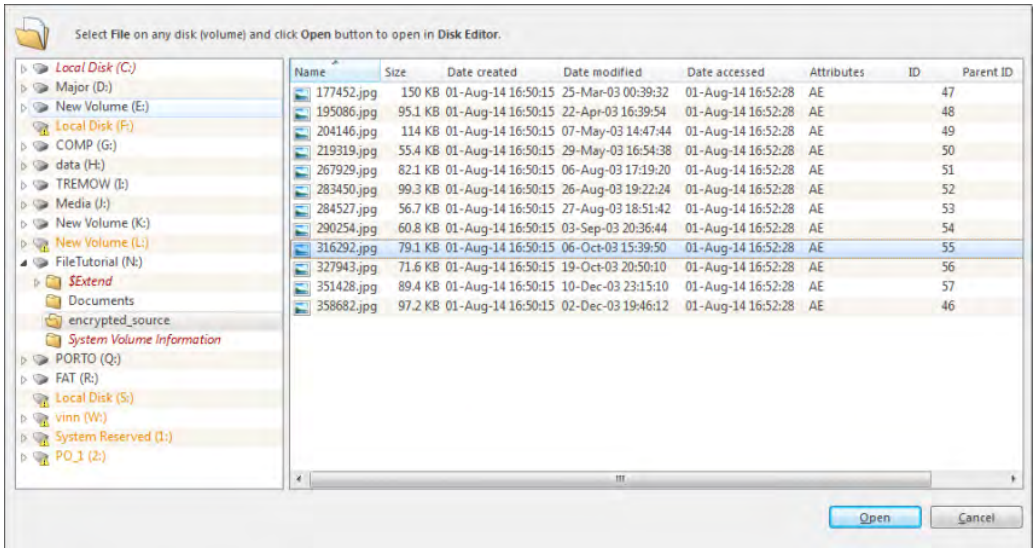


Рис. 1.24. Діалогове вікно відкриття файлу в Active@Disk Editor

Для переходу між режимами «Читання» та «Редагування» необхідно виконати одну з наступних дій:

– на панелі інструментів «Редактор дисків» вибирають «Правка» → «Дозволити редагування вмісту» (Edit → Allow Edit).

– правою кнопкою миші в області редагування вибирають «Дозволити редагування вмісту» («Allow Edit») в контекстному меню.

При копіюванні вибраного тексту з області редагування у буфер обміну можна зберегти його там, в одному з двох форматів, використовуючи наступні команди:

– «Копіювати» («Copy») – вибрані дані копіюються в буфер обміну як двійкові.

– «Копіювати відформатованим» («Copy Formatted») – вибрані дані копіюються у відформатованому тексті, придатному для вставки в текстовий редактор.

Після відкриття об’єкту за допомогою «Редактора дисків» («Disk Editor»), можна переглядати/змінювати його код, за рахунок переміщення від одного блоку до іншого або безпосереднього переходу до певної адреси. Так само можна переходити до системних записів на диску, таких як

завантажувальний сектор (первинний та копія) або таблиця розділів.

За допомогою кнопки «Навігація» («Navigate») на панелі інструментів можна перейти до певної області відкритого об’єкта. Виділення, які відображаються, залежать від типу об'єкта, який редагується. Незалежно від того, який об’єкт відкрито для редагування, першими двома пунктами в «Навігаційному меню» («Navigate») будуть «Перейти до зміщення» («Go to Offset») та «Перейти до сектору» («Go to Sector»).

Виклик команди «Перейти до зміщення» («Go to Offset») відкриває діалогове вікно, яке дозволяє уточнити значення зміщення на диску для переходу. У вікні можна використовувати як десяткові, так і шістнадцяткові значення. На початку шістнадцяткового значення необхідно вказувати 0x. Наприклад, щоб вказати місце 512 як шістнадцяткове число вводять 0x200.

Також можна вказати зміщення від початку нумерації, поточного положення або з кінця. Поруч із полем «Зміщення» («Offset») є дві мітки із зазначенням мінімальної та максимальної дозволеної величини, відображених у вигляді десяткових чисел (рис. 1.25). Викликати це діалогове вікно можна безпосередньо за допомогою сполучення клавіш Ctrl + Shift + G.

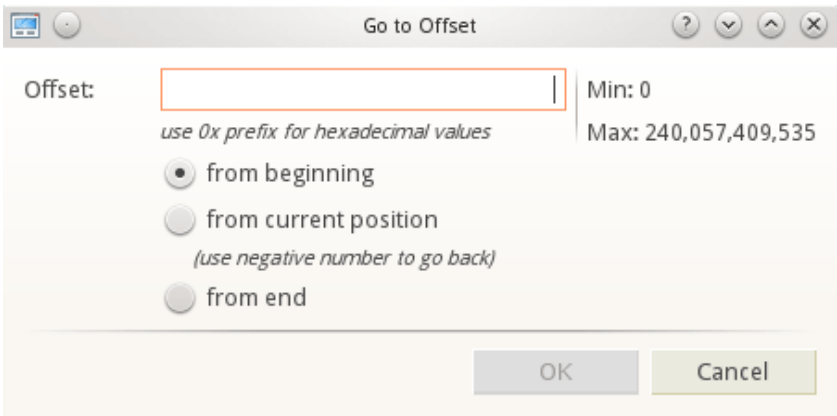


Рис. 1.25. Діалогове вікно «Перейти до зміщення» («Go to Offset»)

Команда «Перейти до сектора» («Go to Sector») дозволяє переходити до початку визначеного сектора або кластеру. У діалоговому вікні переходу (рис. 1.26) є два поля редагування, які дозволяють ввести як номер сектора, так і кластера.

Поле редагування кластера доступне лише для логічних дисків. Як і у діалоговому вікні зміщення, також можна використовувати десяткові і шістнадцяткові числа. Поруч із полем редагування в дужках знаходиться діапазон дозволених значень. Необхідно зауважити, що не всі сектори відповідають кластерам, але кожен кластер відповідає певному сектору.

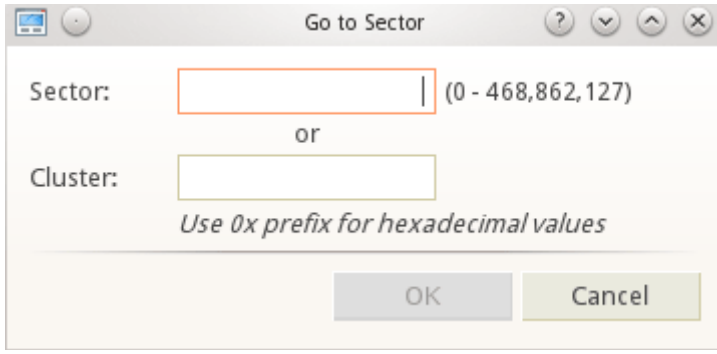


Рис. 1.26. Діалогове вікно «Перейти до сектора» («Go to Sector»)

Залежно від того, яке поле активне, у діалоговому вікні буде використаний сектор або кластер. Якщо ввести номер у полі редагування кластера, відповідний сектор відображається автоматично. Відкрити це діалогове вікно можна також за допомогою сполучення клавіш Ctrl + G.

Під час переходу до точки доступу через меню навігації або до певного зміщення або сектору, ці адреси зберігаються у стеку. Це дозволяє переміщатися назад і вперед до попередніх місць, використовуючи команди «Назад» («Back») і «Вперед» («Forward»), розташовані на панелі інструментів редактора дисків.

Перехід до системних записів фізичного диска здійснюється натисканням на кнопку «Навігація» («Navigate»)

на панелі інструментів. Залежно від файлової системи розділів та вмісту фізичного диска, меню навігації міститиме різні області.

Логічна структура дисків з файловою системою FAT і FAT32 складається з таких областей:

- Завантажувальний сектор (Boot Sector).
- Копія завантажувального сектору (Boot Sector Copy) (тільки у FAT32).
- FAT1.
- FAT2.
- Кореневий каталог (Root Directory).

Логічна структура дисків з файловою системою NTFS складається з таких областей:

- Завантажувальний сектор (Boot Sector).
- Копія завантажувального сектору (Boot Sector Copy).
- $MFT.
- $MFT Mirror.
- Довільний запис MFT.

Логічна структура дисків з файловою системою HFS+ складається з таких областей:

- Заголовок тому (Volume Header)
- Копія заголовку тому (Volume Header Copy)

Логічна структура дисків з файловою системою Ext2/Ext3 складається з таких областей: суперблок (Superblock).

При переході до вказаних логічних областей до них застосовується відповідний структурний шаблон. Наприклад, якщо обрана область – завантажувальний сектор, то до зміщення завантажувального сектора буде застосований шаблон структури завантажувального сектора.

Під час редагування тому також можна переходити до записів файлів. Щоб активувати цю функцію, необхідно натиснути на кнопку «Переглянути записи файлів» («Browse File Entries») на панелі інструментів (рис. 1.27).

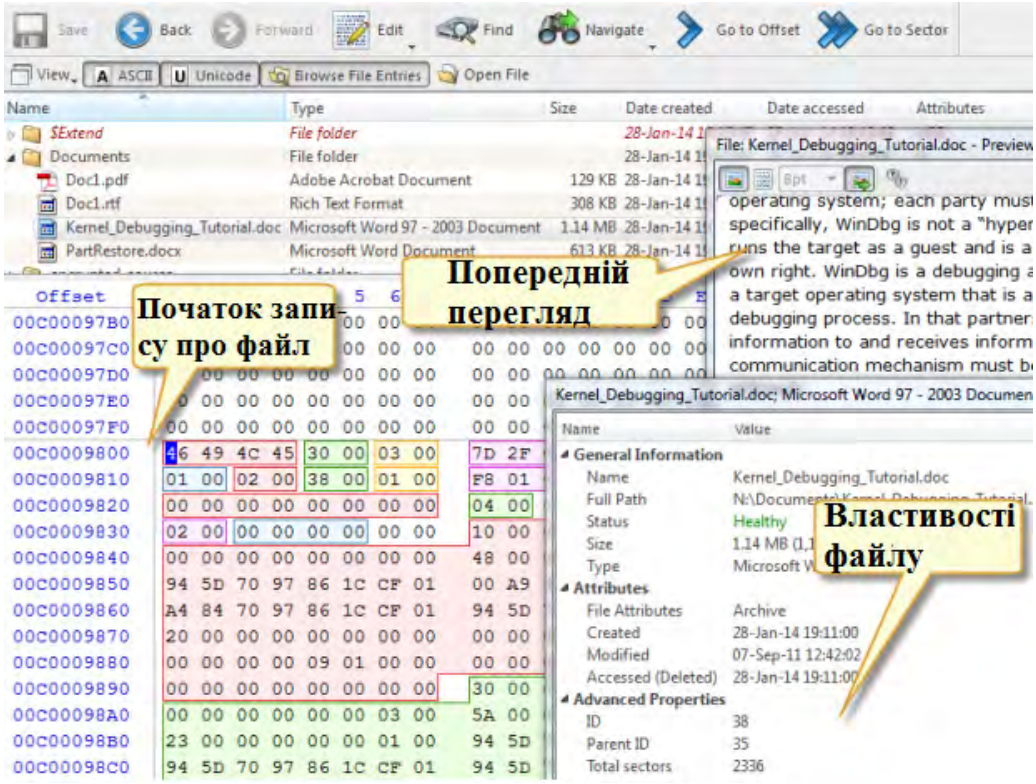


Рис. 1.27. Перегляд записів файлів

Вибравши файл або папку в браузері файлів, Active@Disk Editor автоматично виконає перехід до початку запису про файл. Якщо файл розпізнано, то його можна переглянути на панелі попереднього перегляду файлів, а на панелі властивостей відобразяться найпоширеніші атрибути та властивості файлу.

Для того, щоб відкрити вибраний файл в окремому вікні редактора, натискають кнопку «Відкрити файл» («Open File») на панелі інструментів або двічі натискають на вибраному файлі.

Для того щоб вибрати дані в області редагування, натискають ліву кнопку миші і утримують її, виділяючи необхідну область. Якщо область більша, ніж первинна область редактора, то мишу необхідно перетягнути вниз.

Альтернативний спосіб виділення – це визначення початку та кінця блоку. Цей спосіб може бути зручнішим, коли

потрібно вибрати велику область. Курсор переміщують у положення, з якого потрібно розпочати вибір, і виконують одну з наступних дій:

– Обрати команду меню Правка → Початок блоку (Edit → Beginning of block) з меню «Правка» на панелі інструментів.

– Натиснути правою кнопкою миші та обрати Правка → Початок блоку (Edit → Beginning of block) у контекстному меню.

– Натиснути Ctrl + 1.

Для копіювання області даних її виділяють одним з вказаних способів і командою Правка → Копіювати або натисканням сполучення клавіш Ctrl + C копіюють необхідне. Вибрана область буде скопійована у буфер обміну у двійковому форматі. Якщо згодом необхідно вставити код в текстовий редактор, використовують команду «Копіювати відформатованим» («Copy Formatted»). Зверніть увагу, що у буфер обміну можна скопіювати максимум 1 МБ даних.

Дані, які були скопійовані в буфер обміну, можна вставити в інше місце, перемістивши курсор у положення, куди потрібно їх скопіювати. Для цього необхідно скористатися командою Правка → Вставити або натиснути Ctrl + V.

Якщо скопіювати текст у буфер обміну в текстовому редакторі, він буде вставлений у редактор дисків як текст. В іншому випадку дані будуть скопійовані як двійкові файли.

Виділену область також можна заповнити довільним текстом або двійковими даними. Спершу необхідно виділити область, потім обрати в контекстному меню команду Редагувати → Заповнити блок.

Діалогове вікно «Блок заповнення» (рис. 1.28) дозволяє вводити текстові або шістнадцяткові шаблони значень, які будуть використані для заповнення області. Наприклад, якщо потрібно заповнити область 0 байтами, просто вводять 00 у поле редагування значень шістнадцяткових значень. Якщо необхідно заповнити область шаблоном «Видалено», слово

вводять як текст, який буде повторюватися стільки разів, скільки потрібно, щоб заповнити блок.

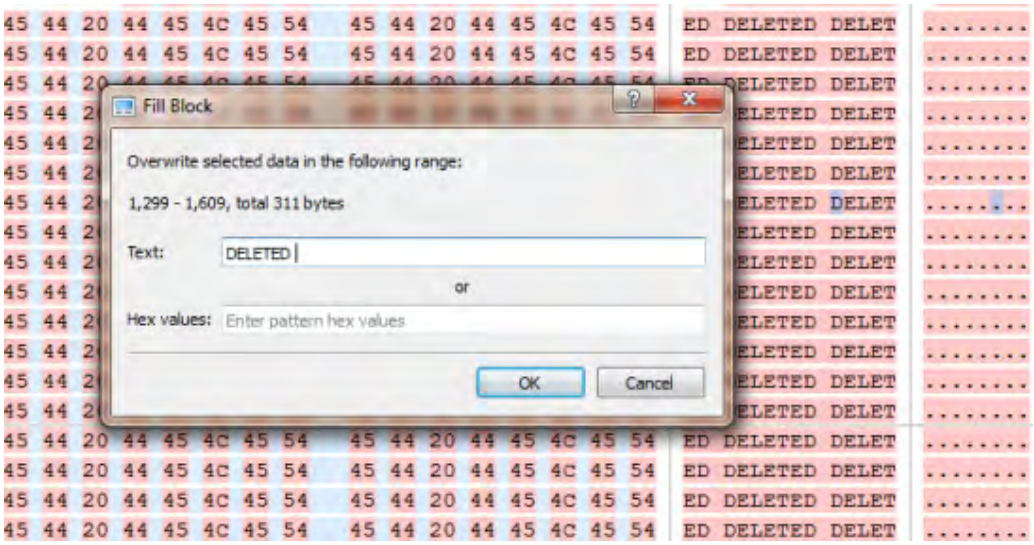


Рис. 1.28. Діалогове вікно «Блок заповнення»

Інструмент «Інспектор даних» («Data Inspector») у невеликому вікні перегляду дозволяє переглядати тип вибраних даних (рис. 1.29). Це може допомогти інтерпретувати дані які відображені в області редагування редактора.

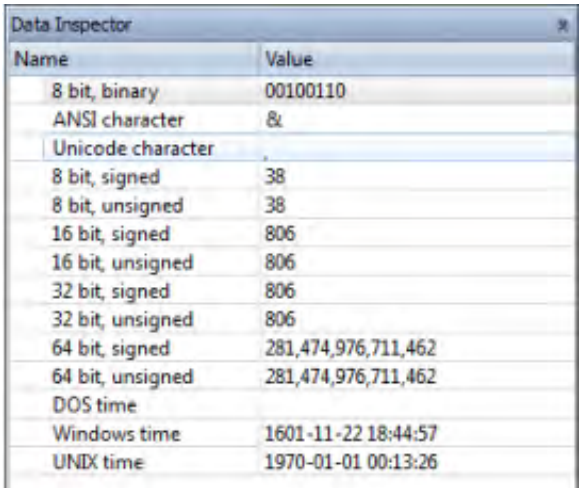


Рис. 1.29. Вікно «Інспектора даних» («Data Inspector»)

Для того, щоб відкрити «Інспектор даних», на панелі інструментів «Редактор дисків» обирають «Перегляд» → «Інспектор даних».

Копіювання інтерпретованих даних з «Інспектора даних» у вигляді тексту виконують так:

1. Натискають правою кнопкою миші будь-де у вікні «Інспектор даних».

2. Обирають команду «Копіювати».

Для переключення між прямим і зворотнім порядком байт даних необхідно:

1. Натискають правою кнопкою миші будь-де у вікні «Інспектор даних».

2. Обирають команду «Прямий порядок» («Big Endian»).

Вікно «Інспектор даних» можна приєднати, і його розташування можна змінити, натиснувши на назву вікна та перетягнувши його на нове місце. Якщо вікно «Інспектора даних» розділяє простір з іншими видами інструментів, його відносне положення можна змінити, клацнувши лівою кнопкою миші та перетягнувши вкладку вікна. Закрити вікно можна натиснувши кнопку [X] у верхньому правому куті вікна та знову відкрити його за допомогою меню «Вид» на панелі інструментів редактора дисків.

У програмі Active@Disk Editor передбачено можливість попереднього перегляду файлів з редагуванням їх вмісту та вивчення вхідних записів тому. Приклад попереднього перегляду графічного файлу наведений на рис. 1.30.

Для того, щоб відкрити панель попереднього перегляду файлів виконують одну з наступних дій:

– Двічі натискають на файл зображення.

– Правою кнопкою миші із контекстного меню файлу зображення обирають «Попередній перегляд файлу» («File Preview»).

– Обирають файл зображення та натискають «Попередній перегляд файлу» («File Preview») на головній панелі інструментів.

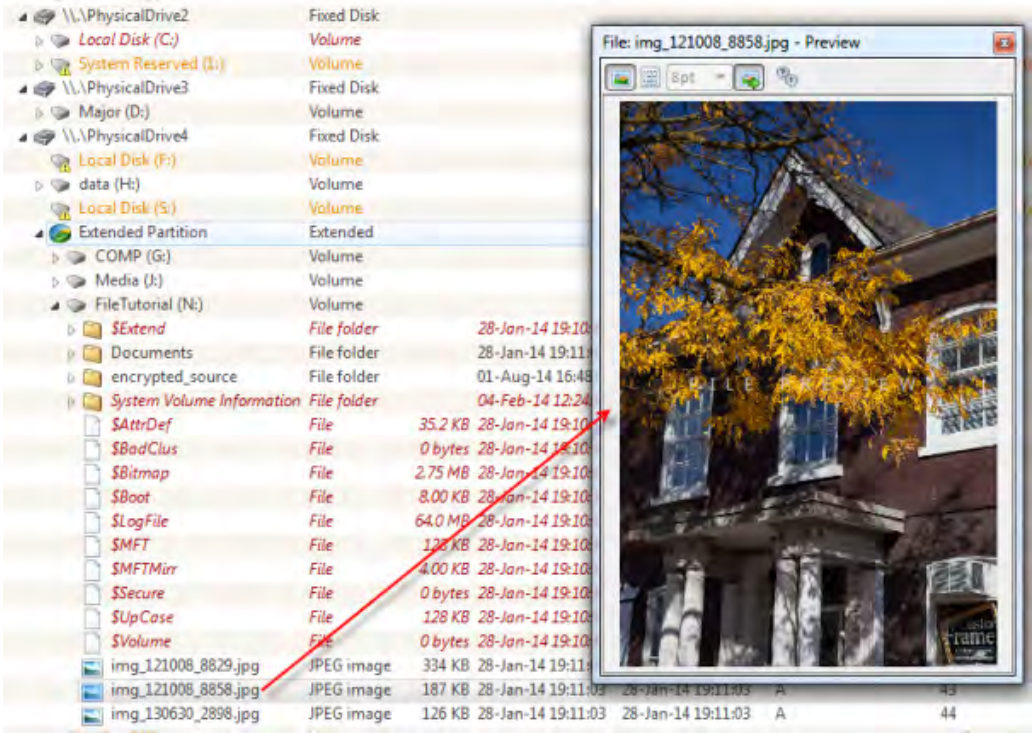


Рис. 1.30. Попередній перегляд файлу в Active@Disk Editor

За допомогою панелі інструментів на панелі попереднього перегляду файлів можна перемикатися між режимами попереднього перегляду:

Режими попереднього перегляду:

1. За замовчуванням. Файл буде переглянуто згідно з даними про його формат.

2. Шістнадцятковий режим. Лише перший сектор буде показаний у шістнадцятковому режимі.

3. Автоматичне налаштування. За допомогою цієї опції файли, вибрані в контекстному джерелі, будуть попередньо переглянуті автоматично. Цю опцію можна вимкнути, якщо з будь-якої причини попередній перегляд файлу викликає затримки в навігації по файлу.

Якщо файл попереднього перегляду недоступний, він відображається в шістнадцятковому або текстовому режимі.

Інструмент «Закладки» («Bookmarks») дозволяє зберегти поточне місце розташування курсору та швидко повернутися до

нього пізніше. Закладці можна дати ім’я, щоб полегшити орієнтацію. Якщо вікно «Закладки» закрите, відкрити його можна за допомогою меню «Перегляд» → «Закладки» (View → Bookmarks). Приклад закладки наведено на рис. 1.31.

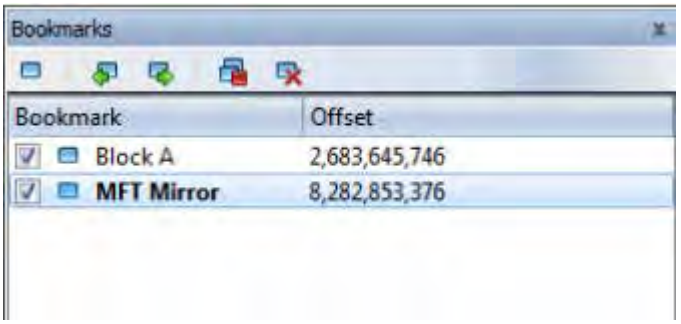


Рис. 1.31. Приклади закладок

Для того, щоб додати закладку натискають Ctrl + F2. Крім того, можна натиснути правою кнопкою миші у шістнадцятковому редакторі і вибрати команду з контекстного меню. Позиція закладок відображається світло-блакитним полем, а також додається до списку закладок у вікні «Закладки».

Для видалення закладки натискають Ctrl + F2, після виділення її курсором миші. Також можна видалити закладку з вікна «Закладки», вибравши закладку на панелі інструментів та натиснувши «Видалити» («Delete»). Функція видалення також може бути обрана з контекстного меню.

Натискання на клавішу F2 реалізує перехід до наступної включеної закладки у списку. Також можна натиснути правою кнопкою миші на закладку, і в контекстному меню вибрати команду «Наступна закладка». Інший варіант – двічі натиснути на ім’я закладки у вікні «Закладки».

Закладки отримують ім’я автоматично при створенні. Можна перейменувати закладку у вікні «Закладки», щоб дати їй значуще ім’я. Для цього один раз натискають по назві закладки та редагують її. Потім необхідно натиснути Enter, щоб прийняти зміни, або Esc, щоб скасувати редагування та повернутися до оригінального імені. Також можна

перейменувати закладку правою кнопкою миші та обрати команду «Перейменувати» («Rename») з контекстного меню.

Іноді замість видалення закладки корисно тимчасово відключити її. Відключена закладка не буде зарахована при переході до наступної закладки. Для того, щоб відключити закладку, з неї знімають прапорець у вікні «Закладки». Для того, щоб одразу вимкнути всі закладки, натискають «Відключити всі закладки» («Disable») на панелі інструментів або обирають цю команду в контекстному меню.

Пошук тексту чи послідовності байт у редакторі дисків виконують за допомогою інструменту «Пошук». Для його виклику використовують комбінацію клавіш Ctrl + F або натискають на кнопку «Знайти» на панелі інструментів «Редактор дисків». Після чого з'явиться діалогове вікно «Пошук», наведене на рис. 1.32.

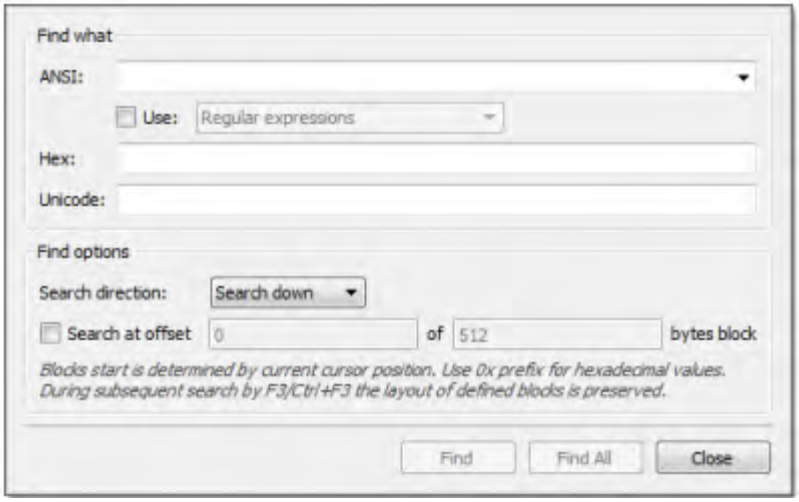


Рис. 1.32. Вікно пошуку в Active@Disk Editor

Пошук даних виконується за ANSI, Hex або Unicode шаблоном. Для прискорення процесу можна задати пошук лише за заданим зміщенням всередині визначених блоків. Регулярні вирази та підстановка ще більше розширюють можливості пошуку. Напрямок пошуку визначатиме напрямок пошуку з поточного положення курсору. При використанні

команди «Знайти всі» («Find All») з'явиться список усіх записів пошуку. Використовують цей список для переходу між записами результатів пошуку (якщо такі є), двічі натиснувши по рядку введення. Приклади використання регулярних виразів:

^ \ d \ d? $ – відповідні цілі числа 0 до 99.

^ \ S + $ – відповідні рядки без пробілу.

\ b (пошта | лист | листування) \ b – рядки відповідності, що містять 'mail' або 'letter' або 'листування', але відповідають лише цілим словам, тобто не 'email'.

& (?! amp;) - відповідні амперсанди, але не &.

\ b (Eric|Eirik) \ b – відповідає Еріку чи Ейріку.

* .html – використання підстановки.

Active @ Disk Editor також можна запустити, використовуючи інструмент «Командний рядок», щоб відкрити деякі елементи зберігання даних за замовчуванням. Приклади команд розглянуті далі.

disk=[disk] or d=[disk] – відкриває вказаний фізичний диск;

diskprefix=[prefix] dp=[prefix] – відкриває фізичний диск із заданим префіксом. Prefix може приймати наступні значення:

0 – флоппі-дисковод;

1 – фізичний диск (за замовчуванням);

2 – CDRom;

3 – віртуальний диск.

volumeoffset=[offset] or vo=[offset] – якщо заданий, відкриває диск і здійснює перехід на зміщення;

offset=[offset] or o=[offset] – перехід до заданого зміщення;

template=[template_name] or t=[template_name] – перехід до заданого зміщення в визначеному шаблоні. Можливі значення:

0 – без шаблону.

1 – шаблон MBR.

2 – шаблон завантажувального сектору NTFS.

3 – шаблон завантажувального сектору FAT.

4 – шаблон завантажувального сектору FAT32.

5 – шаблон завантажувального сектору exFAT.

6 – шаблон завантажувального сектору HFS+.
7 – шаблон завантажувального сектору Ext2/Ext3/Ext4.
8 – шаблон запису MFT NTFS.
9 – шаблон кореневого каталогу FAT.
10 – шаблон кореневого каталогу exFAT.
11 – шаблон заголовку LDM.
12 – шаблон LDM TOC.
13 – шаблон LDM VMDB.
14 – шаблон LDM Klog.
15 – шаблон LDM VBLK.
16 – шаблон кореневого каталогу HFS+.
17 – шаблон запису файлу HFS+.
18 – шаблон таблиці розділів GUID.
19 – шаблон UFS Superblock.
20 – шаблон UFS Inode.
21 – шаблон Ext2/Ext3/Ext4 Inode.

file = [ім'я файлу] або f = [ім'я файлу] – якщо вказано, цей файл буде відкрито як вихідний образ диска.

Інструмент перегляду журналу «Application Log» відстежує кожну дію, яку здійснює програма, та відображає повідомлення, сповіщення та іншу службову інформацію. Використовувати ці повідомлення можна для спостереження та подальшого розуміння потоку процесу відновлення. Записи у цьому файлі допоможуть розробникам вирішити певні проблеми. Щоб підготувати файл журналу, у діалоговому вікні «Налаштування» вмикають параметри відображення трасування і запису журналу на диску. Найкраще зберегти файл журналу на фізичному диску, який відрізняється від диска, на якому зберігаються видалені дані. Це забезпечить зниження ризику запису даних, які необхідно буде відновити.

Робота з будь-якою системною утилітою потребує уваги, оскільки внесені зміни можуть поширюватися на цілісність структури диска. Отже, зберігати результати змін необхідно тільки за умови цілковитої впевненості в збереженні цілісності усіх структур даних.

1.3.2. Завдання для самостійного виконання

1. Завантажити з офіційного сайту редактор Active@Disk Editor та встановити його на комп'ютері.
2. Переглянути в Active@Disk Editor вміст обраного диску в режимі «Читання».
3. Переглянути в Active@Disk Editor вміст обраного файлу в режимі «Читання».
4. Перейти до максимально можливого зміщення на жорсткому диску комп'ютера.
5. Перейти до максимально можливого кластера на жорсткому диску комп'ютера.
6. У вікні інструмента «Інспектор даних» вивчити характеристики жорсткого диску комп'ютера.
7. Створити закладку для переходу на останній кластер жорсткого диску. Надати їй ім'я. Перейменувати закладку.
8. Здійснити пошук сектора розташування файлу за ключовим словом у ньому.

1.3.3. Запитання для самоперевірки

1. Назвіть та охарактеризуйте робочі області програми Active@Disk Editor.
2. Яким чином можна повністю переглянути і оновити всі підключені локальні сховища даних комп'ютера?
3. Які режими роботи має Active@Disk Editor? Як здійснити перехід між ними?
4. Наведіть приклади відображення атрибутів та властивостей файлу на панелі властивостей.
5. Поясніть суть застосування структурних шаблонів. У яких випадках Active@Disk Editor їх застосовує?
6. Які дії необхідно виконати для відкрити панелі попереднього перегляду файлів?
7. Які відносні позиції пропонує Active@Disk Editor при переході до зміщення?
8. Назвіть особливості використання шістнадцяткових значень при переході до заданого сектору або зміщення.

9. У якому випадку у вікні «Перейти до сектора» («Go to Sector») буде активним поле «Перейти до кластера» («Go to cluster»)?

10. Яку задачу виконує інструмент «Інспектор даних» («Data Inspector»)?

11. Наведіть приклади використання регулярних виразів для пошуку даних на диску.

12. Назвіть шаблони, за якими виконується пошук даних в Active@Disk Editor.

Тема 1.4. Виправлення логічних помилок накопичувачів даних у MBR-секторі і PBR-області.

Мета: ознайомлення з алгоритмом виправлення помилок накопичувачів даних після збоїв.

1.4.1. Теоретичні відомості

Одним з типів фізичних помилок накопичувача є пошкодження флеш-пам'яті контролера. У цьому випадку потрібно його перепрошити на основі ідентифікатора виробника – Vendor Identifier (VID) і ідентифікатора моделі пристрою – Product Identifier (PID).

Визначення VID і PID виконуються за наступним алгоритмом:

1. Відкрити «Диспетчер устройств».

2. Знайти USB пристрій, VID і PID якого потрібно визначити.

3. Після кліка правою кнопкою, вибрати пункт «Свойства» (рис. 1.33).

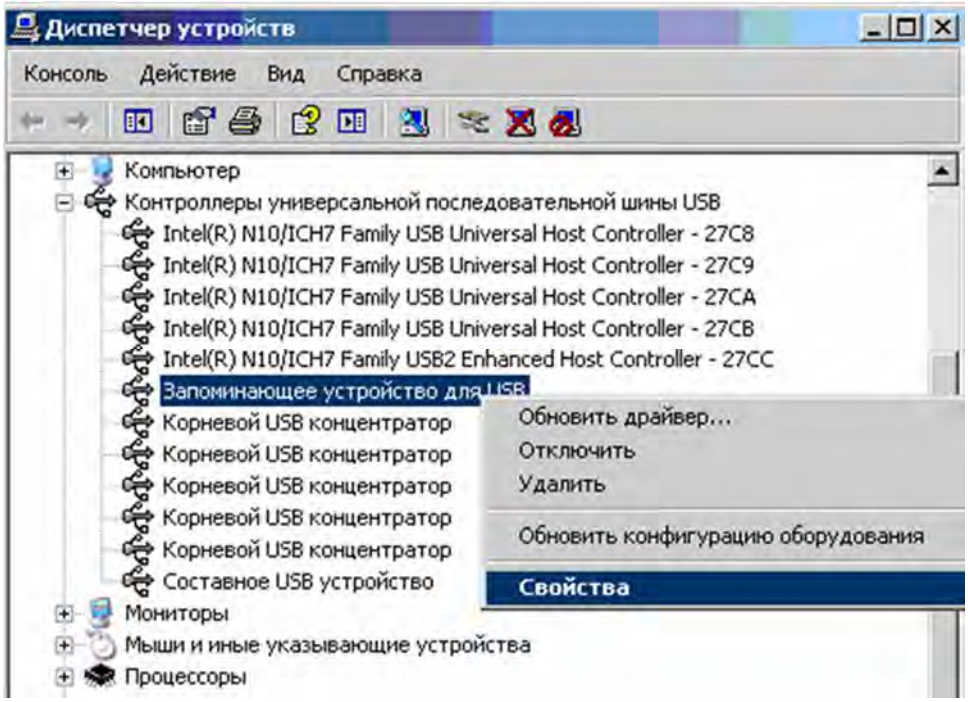


Рис. 1.33. Вибір властивостей накопичувача

4. У вікні перейти на вкладку «Сведения» і вибрати зі списку «Код экземпляра устройства» або «Родитель» (рис. 1.34).

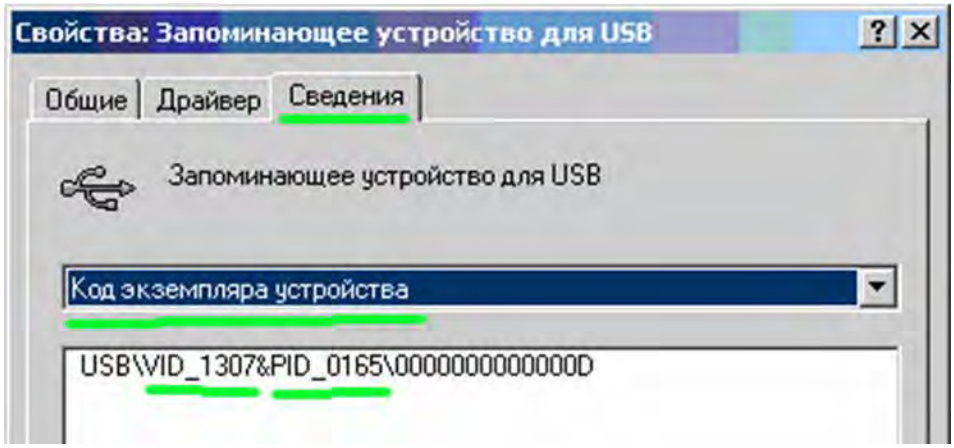


Рис. 1.34. VID і PID накопичувача

Необхідну інформацію отримали, наприклад: VID (4146), PID (ba65) і виробник (Pretec). За цією інформацією необхідно

знайти утиліту для перепрошивки саме цього типу контролерів. У мережі багато сайтів з утилітами, наприклад, flashboot.ru, а саме розділ бази даних флешок iFlash (рис. 1.35).

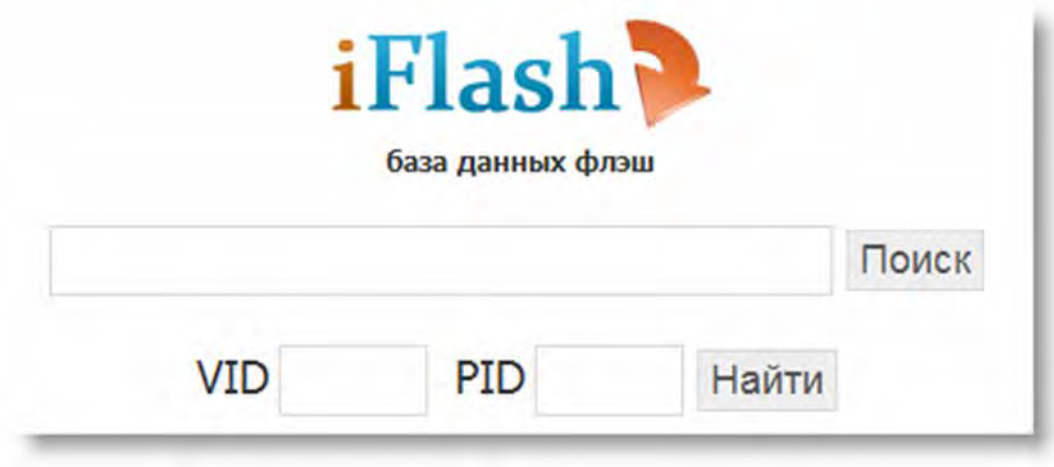
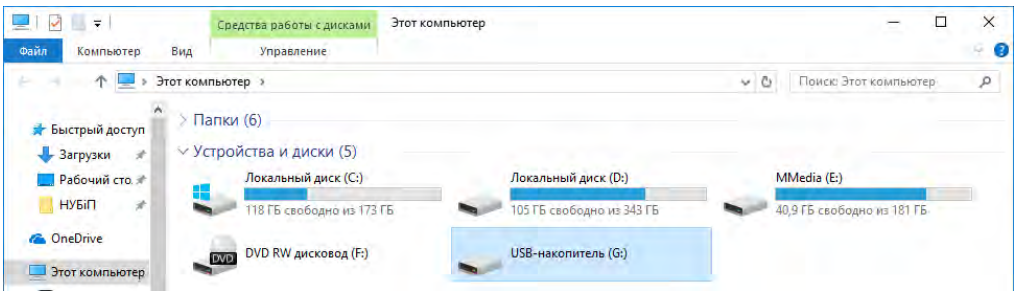


Рис. 1.35. Пошук прошивки для флеш-накопичувача

Виникнення помилки MBR-сектора призводить до ситуації, показаної на рис. 1.36. Для того, щоб виправити її, накопичувач відкривають в шістнадцятковому редакторі (наприклад, Active Disk Editor) і проводять аналіз його цілісності. За замовчуванням редактор відкривається в режимі читання (Read).



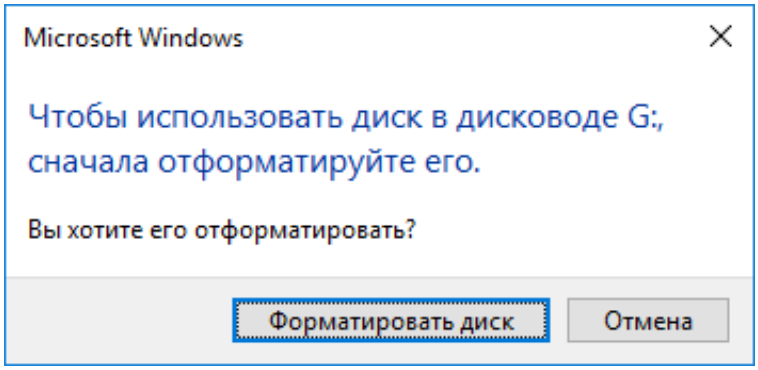


Рис. 1.36. Помилка розпізнавання накопичувача

Перші 446 байт (440 + 6) MBR сектора відведені під код завантажувача і зарезервовані сектора, тому виконувати зміни в них не можна!

Із зміщення 1BE починається Partition Table розміром 64 байт. У першому байті записується ідентифікатор розділу:

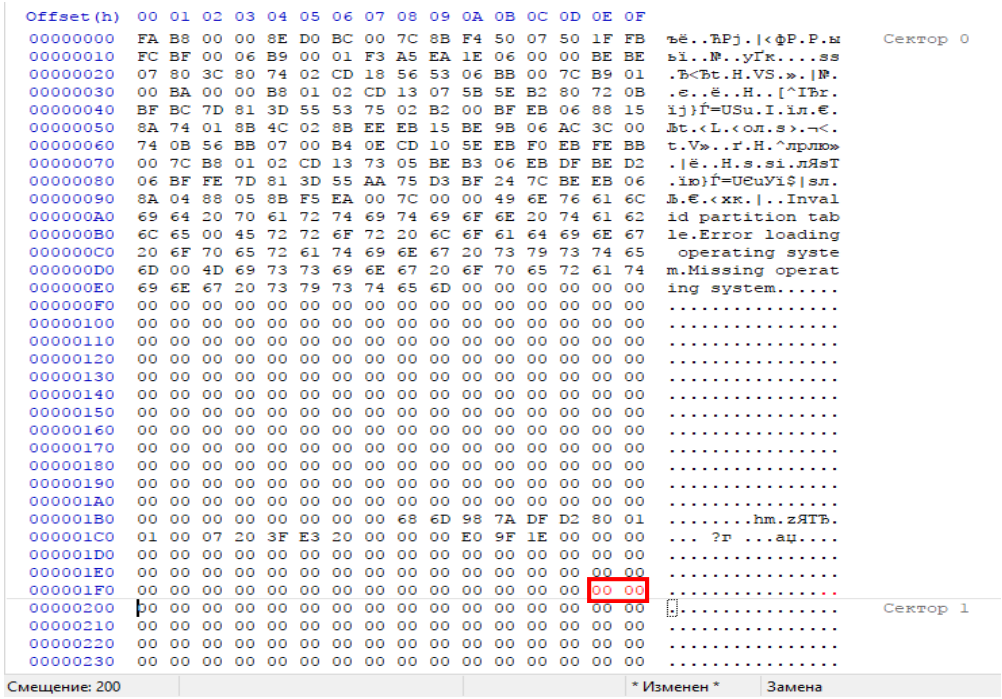
80 – активний (завантажувальний) розділ;

00 – не завантажувальний розділ.

Решта записів будуть невірні і проігноровані системою.

В останніх двох байтах Partition Table знаходиться сигнатура кінця MBR – 55 AA. У разі будь-якого відмінного запису (рис. 1.37) виконують перехід в режим читання / запису (Read / Write) (Ctrl + Alt + E) і виправляють на 55 AA. Після цього зміни зберігають, редактор закривають і витягають накопичувач. Зміни вступлять в силу під час наступного використання пристрою.

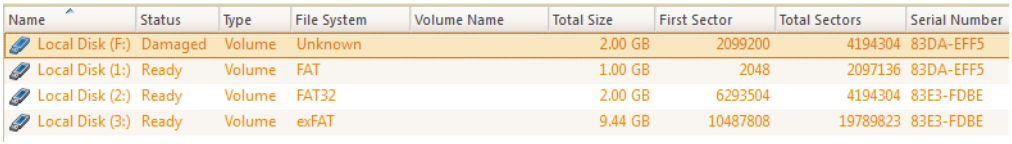
Аналогічну сигнатуру кінця має і Partition Boot Record (PBR). При її зміні накопичувач ініціалізується, але розділ позначається як пошкоджений (damaged) (рис. 1.38). Процес виправлення помилки аналогічний розглянутому для MBR.



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   FA B8 00 00 8E D0 BC 00 7C 8B F4 50 07 50 1F FB   ъё..ЋРj.|‹фР.Р.ы     Сектор 0
00000010   FC BF 00 06 B9 00 01 F3 A5 EA 1E 06 00 00 BE BE   ьї..№..уҐк....ss
00000020   07 80 3C 80 74 02 CD 18 56 53 06 BB 00 7C B9 01   .Ъ<Ъt.H.VS.».|№.
00000030   00 BA 00 00 B8 01 02 CD 13 07 5B 5E B2 80 72 0B   .є..ё..H..[^IЂr.
00000040   BF BC 7D 81 3D 55 53 75 02 B2 00 BF EB 06 88 15   ij}Ґ=USu.I.iл.€.
00000050   8A 74 01 8B 4C 02 8B EE EB 15 BE 9B 06 AC 3C 00   Љt.‹L.‹ол.s›.¬<.
00000060   74 0B 56 BB 07 00 B4 0E CD 10 5E EB F0 EB FE BB   t.V»..ґ.H.^лрлю»
00000070   00 7C B8 01 02 CD 13 73 05 BE B3 06 EB DF BE D2   .|ё..H.s.si.лЯsT
00000080   06 BF FE 7D 81 3D 55 AA 75 D3 BF 24 7C BE EB 06   .ію}Ґ=UЄuYї$|зл.
00000090   8A 04 88 05 8B F5 EA 00 7C 00 00 49 6E 76 61 6C   Љ.€.‹хк.|..Inval
000000A0   69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61 62   id partition tab
000000B0   6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67   le.Error loading
000000C0   20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65    operating syste
000000D0   6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74   m.Missing operat
000000E0   69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00 00   ing system......
000000F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000001A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000001B0   00 00 00 00 00 00 00 00 68 6D 98 7A DF D2 80 01   ........hm.zЯТЂ.
000001C0   01 00 07 20 3F E3 20 00 00 00 E0 9F 1E 00 00 00   ... ?г ...ау....
000001D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000001E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000001F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000200   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................     Сектор 1
00000210   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000220   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000230   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Смещение: 200 | * Изменен * | Замена

Рис. 1.37. Відсутність сигнатури кінця MBR



Name	Status	Type	File System	Volume Name	Total Size	First Sector	Total Sectors	Serial Number
Local Disk (F:)	Damaged	Volume	Unknown		2.00 GB	2099200	4194304	83DA-EFF5
Local Disk (1:)	Ready	Volume	FAT		1.00 GB	2048	2097136	83DA-EFF5
Local Disk (2:)	Ready	Volume	FAT32		2.00 GB	6293504	4194304	83E3-FDBE
Local Disk (3:)	Ready	Volume	exFAT		9.44 GB	10487808	19789823	83E3-FDBE

Рис. 1.38. Помилка сигнатури кінця PBR

Помилка в роботі знімного накопичувача також виникає через так званий "брудний байт ", який є особливою позначкою про необхідність його перевірки після підключення. Якщо "брудний біт " наявний, то система видає повідомлення, наведене на рис. 1.39. Причиною є незакінчені зміни через вимкнення комп'ютера до повної їх передачі на накопичувач або через виявлені пошкодження накопичувача.

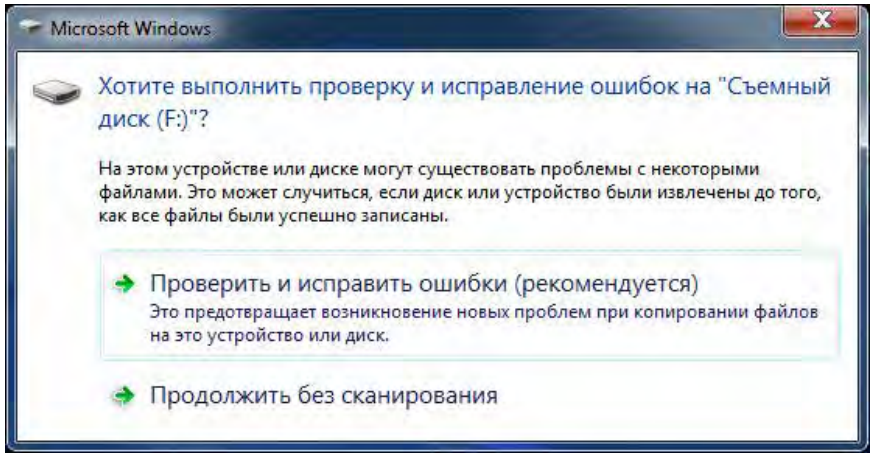


Рис. 1.39. Помилка «брудний байт» на зйомному накопичувачі з FAT 16

Вирішити цю проблему можна двома способами:

1. Запустити перевірку накопичувача на справність через вікно, що з'явилось або через командний рядок і команду chkdsk / f. Після закінчення перевірки "брудний байт " має автоматично обнулятися, проте так відбувається не завжди. Крім того, при великому об'ємі накопичувача перевірка буде здійснюватися досить довгий час.

2. Гарантовано і швидко очистити «брудний байт» можна через шістнадцятковий редактор. Він знаходиться в PBR за зміщенням 37_{10} від початку сектора і є одним з зарезервованих файловою системою (рис. 1.40).

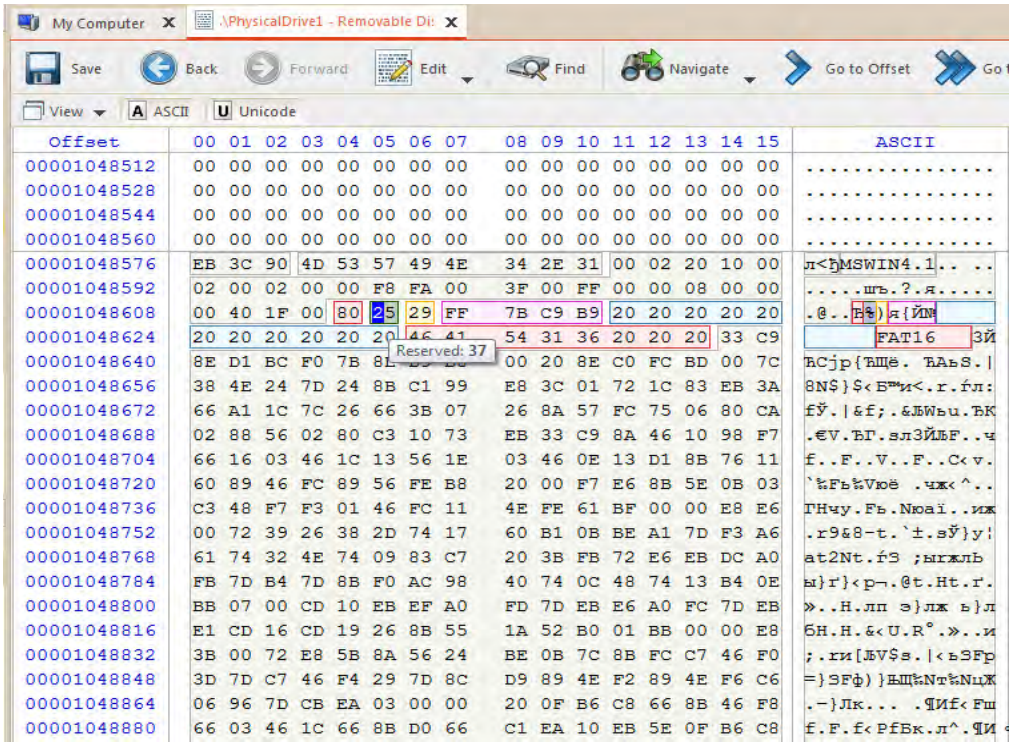


Рис. 1.40. «Брудний біт» в структурі PBR накопичувача з FAT16

Для того, щоб виправити цю помилку необхідно перейти в режим Read/Write і обнулити цей байт як показано на рис. 1.41. Після збереження всіх змін накопичувач відновить коректну роботу.

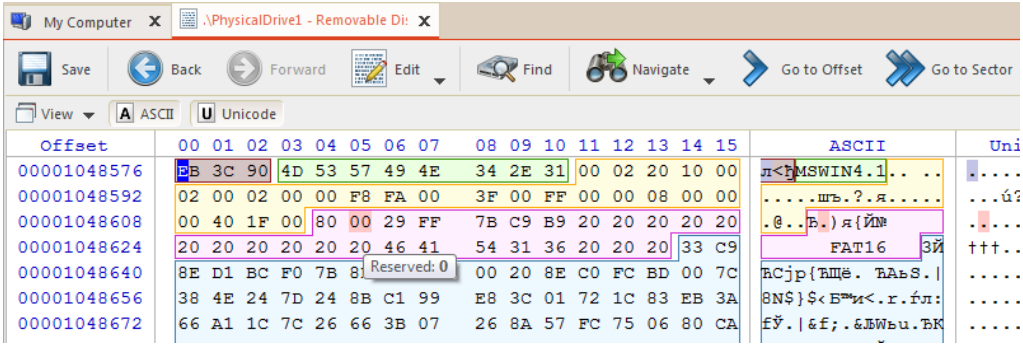


Рис. 1.41. Виправлення «брудного біта» на накопичувачі з FAT16

Для того, щоб впевнитися у відсутності «брудного біта» викликають системну утиліту **fsutil** в командному рядку і вводять команду "fsutil dirty query X:", де "X" – диск, що перевіряється (рис. 1.42).

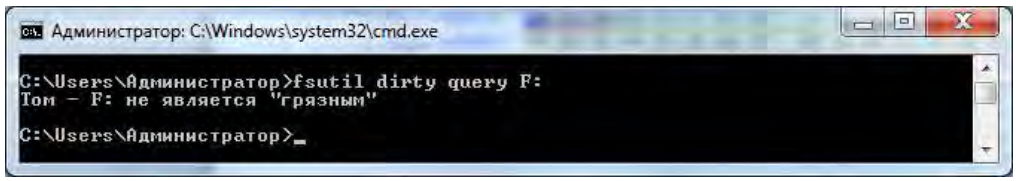


Рис. 1.42. Перевірка накопичувача після виправлення «брудного біта»

1.4.2. Завдання для самостійного виконання

1. Визначити VID і PID флеш-накопичувача і жорсткого диска (скріншот).
2. Підготувати завантажувальний флеш-накопичувач. Виконати відновлення флеш-накопичувача після помилки MBR-сектора (скріншот до відновлення і після).
3. Виписати і представити в десятковому виді основні параметри Partition table (PT) (представити у виді таблиці).
4. Виписати і представити в десятковому виді основні параметри Partition boot record (PBR) (представити у виді таблиці).
5. Змоделювати помилку «брудний біт» або використати наявний накопичувач з такою помилкою.
6. Виправити помилку «брудний біт» на накопичувачі, підтвердивши дії відповідними скріншотами.

1.4.3. Запитання для самоперевірки

1. Чому номінальний обсяг пам'яті накопичувача ніколи не буде відповідати фактичному?
2. За що відповідають ідентифікатори накопичувача VID і PID відповідно?
3. Який вид характеристик накопичувача представлений в Partition Table (PT)? Наведіть приклади.

4. Які види розділів можуть бути записані в РТ? Наведіть їх назви і ідентифікатори.

5. З якого зміщення починається РТ? Чому дорівнює її розмір? Скільки розділів може містити РТ накопичувача із розміткою MBR?

6. Що відбувається при зміні сигнатури кінця РТ на будь-яке інше значення окрім 55AA?

7. Який вид характеристик накопичувача представлений в Partition Boot Record (PBR)? Наведіть приклади.

8. Чому при пошкодженні PBR зберігається можливість роботи з накопичувачем? Чи завжди так буде?

9. Від чого залежить максимальна кількість елементів (директорій і файлів), що створюються на накопичувачі?

10. Які ще способи відновлення накопичувачів Вам відомі? На чому вони засновані?

11. Назвіть причини пошкодження магнітної поверхні жорсткого диска.

12. Дайте визначення поняття «брудний біт». Якими способами «брудний біт» може бути виправлений?

Тема 1.5. Файлова система FAT16

Мета: отримати розуміння зі структури файлової системи FAT16.

1.5.1. Теоретичні відомості

Перед тим як вивчати структуру файлової системи FAT16, необхідно створити завантажувальну флешку і відформатувати її в файловій системі FAT16. Для цього є безліч програм. Наприклад, програма BootIce, головне вікно якої наведено на рис. 1.43. Поширюється BootIce вільно для 32-х і 64-х розрядних ОС.

Для початку роботи в програмі потрібно вибрати в полі Destination Disk флешку, яка буде завантажувальною, і перейти в розділ «Parts Manage» (рис. 1.44).

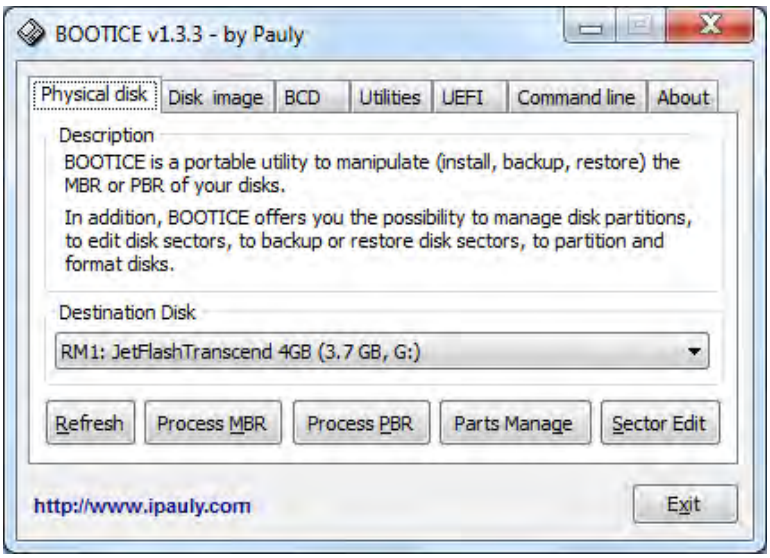


Рис. 1.43. Головне вікно програми BootIce

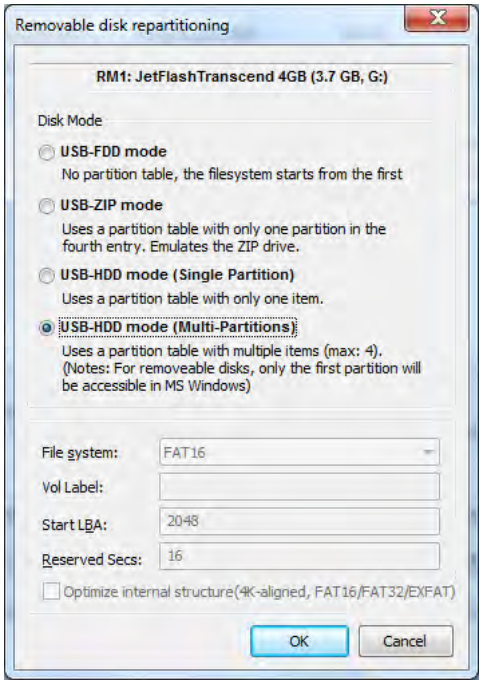


Рис. 1.44. Розділ «Parts Manage»

Далі вибрати режим форматування USB-HDD mode (Multi-Partitions), в якому можлива розбивка на кілька розділів.

На наступному кроці потрібно вибрати кількість розділів, їх розмір і тип файлової системи. Для прикладу на рис. 1.45 створено 4 розділи приблизно по 1 Гб кожен. Важливо на цьому етапі в розділі «Тип таблиці розділів» («Partition table type») встановити перемикач в положення «MBR partition table», який визначить флешку як завантажувальну.

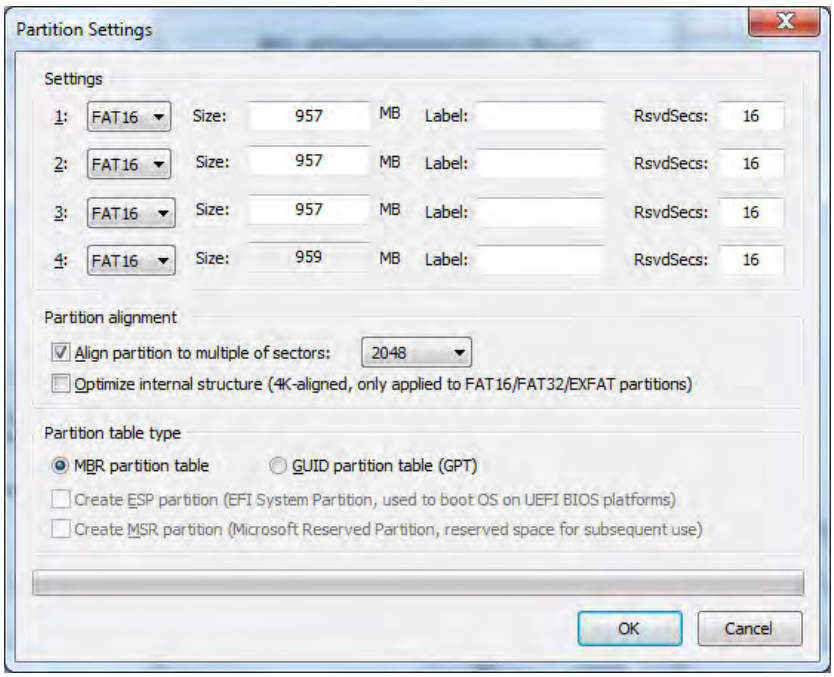


Рис. 1.45. Налаштування параметрів форматування завантажувальної флешки

Після натискання на кнопку «Ок» програма видасть попередження про видалення всіх даних і розділів на флешці (рис. 1.46).

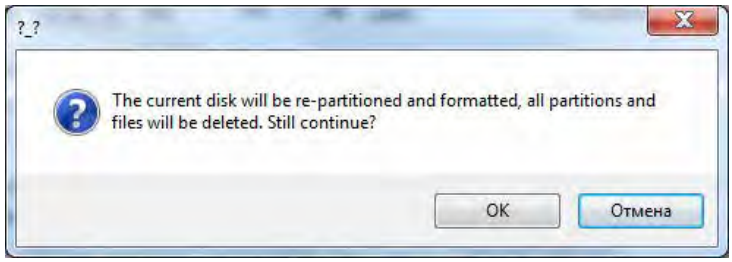


Рис. 1.46. Попередження про форматування накопичувача

Процес форматування запускається натисканням кнопки «Ок» і триває кілька секунд. У результаті отримаємо завантажувальну флешку з 4 розділами на ній в файловій системі FAT16, як показано на рис. 1.47.

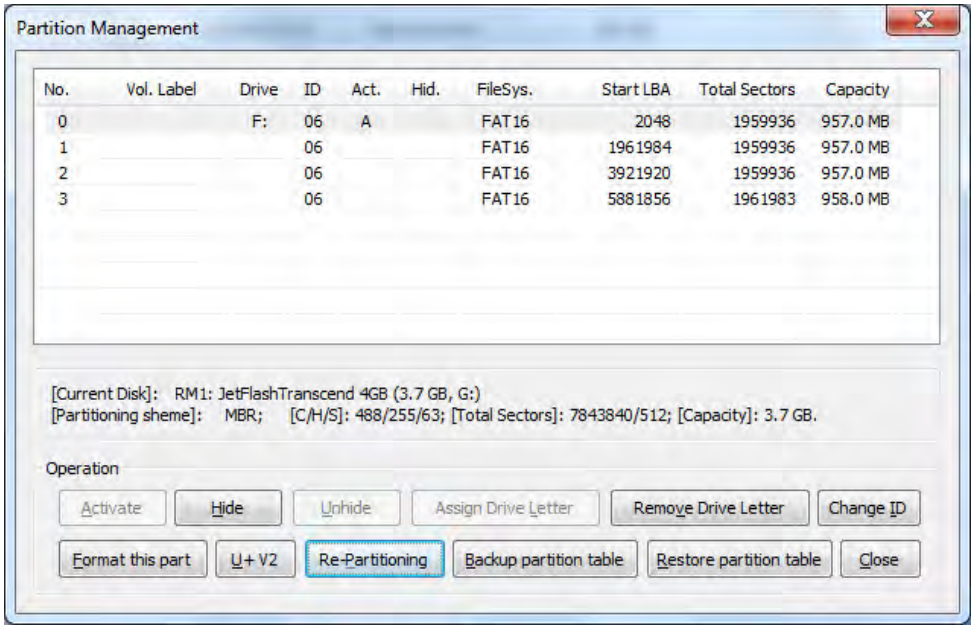


Рис. 1.47. Результат форматування накопичувача

Програма BootIce також містить вбудований шістнадцятковий редактор коду, перейти в який можна, натиснувши клавішу «Sector Edit» в головному вікні.

Як видно з рис. 3.6, ідентифікатор файлової системи FAT16 – 06. При збої в роботі обладнання ідентифікатор був випадковим чином змінений на 08. Це видно у відповідному байті Partition Table при відкритті в Active Disk Editor (рис. 1.48).

Залежно від версії ОС, ця зміна може або привести до помилки ідентифікації обладнання, або буде виправлена ОС. На лабораторній роботі ми переконалися, що ОС Windows, починаючи з версії 8.1, виправляє цю помилку, і коректно зчитує структуру файлової системи. Для більш ранніх версій можливі помилки в залежності від збірки.

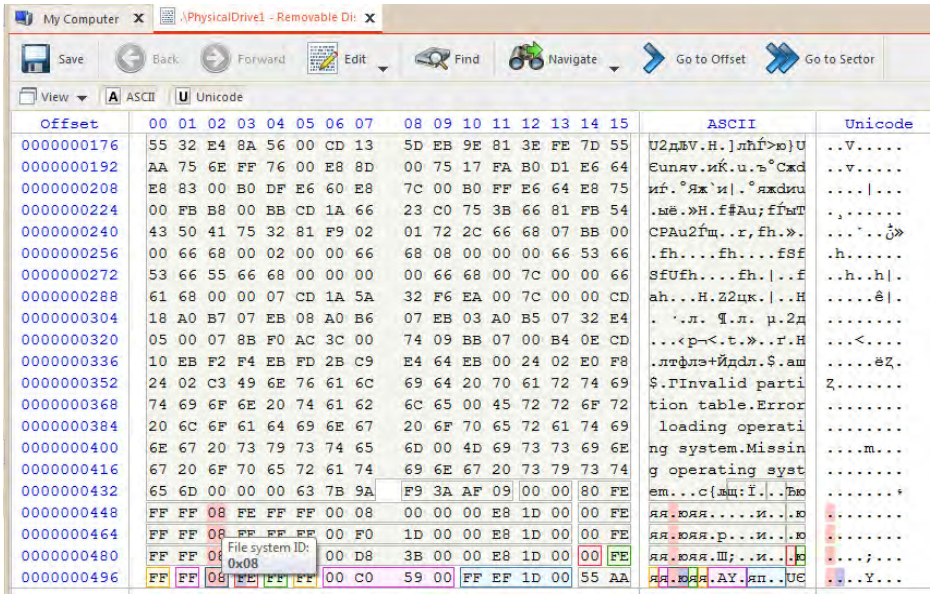


Рис. 1.48. Збій ідентифікатора файлової системи FAT16

Для вивчення структури файлової системи FAT16 створимо на флешці дві папки з іменами 1 і 2 з текстовими файлами 1.txt і 2.txt відповідно. Файли повинні бути не порожніми, інакше файлова система не виділить під них місце. Далі розглянемо алгоритм пошуку файлів і їх вмісту в файловій системі FAT16.

1. Переходимо до першого розділу Partition Table.

Знаходимо 9-12 байти по порядку від початку розділу, в яких записаний номер першого сектора розділу в шістнадцятковій системі. Випишемо ці байти в порядку їх запису: 00 08 00 00 (рис. 1.49). Для перетворення в десяткову систему байти попарно потрібно переставити. Після перестановки отримаємо: 00 00 08 00. Тоді $800_{16} = 8 * 16^2 = 2048$. Отже, перший розділ починається з 2048 сектора.

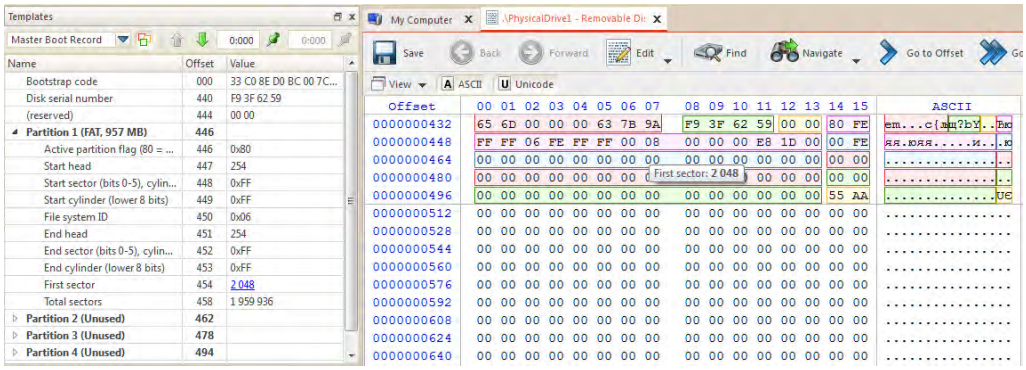


Рис. 1.49. Визначення номера першого сектора активного розділу

2. Виконуємо перехід в PBR першого розділу. У відповідних байтах знаходимо кількість секторів на кластер (14 байт) (рис. 1.50), резервних секторів (15 і 16 байти) (рис. 1.51), секторів на FAT (22 і 23 байти) (рис. 1.52). Значення з шістнадцяткової системи переводимо в десяткову.

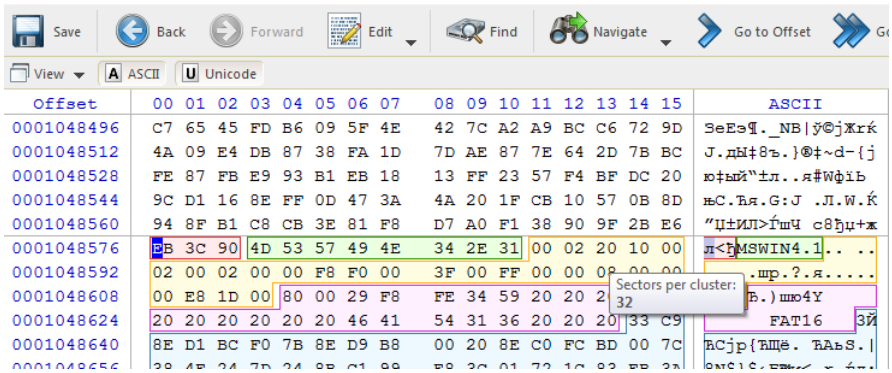


Рис. 1.50. Визначення кількості секторів на кластер

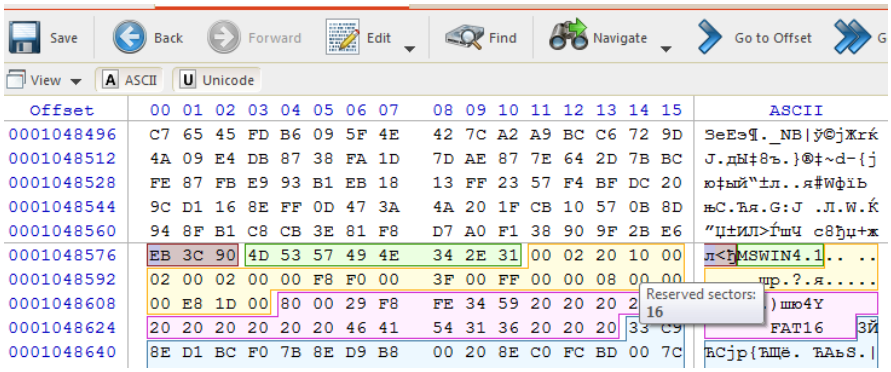


Рис. 1.51. Визначення кількості резервних секторів в розділі

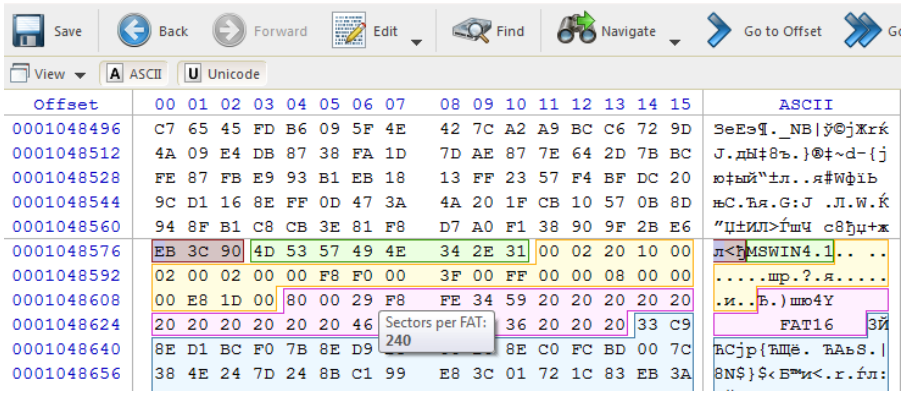


Рис. 1.52. Визначення кількості секторів на FAT

3. Визначаємо номер сектора, в якому починається кореневий каталог за формулою: ROOT = № 1-го сектора PBR + 2FAT + RESERVED = 2048 + 2 * 240 +16 = 2544. Перевірка розрахунку приведена на рис. 1.53.

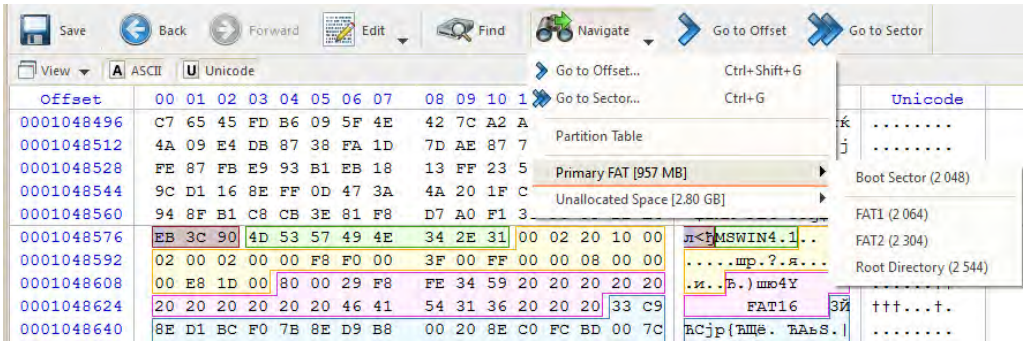


Рис. 1.53. Перевірка обчислень початку кореневого каталогу

4. Переходимо в початок кореневого каталогу – 2544 сектор. Тут знаходимо записи про директорії 1 і 2 (рис. 1.54). Далі алгоритм дій розглянемо тільки для папки «1».

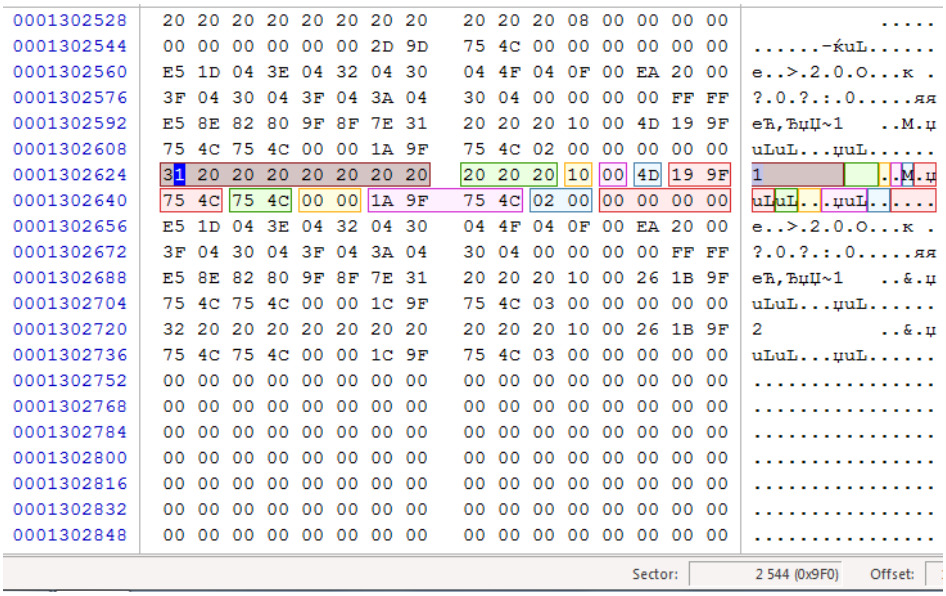


Рис. 1.54. Запис про директорію «1» в кореневому каталозі

У структурі запису про папку знаходимо 19 (high word) і 21 (low word) байти, в яких записаний номер кластера розташування папки.

Випишемо ці байти: 00 00 02 00. Після перестановки байт отримаємо: 00 00 00 02_{16} = 2 * 16^0 = 2_{10}. Значить, папка знаходиться в другому кластері.

5. Переходимо у вміст папки «1». Для цього виконаємо розрахунок за формулою: FOLDER = ROOT + (N-1), де N – номер кластера вмісту папки. В результаті отримаємо: FOLDER = 2544 + 1 кластер = 2544 + 32 сектора = 2576 сектор.

За допомогою команди Navigate переходимо в 2576 сектор (рис. 1.55).

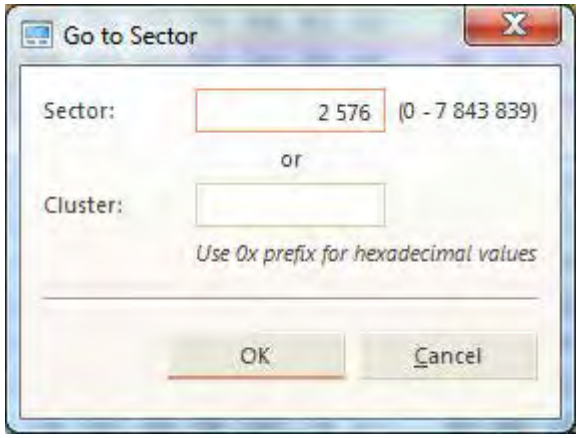


Рис. 1.55. Перехід в зазначений сектор

І виділяємо запис про вміст папки – файлу 1.txt (рис. 1.56).

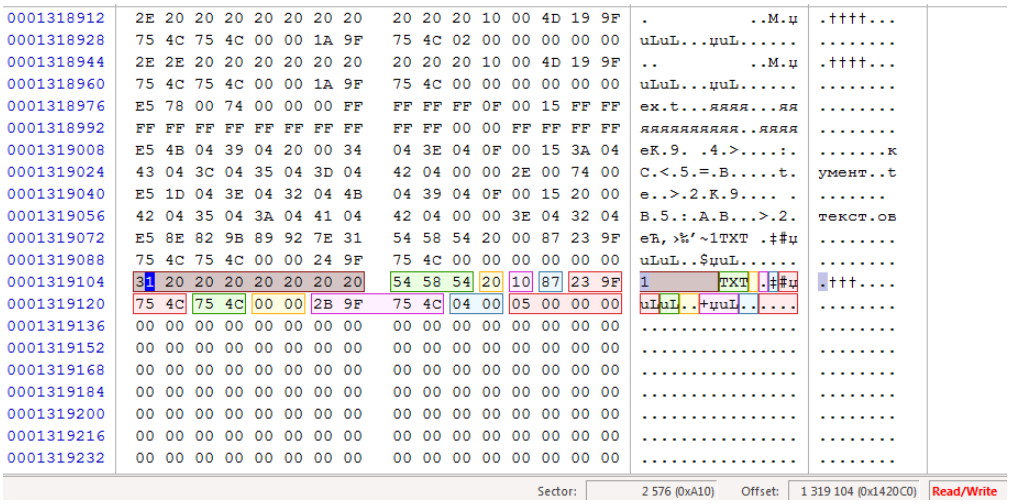


Рис. 1.56. Запис про файл 1.txt

6. У записі про файл 1.txt знаходимо 19 (high word) і 21 (low word) байти і виписуємо їх: 00 00 04 00. Після перестановки отримаємо: 00 00 00 0416 = 4 * 160 = 4 кластер.

Значить, вміст файлу 1.txt знаходиться за формулою: FILE = ROOT + (N-1) = 2544 + 3 кластера = 2544 + 3 * 32 секторів = 2640 сектор (рис. 1.57).

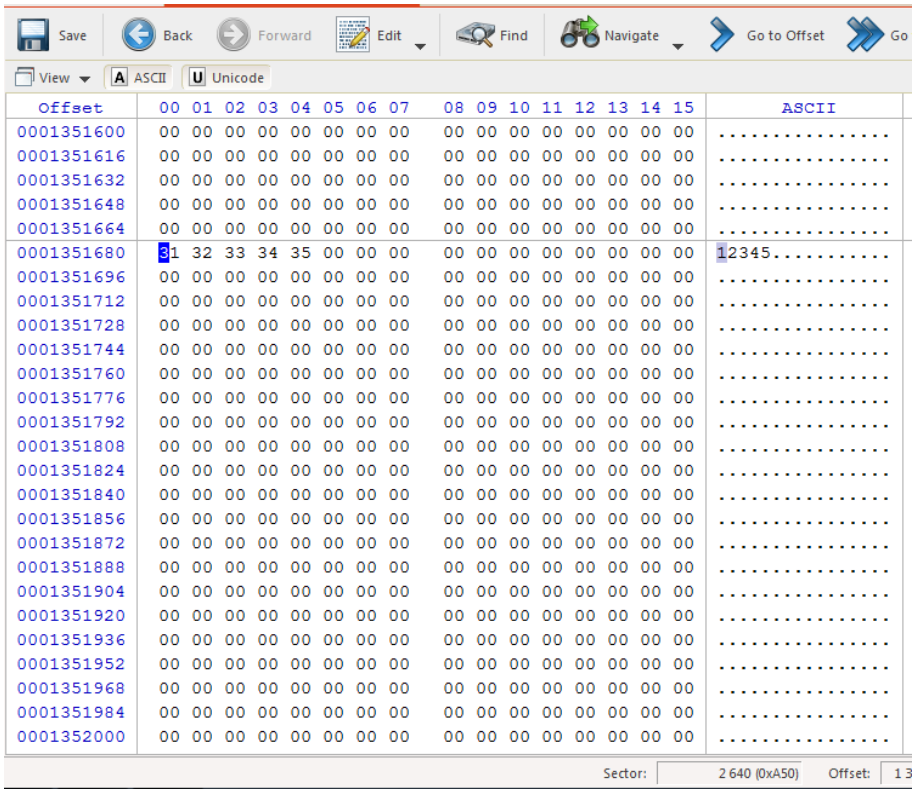


Рис. 1.57. Вміст файлу 1.txt

Для визначення часу та дати створення файлу використовують наступні байти запису про файл: час – 14,15 байти, дата – 16,17 байти.

Розглянемо приклад на рис. 1.58:

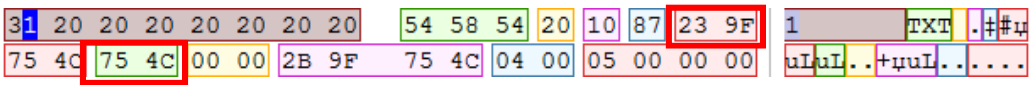


Рис. 1.58. Записи про дату і час створення файлу

Час = $239F_{16}$

$9F23_{16}$= 10011 111001 00011_2

10011_2 = 19 ч;

111001_2 = 57 хв;

00011_2 = 3×2 = 6 с.

Час створення файлу: 19 г 57 хв 6 с.

Дата = $754C_{16}$

$4C75_{16} = 100110\ 0011\ 10101_2$
$100110_2 = 38+1980=2018$ р.
$0011_2 = 3$ м;
$10101_2 = 21$ д.
Дата створення файлу: 21.03.2018 р.

1.5.2. Завдання до самостійного виконання

1. Створити завантажувальну флешку з файловою системою FAT16.
2. Перейти в Partition Table і змінити ідентифікатор файлової системи. Зробити висновок після збереження дій.
3. Створити в кореневому каталозі папки і файли в них. Файли не повинні бути порожніми !!!
4. Знайти сектора, в яких розташовані всі створені папки і файли, проілюструвавши свої дії і зміст структурних елементів скріншотами.
5. В процесі пошуку виписати основні дані, що зберігаються в PBR у вигляді таблиці.
6. Визначити дату створення файлу.
7. Визначити час створення файлу.
8. Змоделювати помилку «брудний біт» або використати наявний накопичувач з такою помилкою.
9. Виправити помилку «брудний біт» на накопичувачі і виконати перевірку за допомогою системної утиліти, підтвердивши дії відповідними скріншотами.

Примітки:

1. Не забувайте про вибір потрібного Template в правій частині Active Disk Editor, інакше програма буде накладати неправильний шаблон.
2. Для внесення змін використовується режим Read / Write, для переходу в який використовується поєднання клавіш: Ctrl + Alt + E.

1.5.3. Запитання для самоперевірки

1. Назвіть в порядку слідування логічні області файлової системи FAT16. Поясніть їх функціональні особливості.
2. Що таке high word і low word у записі про файл або директорію?
3. Назвіть особливість, присутню у всіх файлових системах, для коректного зчитування даних з high word і low word.
4. Наведіть і поясніть формулу, за якою визначається номер сектора початку кореневого каталогу.
5. Наведіть і поясніть формулу, за якою визначається номер сектора початку запису про директорію або файл.
6. Поясніть необхідність наявності даних у створюваних на накопичувачі файлах.
7. Наведіть ідентифікатор для файлової системи FAT16. Поясніть, що відбувається при його випадковій зміні.
8. Посніть різницю у логічній структурі завантажувального накопичувача і не завантажувального.
9. За допомогою якої команди Active Disk Editor здійснюється перехід по секторах і кластерах?
10. Назвіть два режими роботи програми Active Disk Editor. Яким чином здійснюється перехід між ними?
11. Наведіть алгоритм визначення дати створення файлу.
12. Наведіть алгоритм визначення часу створення файлу.

Тема 1.6. Файлова система FAT32

Мета: отримати знання зі структур файлової системи FAT32.

1.6.1. Теоретичні відомості

Перед початком виконання даної лабораторної роботи необхідно так само як і в лабораторній роботі №3 створити

завантажувальний флеш-накопичувач за допомогою програми BootIce, і відформатувати його в файловій системі FAT32. Необхідний результат форматування наведений на рис. 1.59.

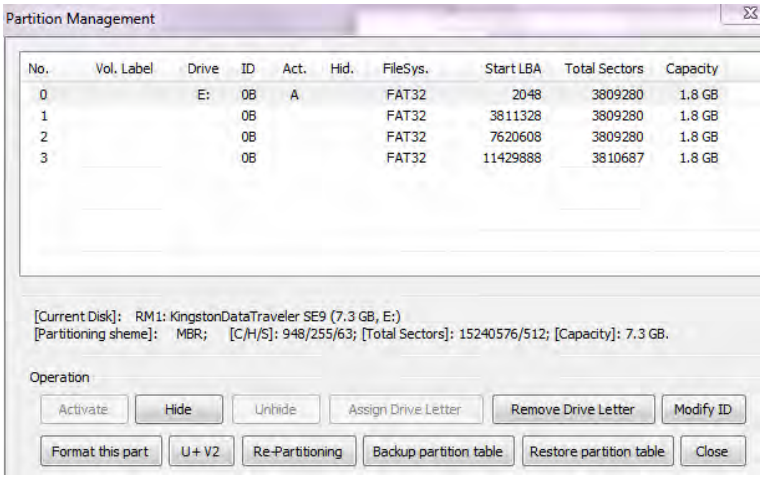


Рис. 1.59. Результат форматування флеш-накопичувача в FAT32

Ідентифікатор файлової системи FAT32 – 08, проте у стовпці ID видно, що ідентифікатор приймає значення 0B. Обидва ідентифікатора відповідають файловій системі FAT32, проте 08 зустрічається у більш ранніх моделях флеш-накопичувачів. Ідентифікатори 0B, а також 0C зустрічаються у більш нових моделях, і використовуються у відповідності до номінального об’єму.

При моделюванні помилки збою ідентифікатора різні операційні системи реагують по-різному. Наприклад, змінимо ідентифікатор 0C, як показано на рис. 1.60, на ідентифікатор файлової системи FAT16 – 06.

У лабораторній роботі №3 ми переконалися, що ОС Windows, починаючи з версії 8.1, виправляє цю помилку, і коректно зчитує структуру файлової системи. Для більш ранніх версій можливі помилки в залежності від збірки.

До зміни:

00000000432	65	6D	00	00	00	63	7B	9A	81	6D	00	00	00	00	80	FE
00000000448	FF	FF	0C	FE	FF	FF	00	08	00	00	FF	B7	9C	03	00	00
00000000464	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000496	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA

Після зміни:

00000000432	65	6D	00	00	00	63	7B	9A	81	6D	00	00	00	00	80	FE
00000000448	FF	FF	06	FE	FF	FF	00	08	00	00	FF	B7	9C	03	00	00
00000000464	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000496	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA

Рис. 1.60. Зміна ідентифікатора файлової системи в Partition Table

Для вивчення структури файлової системи FAT32 створимо на флешці директорію з іменем 111 з текстовим файлом 222.txt (рис. 1.61, 1.62). Файл повинен бути не порожнім, інакше файлова система не виділить під нього місце.

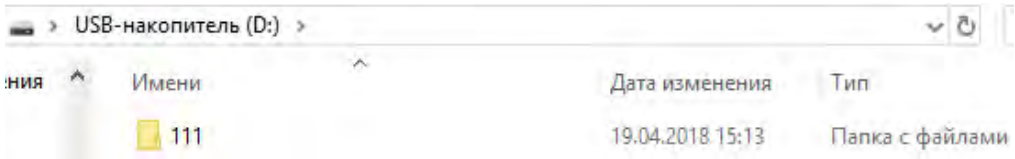


Рис. 1.61. Вміст флешки

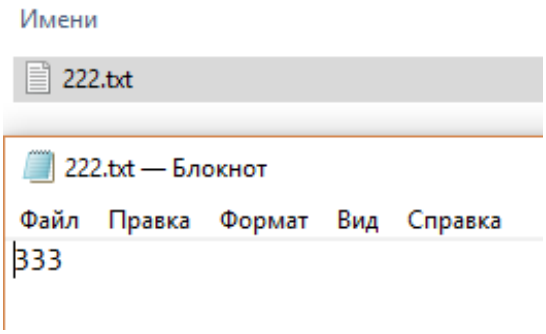


Рис. 1.62. Вміст файлу

Далі розглянемо алгоритм пошуку файлів і їх вмісту в файловій системі FAT32.

1. Переходимо до першого розділу Partition Table.

Знаходимо 9-12 байти по порядку від початку розділу, в яких записаний номер першого сектора розділу в шістнадцятковій системі. Випишемо ці байти в порядку їх запису: 00 08 00 00 (рис. 1.63). Для перетворення в десяткову систему байти попарно потрібно переставити. Після перестановки отримаємо: 00 00 08 00. Тоді $800_{16} = 8*16^2 = 2048$. Отже, перший розділ починається з 2048 сектора.

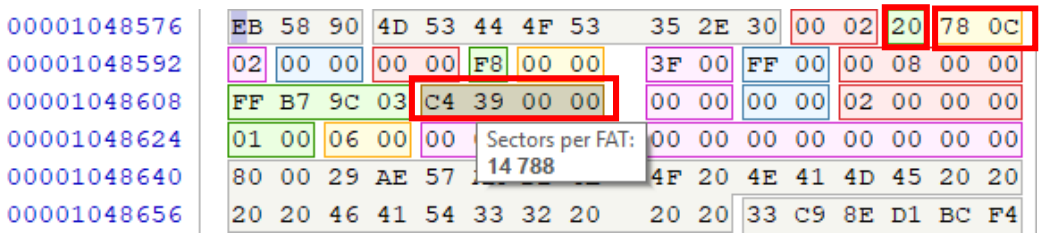


Рис. 1.63. Визначення параметрів PBR першого розділу

2. Виконуємо перехід в PBR першого розділу. У відповідних байтах знаходимо кількість секторів на кластер (14 байт – 20_{16} – 32_{10} сектора на кластер), резервних секторів (15 і 16 байти – 0C 78_{16} – 3192_{10} резервних секторів), секторів на FAT (36-39 байти – 00 00 39 $C4_{16}$ – 14788_{10} секторів на FAT).

Визначаємо номер сектора, в якому починається кореневий каталог за формулою: ROOT = № 1-го сектора PBR + 2FAT + RESERVED = 2048 + 2* 14788 + 3192 = 34816.

Як видно з рис. 1.64, номер сектора кореневого каталогу співпав з номером, який вказано в темплейті.

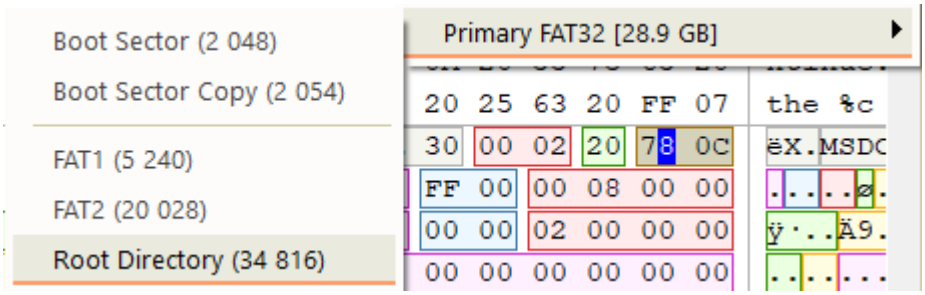


Рис. 1.64. Номер сектора початку кореневого каталогу

Переходимо в початок кореневого каталогу – 34816 сектор. Тут знаходимо записи про директорію 111 (рис. 1.65).

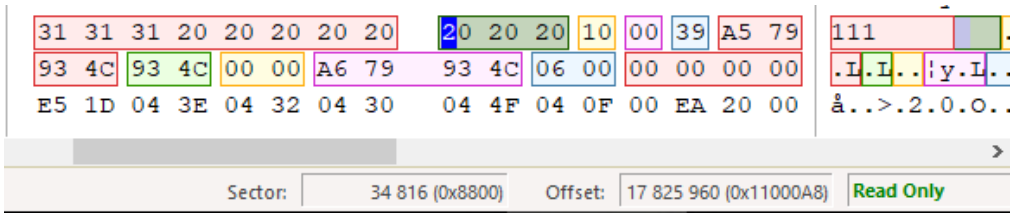


Рис. 1.65. Запис про директорію 111

У структурі запису про директорію знаходимо «high word» і «low word» байти, в яких записаний номер кластера розташування папки. Випишемо ці байти: 00 00 00 06 = 6_{10}. Значить, директорія знаходиться в шостому кластері.

Переходимо у вміст директорії «111». Для цього виконаємо розрахунок за формулою: FOLDER = ROOT + (N-2), де N – номер кластера вмісту папки. У результаті отримаємо: FOLDER = 34816 + 4*32 = 34944 сектор (рис. 1.66).

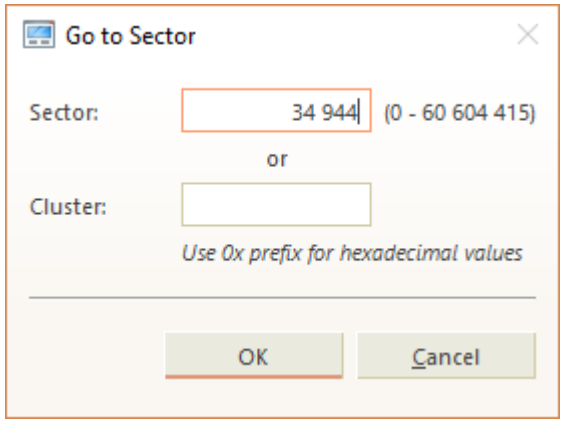


Рис. 1.66. Перехід в сектор вмісту директорії 111

У запису про файл 222.txt знаходимо «high word» і «low word» байти і виписуємо їх: 00 00 07 00 (рис. 1.67). Після перестановки отримаємо: 00 00 00 0716 = 7 кластер.

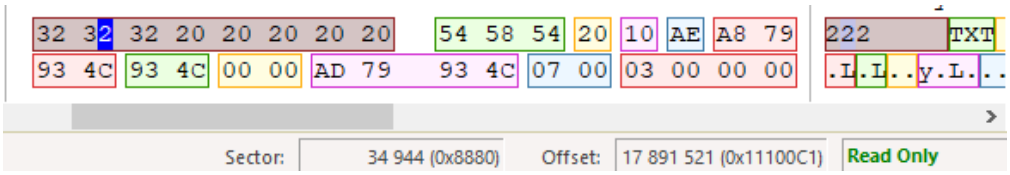


Рис. 1.67. Перехід в сектор запису про файл 2.txt

Значить, вміст файлу 222.txt знаходиться за формулою: FILE = ROOT + (N-2) = 34816 + 5 кластера = 34816 + 5*32 секторов = 34976 сектор (рис. 1.68).

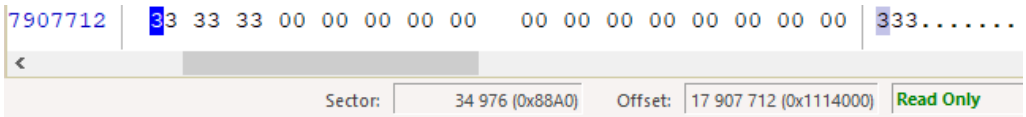


Рис. 1.68. Перехід в сектор із вмістом файлу 2.txt

Для визначення часу та дати створення файлу використовують наступні байти запису про файл: час – 14,15 байти, дата – 16,17 байти.

Розглянемо приклад на рис. 1.69:

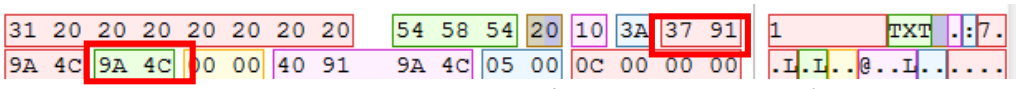


Рис. 1.69. Записи про дату і час створення файлу

Час = 3791_{16}
9137_{16}= 10010 001001 10111_2
10010_2=18 г;
001001_2=9 хв;
10111_2=23=23*2=46 с.

Дата = 9A 4C
4C $9A_{16}$ = 100110 0100 11010_2
100110_2 = 1980+38=2018 р;
0100_2 = 4 м;
11010_2 = 2+8+16 = 26 д.

Некоректну роботу накопичувача часто визиває так званий «брудний біт», який є особливою позначкою про необхідність його перевірки після підключення. Якщо "брудний біт " наявний, то система видає повідомлення, наведене на рис. 1.70. Причиною є незакінчені зміни через вимкнення комп'ютера до повної їх передачі на накопичувач або через виявлені пошкодження накопичувача.

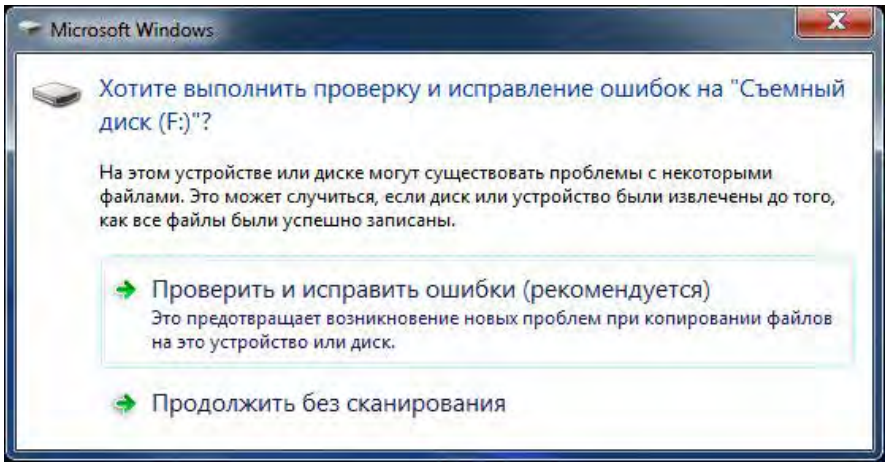


Рис. 1.70. Помилка «брудний біт» на зйомному накопичувачі з FAT32

Вирішити цю проблему можна двома способами:

1. Запустити перевірку накопичувача на справність через вікно, що з'явилось або через командний рядок і команду chkdsk / f. Після закінчення перевірки "брудний біт " має автоматично обнулятися, проте так відбувається не завжди. Крім того, при великому об'ємі накопичувача перевірка буде здійснюватися досить довгий час.

2. Гарантовано і швидко очистити «брудний біт» можна через шістнадцятковий редактор. Він знаходиться в PBR за зміщенням 65_{16} (41_{10}) від початку сектора і є одним з зарезервованих файловою системою (рис. 1.71).

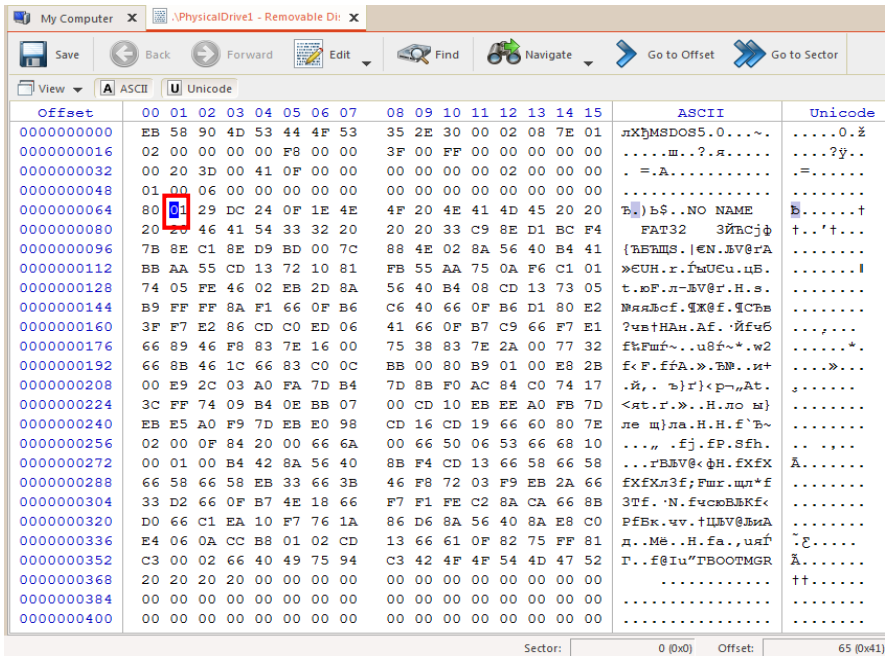


Рис. 1.71. Виявлення «брудного біта» на накопичувачі з FAT32

Для того, щоб виправити цю помилку необхідно перейти в режим Read/Write і обнулити цей біт як показано на рис. 1.72. Після збереження всіх змін накопичувач відновить коректну роботу.

Для того, щоб впевнитися у відсутності «брудного біта» викликають системну утиліту **fsutil** в командному рядку і виконують запит показаний на рис. 1.73.

1.6.2. Завдання до самостійного виконання

1. Створити завантажувальну флешку з файловою системою FAT32.
2. Перейти в Partition Table і змінити ідентифікатор файлової системи. Зробити висновок після збереження дій.
3. Створити в кореневому каталозі папки і файли в них. Файли не повинні бути порожніми !!!
4. Знайти сектора, в яких розташовані всі створені папки і файли, проілюструвавши свої дії скріншотами.
5. Визначити дату і час створення файлів.

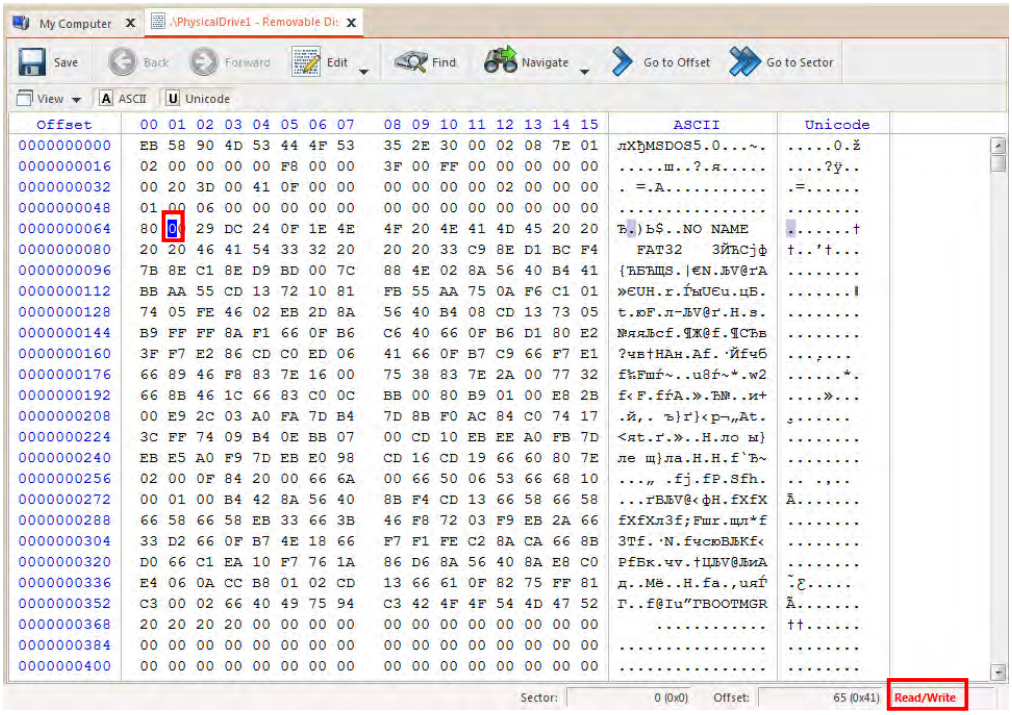


Рис. 1.72. Виправлення «брудного біта» на накопичувачі з FAT32

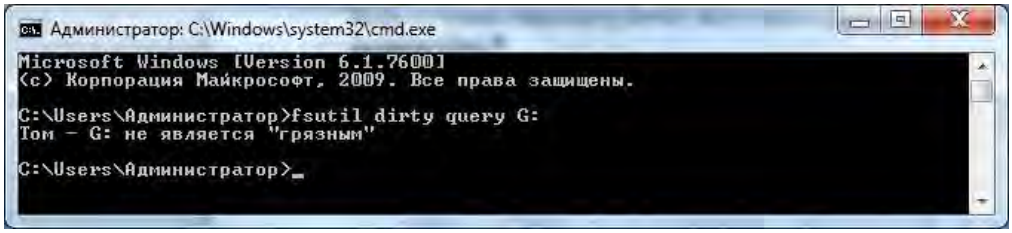


Рис. 1.73. Перевірка наявності «брудного біта» на накопичувачі з FAT32

6. У процесі пошуку виписати основні дані, що зберігаються в PBR у виді таблиці.

7. Змоделювати помилку «брудний біт» або використати наявний накопичувач з такою помилкою.

8. Виправити помилку «брудний байт» на накопичувачі і виконати перевірку за допомогою системної утиліти, підтвердивши дії відповідними скріншотами.

1.6.3. Запитання для самоперевірки

1. Назвіть в порядку слідування логічні області файлової системи FAT32.

2. Порівняйте між собою логічні області файлових систем FAT16 і FAT32.

3. Наведіть і поясніть формулу, за якою визначається номер сектора початку запису про директорію або файл. Порівняйте із файловою системою FAT16. Поясніть відмінності.

4. Наведіть ідентифікатор для файлової системи FAT32. Поясніть, що відбувається при його випадковій зміні.

5. Файл на логічному диску FAT32 займає один кластер. Який код (в hex) буде в дескрипторі цього кластера в FAT-таблиці?

6. Поясніть на прикладі як визначається дата створення файлу.

7. Поясніть на прикладі як визначається час створення файлу.

Тема 1.7. Обчислення і перевірка контрольної суми

Мета: Навчитися обчислювати контрольну суму файлів з використанням різних алгоритмів.

1.7.1. Теоретичні відомості

Контрольна сума (CRC) – це метод перевірки цілісності прийнятої інформації на стороні приймача при передачі по каналам зв'язку.

Наприклад, одна з простих перевірок – використання біта парності. Виконується обчислення суми всіх бітів повідомлення для передачі, і якщо сума виявляється парною, то в кінці повідомлення додається 0, якщо ні – то 1. При прийомі також підраховується сума кодів повідомлень, і порівнюється з прийнятим бітом парності. Якщо він відрізняється, значить, при передачі виникли помилки, і інформація була викривлена. Так само контрольні суми можуть бути використані для швидкого порівняння двох наборів даних на нееквівалентність: з великою ймовірністю різні набори даних матимуть неадекватні контрольні суми. Це може бути використане, наприклад, для виявлення комп'ютерних вірусів. Незважаючи на свою назву, контрольна сума не обов'язково розраховується шляхом додавання.

Але такий спосіб визначення наявності помилок дуже неінформативний і працює не завжди, оскільки при розбитті декількох повідомлень біт парності суми може не змінюватися. Тому існує безліч більш складних перевірок.

По суті, CRC – це не сума, а результат розподілу деякого обсягу інформації (інформаційного повідомлення) на константи, а точніше – залишок від розподілу повідомлення на константи. Тим не менше, CRC історично також називають «контрольна сума».

У значення CRC вносить вклад кожний біт повідомлення. Тож, якщо хоча б один біт вихідного повідомлення змінюється при передачі, контрольна сума теж змінюється, причому істотно. Це великий плюс такої перевірки, оскільки вона

дозволяє однозначно визначити, спотворено вихідне повідомлення при передачі чи ні.

Вихідним повідомленням є безперервна послідовність бітів необмеженої довжини. Константа, на яку потрібно розділити вихідне повідомлення, є деяким числом також будь-якої довжини, але звичайно використовуються байти, адже робота виконується з байтами, а не з бітами.

Константа–дільник записуються у вигляді полінома (многочлена) в такому виді: $x^8 + x^2 + x^1 + x^0$. Тут ступінь числа «x» означає позицію біта-одиниці в кількості, починаючи з нуля, а старший розряд вказує на ступінь полінома і відкидається при інтерпретації числа. Записаний раніше номер – це не що інше як 00000111 в двійковій системі числення, або 7 в десятковій.

Ось ще приклад: $x^{16} + x^{15} + x^2 + x^0$ = (1) 1000000000000101 = 0x8005 = 32773.

Зазвичай використовують стандартні многочлени для різних типів CRC. Наприклад, деякі з них мають вид:

Алгоритм CRC	Утворювальний многочлен
CRC–16	0x8005
CRC–16–CCITT	0x1021
CRC–16–DNP	0x3D65
CRC–32–IEEE 802.3	0x04C11DB7
CRC–32C	0x1EDC6F41
CRC–32K	0x741B8CD7

Існує базовий метод – ділення повідомлення на поліном «в лоб» і його модифікація з метою зменшення кількості обчислень і, відповідно, прискорення розрахунку CRC.

Значення контрольної суми додається в кінець блока даних безпосередньо перед початком передачі або запису даних на носій інформації. У результаті вона перевіряється для підтвердження цілісності даних.

Популярність використання контрольних сум для перевірки цілісності даних обумовлена тим, що подібна

перевірка просто реалізується в двійковому цифровому обладнанні, легко аналізується і добре підходить для виявлення спільних помилок, викликаних наявністю шуму в каналах передачі даних.

Криптографічна функція MD5 вже майже не використовується для визначення контрольних сум, оскільки виявилося, що для неї можна швидко створити за допомогою сучасних комп'ютерів два різних файли, що мають різну довжину в байтах, але однакові величини контрольних сум, підрахованих за допомогою алгоритму MD5.

Використання терміна «сума» пов'язана з тим, що на початок цифрового зв'язку при байтових передачах інформаційними були 7 біт, а восьмий – контрольним.

Циклічний вичерпний код (зокрема, CRC8, CRC16, CRC32) застосовується для перевірки цілості передачі даних. Програми-архіви включають CRC вихідних даних у створеному архіві, щоб одержувач міг переконатися в правильності отриманих даних. Така контрольна сума проста у реалізації та забезпечує низьку вірогідність виникнення колізій.

MD5 та інші криптографічні хеш-функції використовуються, наприклад, для підтвердження цілісності та автентичності даних.

Виконати порівняння обчислюваної величини контрольної суми з оригіналом. Для програмного продукту порівняйте контрольну суму, зазначену на сайті виробника (рис 1.74) з розрахунковою після завантаження.

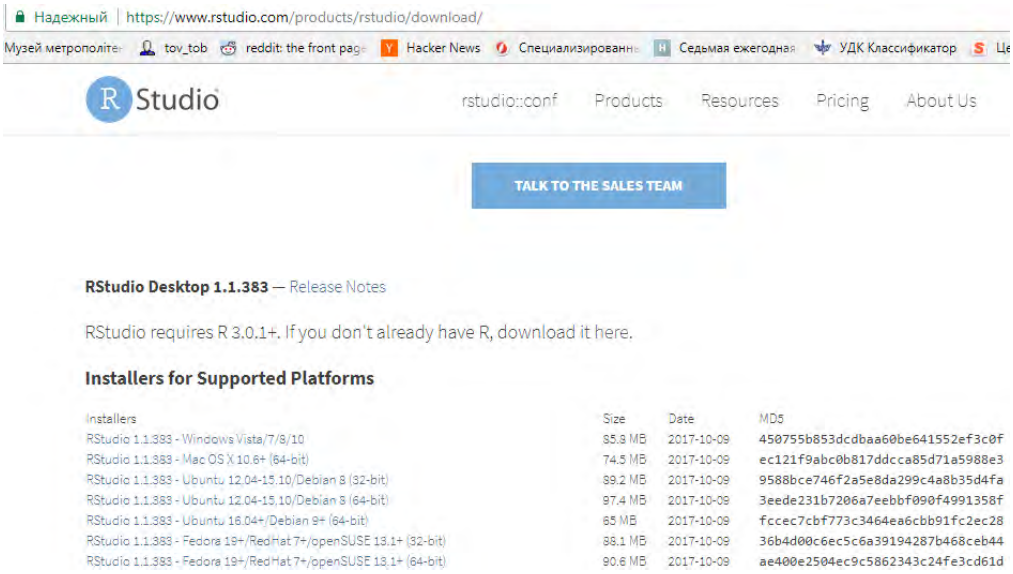


Рис. 1.74. Приклад оригінальної контрольної суми для програмного продукту

Дізнатися хеш-суму файлу можна через командну строку.

Для цього використовується утиліта CertUtil в комплекті Windows. Команда має наступний синтаксис:

certutil –hashfile c:file

де, c: file - шлях до файлу.

За замовчанням утиліта обчислює хеш-суму за допомогою алгоритму SHA1 (рис. 1.75). Так само доступні MD5 MD4 MD2 SHA512 SHA384 SHA256 SHA1 (рис. 1.76). Наприклад, **certutil-hashfile c: файл MD5**.

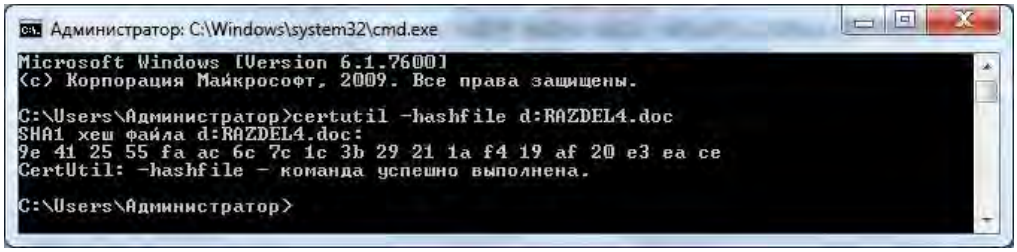


Рис. 1.75. Обчислення контрольної суми файлу за алгоритмом SHA1

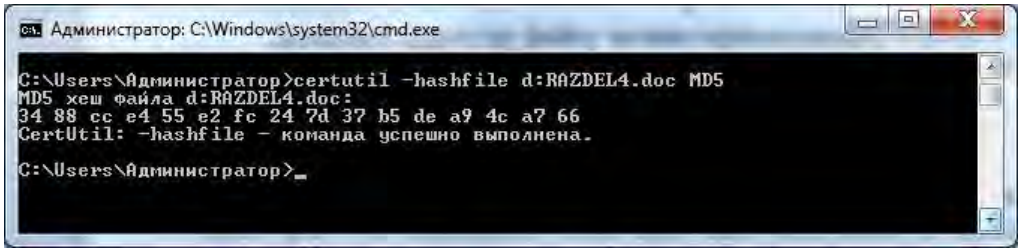


Рис. 1.76. Обчислення контрольної суми файлу за алгоритмом MD5

Виведення відмінностей у змісті файлів.

Якщо в даних файлах є відмінності, то контрольна сума покаже тільки наявність таких, але без вказівки конкретних невідповідностей. Команда FC (File Compare) дозволяє порівнювати вміст двох файлів, шукаючи між ними несумісності. Допустимо, є два файли file1.docx та file2.docx, і необхідно їх порівняти.

Виконайте в консолі наступну команду: fc / U "D: file1.docx" "D: file2.docx".

У даному прикладі ми порівняли два прості текстових документа. Інтерпретатор командного рядка знайшов невідповідність в одному з речень і вказав конкретне місце (рис. 1.77).

Команда fc також дозволяє порівнювати бінарні файли, файли в кодировці юнікод, визначити кількість невідповідностей та ін. Якщо порівняні файли виявляться ідентичними, при виконанні команди fc буде виведено відповідне повідомлення.

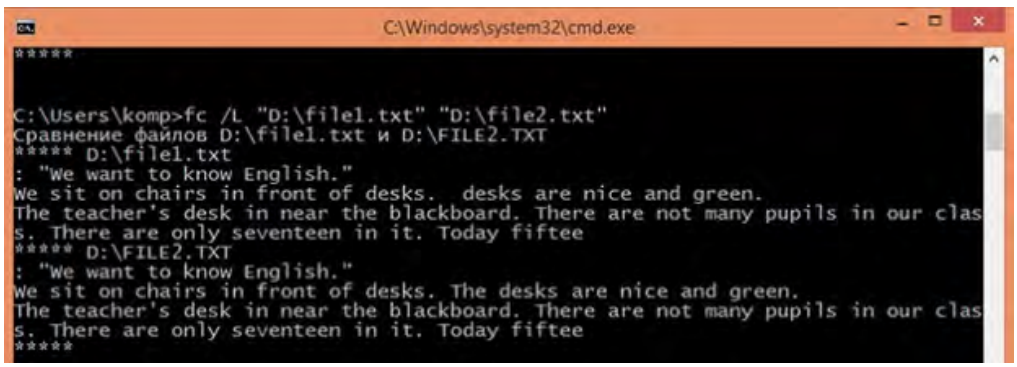


Рис. 1.77. Результат виконання команди fc

Для команди порівняння файлів fc можливі наступні ключі:

FC [/ A] [/ C] [/ L] [/ LBn] [/ N] [/ T] [/ W] [/ nnnn] [диск1:] [шлях1] ім'я_файла1 [диск2:] [шлях 2] ім'я_файла2

FC / B [диск1:] [путь1] ім'я_файла1 [диск2:] [путь2] ім'я_файла2

/ A Виводити тільки перший та останній рядок для кожної групи різниці.

/ B Порівняння двійкових файлів.

/ C Ігнорування регістру символів.

/ L Порівняння файлів у форматі ASCII.

/ LBn Максимальне число відмінностей для заданого числа строк.

/ N Виведення номерів строк при порівнянні текстових файлів ASCII.

/ T Символи табуляції не замінюються еквівалентним числом пробілів.

/ W При порівнянні пробілів і символів табуляції ігноруються.

/ nnnn Число послідовних рядків, що збігаються, які повинні бути після групи не співпадаючих.

Крім порівняння файлів також виконується порівняння дисків за допомогою команди DISKCOMP. Команда DISKCOMP застосовується для порівняння дисків, наприклад, копії та оригіналу. DISKCOMP автоматично визначає кількість строк і секторів, що підлягають порівнянню, за першим іменем диска в командному рядку. Команда має наступний синтаксис:

DISKCOMP ім'я диска1 ім'я диска2 [/ ключі]

Наприклад, A> DISKCOMP A: B: порівнює оригінал (A :) з копією (B :).

При відсутності на комп’ютері програм-архіваторів можна створити саморозпаковувальний архів з розширенням exe. В командному рядку вводять команду Iexpress, яка запускає

майстер створення архіву (рис. 1.78). Далі згідно з приведеною нижче інструкцією.

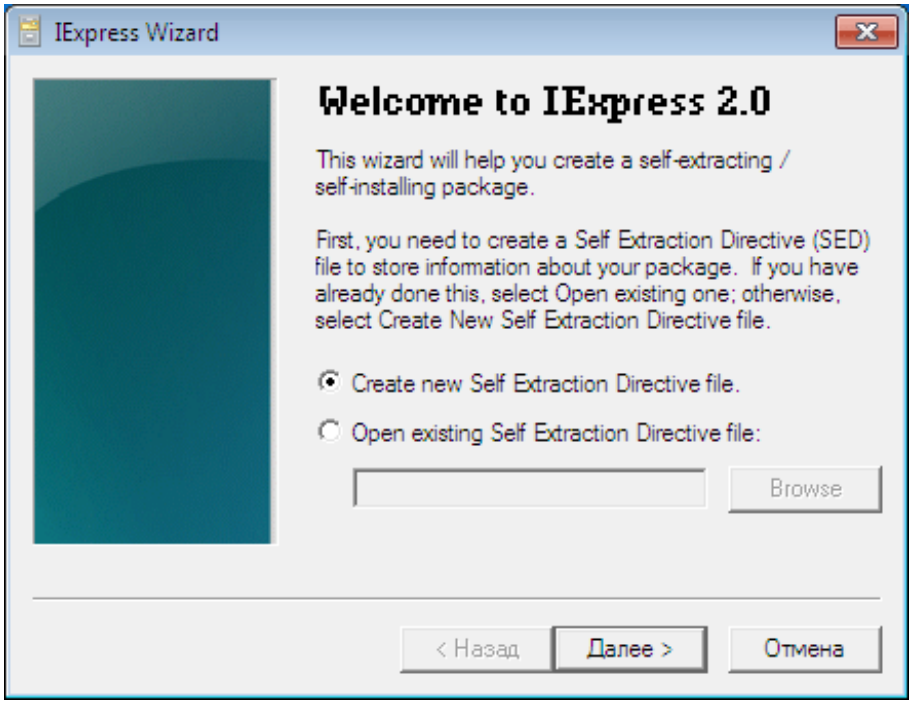


Рис. 1.78. Майстер створення архіву

1. Обрати пункт «Створити нову директиву» і натиснути «Далее»>.

2. Активувати пункт «Extract only files», натиснути «Далее», ввести назву архіву і натиснути «Далее»> до тих пір, поки на екрані не з'явиться вікно додавання файлів.

3. За допомогою кнопки «Додати» виберіть файли, які потрібно заархівувати, і в одному з наступних кроків вкажіть місце зберігання архіву.

4. Відмітити пункт «Не зберігати» і перейти до створення архіву за допомогою декількох натискань на «Далі».

1.7.2. Задання до самостійного виконання

1. Знайдіть в мережі 3 програмних продукти з зазначенням оригінальної контрольної суми. Завантажте програми і виконайте перевірку контрольної суми. Приведіть

підтвердження рівності контрольних сум і вкажіть причини їх можливих невідповідностей.

2. Створіть текстовий документ із вільним вмістом. Виконайте для нього розрахунок контрольної суми за алгоритмами MD5 і SHA1 в командному рядку. Відправте файли по електронній пошті і виконайте перевірку контрольних сум після отримання. Чи можна використати отриману контрольну суму як криптографічну хеш-функцію?

3. Створіть 2 текстові файли з однаковими вмістом. Внесіть зміни в один з них. Виконайте порівняння файлів за допомогою команди fc.

4. Створіть саморозпаковувальний архів і помістіть в нього робочі файли лабораторної роботи.

1.7.3. Запитання для самоперевірки

1. У чому полягає перевірка на цілісність прийнятої інформації за допомогою контрольної суми?

2. Наведіть відомі Вам способи обчислення контрольної суми.

3. Куди і коли додається контрольна сума при перевірці цілісності прийнятої інформації?

4. Що таке хеш-функція? Що окрім цілісності даних вона може підтвердити?

5. Яким чином можна не лише перевірити цілісність інформації, але і визначити відмінності у даних в разі відмінності контрольних сум двох файлів?

6. Для яких випадків і за допомогою якого засобу перевірять цілісність даних, переданих з одного жорсткого диска на інший? У чому полягає процес порівняння?

7. Яке розширення має готовий саморозпаковувальний архів. Поясніть це. Який стандартний засіб дозволяє його створити?

8. Виконайте порівняння розміру саморозпаковувального архіву з результатами архівації за алгоритмами rar і zip. Проаналізуйте і поясніть отриманий результат.

Тема 1.8. Виправлення помилки «брудний біт» у різних файлових системах

Мета: навитися визначати місцеположення і виправляти «брудний біт» у файлових системах FAT16, FAT32, NTFS.

1.8.1. Теоретичні відомості

Помилка в роботі змінного накопичувача часто виникає через так званий "брудний біт", який є особливою позначкою про необхідність перевірки накопичувача після підключення. Якщо "брудний біт" наявний на накопичувачі з будь-якою файловою системою, то ОС видає повідомлення, наведене на рис. 1.79. Причиною є незакінчені зміни через вимкнення комп'ютера до повної передачі даних на накопичувач або через виявлені пошкодження накопичувача.

Вирішити цю проблему для накопичувача з FAT16 можна двома способами:

1. Запустити перевірку накопичувача на справність через вікно, що з’явилось або через командний рядок і команду chkdsk / f. Після закінчення перевірки "брудний біт" має автоматично обнулятися, проте так відбувається не завжди. Крім того, при великому об’ємі накопичувача перевірка буде здійснюватися досить довгий час.

2. Гарантовано і швидко очистити «брудний біт» можна через шістнадцятковий редактор. Він знаходиться в PBR активного розділу за зміщенням 37_{10} від початку сектора і є одним з зарезервованих файловою системою (рис. 1.80).

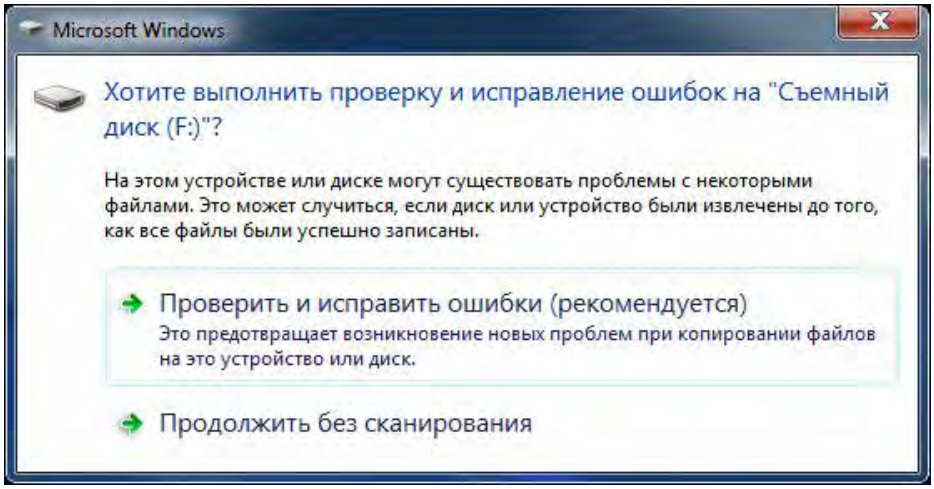


Рис. 1.79. Помилка «брудний біт» на зйомному накопичувачі з FAT16

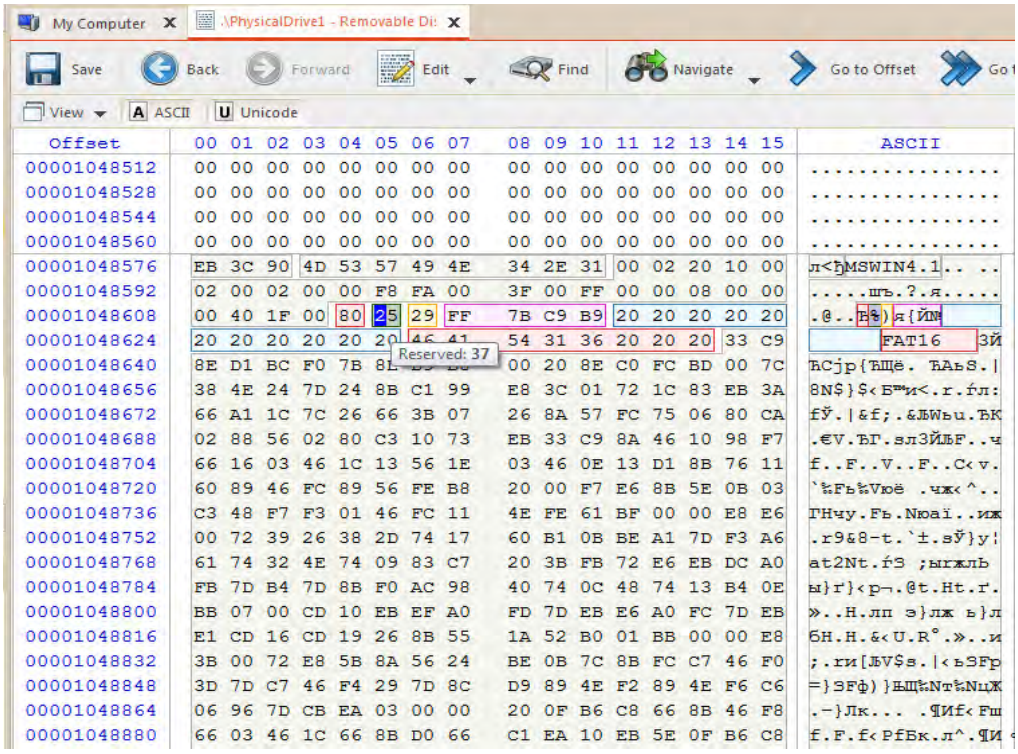


Рис. 1.80. «Брудний біт» в структурі PBR накопичувача з FAT16

Для того, щоб виправити помилку необхідно перейти в режим Read/Write і обнулити цей біт як показано на рис. 1.81.

Після збереження всіх змін, відключення накопичувача і його повторного включення, коректна робота буде відновлена.

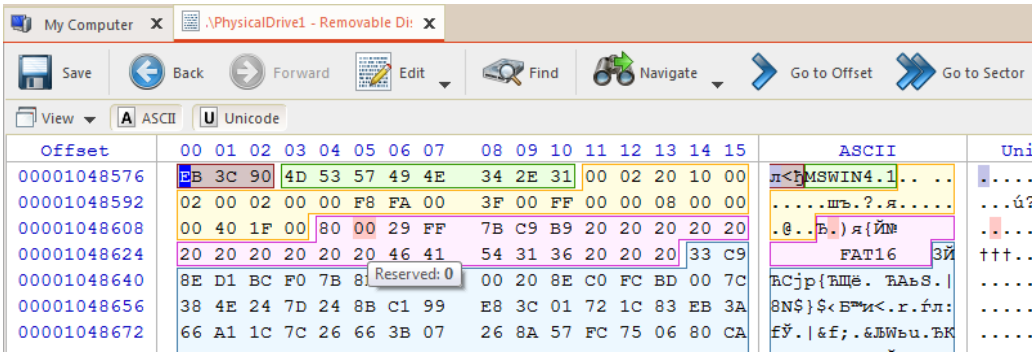


Рис. 1.81. Виправлення «брудного біта» на накопичувачі з FAT16

Для того, щоб впевнитися у відсутності «брудного біта» викликають системну утиліту **fsutil** в командному рядку і вводять команду "fsutil dirty query X:", де "X" – диск, що перевіряється (рис. 1.82).

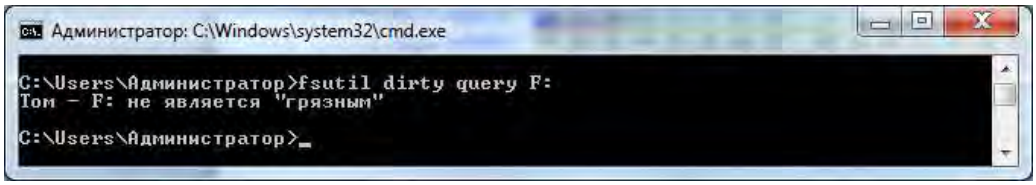


Рис. 1.82. Перевірка накопичувача після виправлення «брудного біта»

Для того, щоб змоделювати, а потім виправити «брудний біт» на накопичувачі з файловою системою FAT32 необхідно перейти в PBR активного розділу за зміщенням 65_{16} (41_{10}) від початку сектора. Цей біт є одним з зарезервованих файловою системою (рис. 1.83).

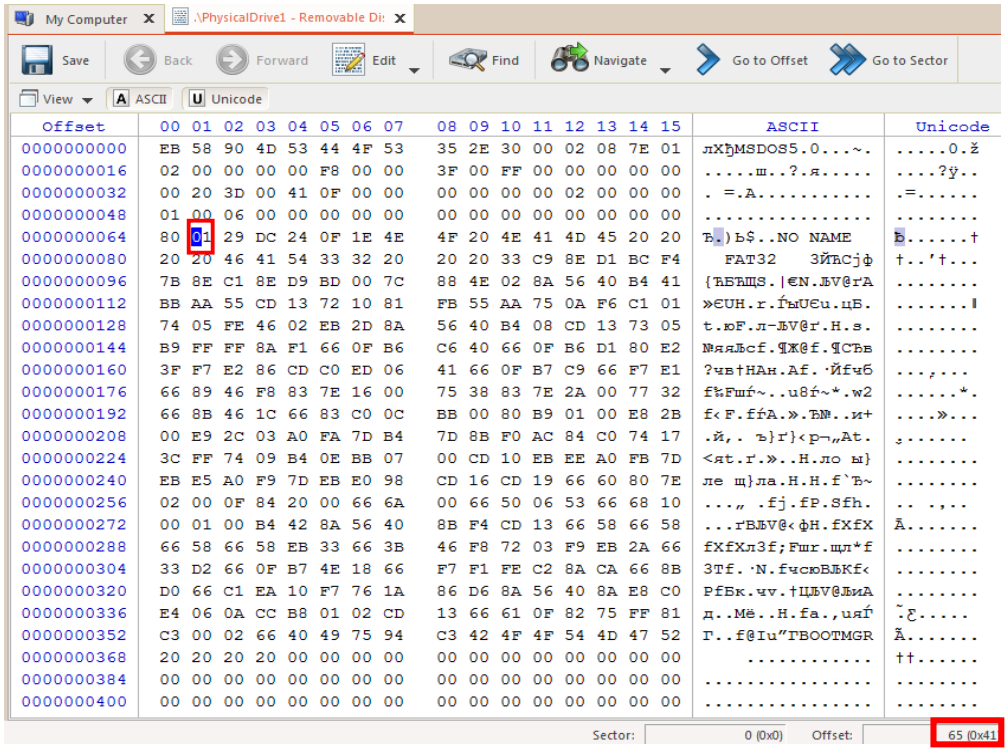


Рис. 1.83. Виявлення «брудного біта» на накопичувачі з FAT32

Для того, щоб виправити цю помилку необхідно перейти в режим Read/Write і обнулити цей біт як показано на рис. 1.84. Після збереження всіх змін, відключення накопичувача і його повторного включення, коректна робота буде відновлена.

Для того, щоб впевнитися у відсутності «брудного біта» викликають системну утиліту **fsutil** в командному рядку (рис. 1.85).

Файлова система NTFS на відміну від попередніх розглянутих має характерні відмінності. Детально структура NTFS буде розглянута на наступних лабораторних роботах. Зараз виконайте дії за наведеним нижче зразком.

При використанні утиліти BootIce в процесі форматування накопичувача в NTFS обирають USB/HDD mode (Single Partition) (рис. 1.86) і в результаті отримують один активний розділ з ідентифікатором файлової системи накопичувача 07 (рис. 1.87).

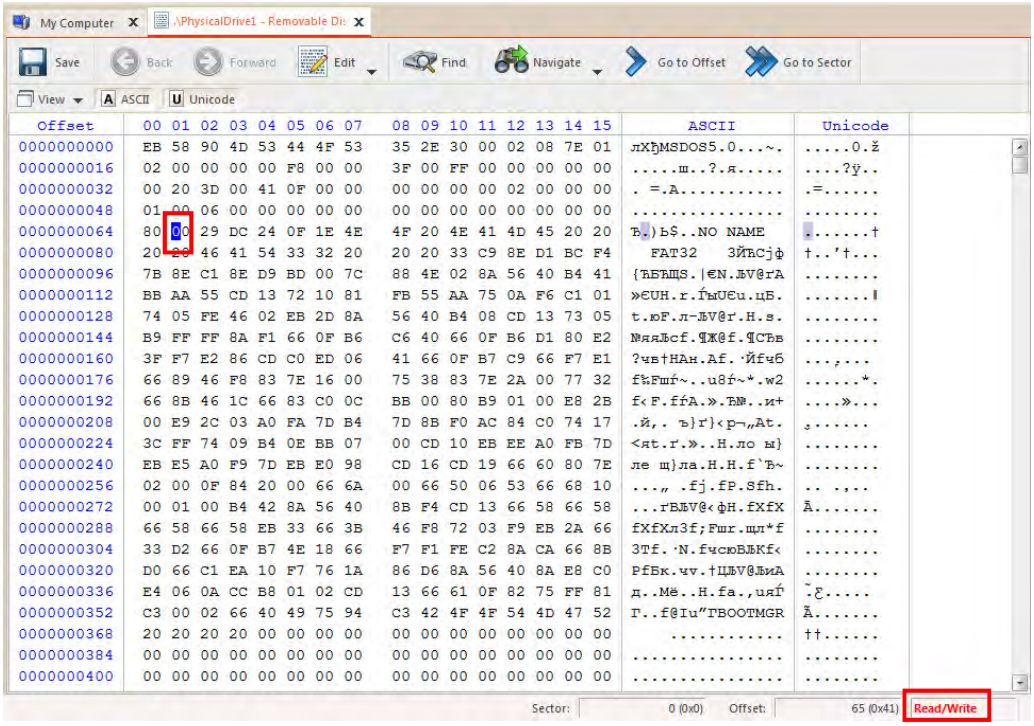


Рис. 1.84. Виправлення «брудного біта» на накопичувачі з FAT32

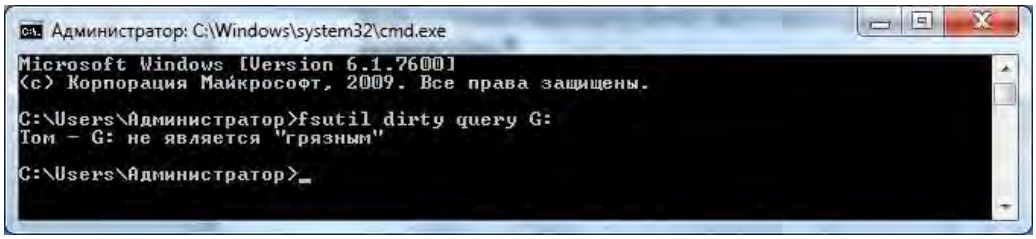


Рис. 1.85. Перевірка наявності «брудного біта» на накопичувачі з FAT32

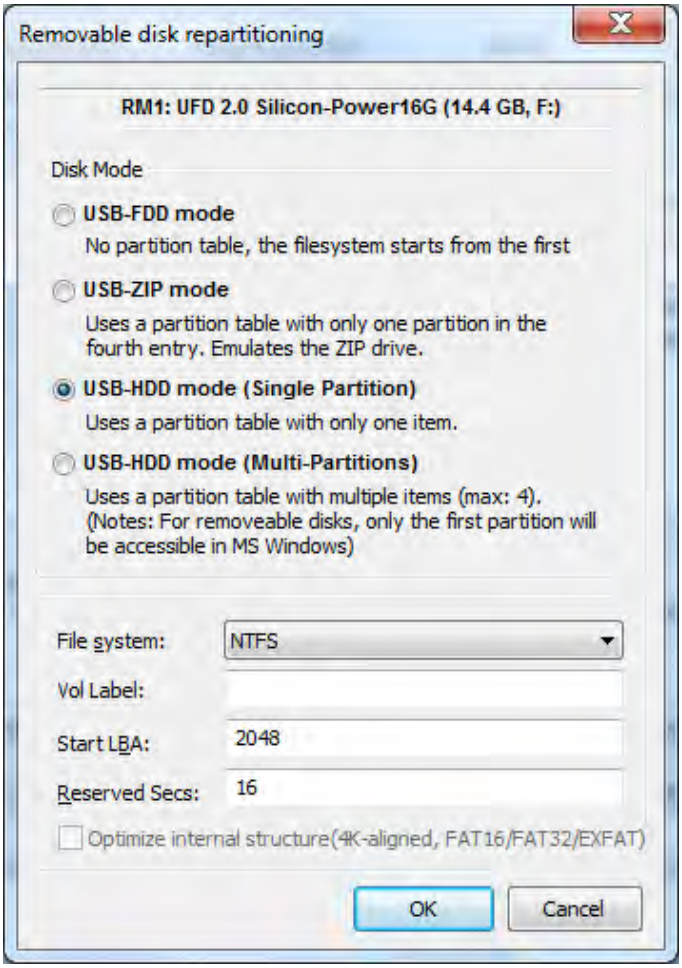


Рис. 1.86. Вибір кількості розділів при форматуванні накопичувача

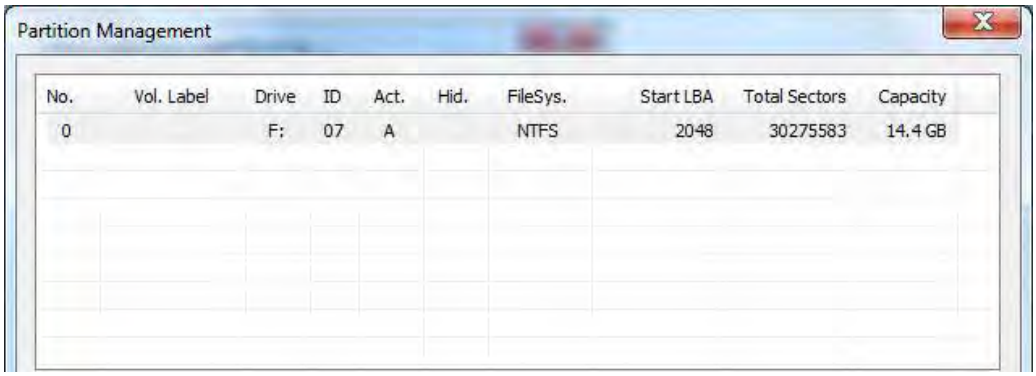


Рис. 1.87. Результат форматування накопичувача в файловій системі NTFS

У випадку з NTFS можна не тільки перевірити, але і змоделювати наявність «брудного біта» на накопичувачі за допомогою **fsutil**.

Для встановлення «брудного біта» на накопичувачі в командному рядку вводять "**fsutil dirty set X:**", де "X" – диск, на якому встановлюють «брудний біт» Одразу після цього виконують перевірку на наявність як у попередніх прикладах (рис. 1.88).

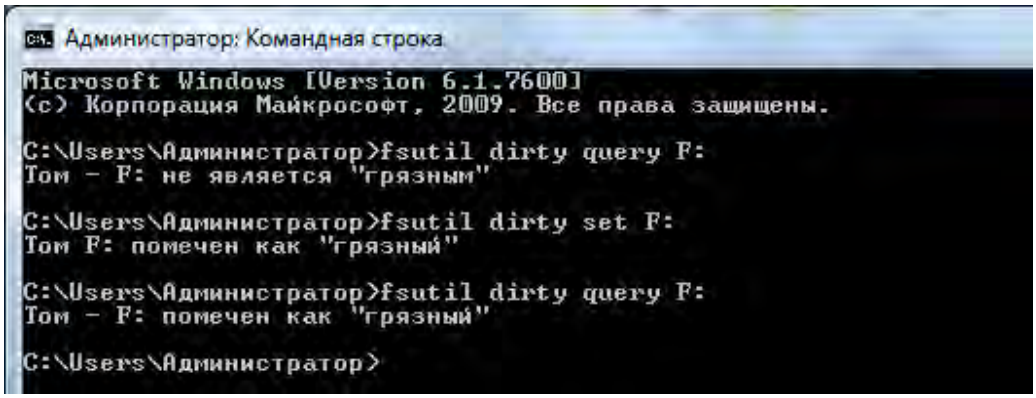


Рис. 1.88. Встановлення і перевірка наявності «брудного біта» на накопичувачі з NTFS

Фіксованого зміщення для пошуку «брудного біта» в NTFS не має, тому необхідно орієнтуватися на службовий файл $Volume, який знаходиться в головному записі MFT таблиці.

Знайти «брудний біт» на накопичувачі з NTFS в редакторі Active Disk Editor можливо, проте виправити неможна тому, що він не підтримує необхідне блокування томів в автоматичному режимі. Через це далі для виправлення помилки обраний інший шістнадцятковий редактор – DMDE. Він сумісний з останніми версіями Windows і забезпечує перегляд і редагування файлів і дискових структур (таблиці розділів MBR, GPT, завантажувальні сектори, таблиці, елементи директорій файлових систем), швидкі переходи між пов'язаними елементами, а також застосування шаблонів. Карта кластерів дозволяє визначати файли по їх розташуванню на диску.

Для пошуку «брудного біту» в Active Disk Editor переходимо в меню Navigate і обираємо перехід до $MFT – головної файлової таблиці в NTFS, наближений аналог FAT у FAT16 і FAT32 (рис. 1.89).

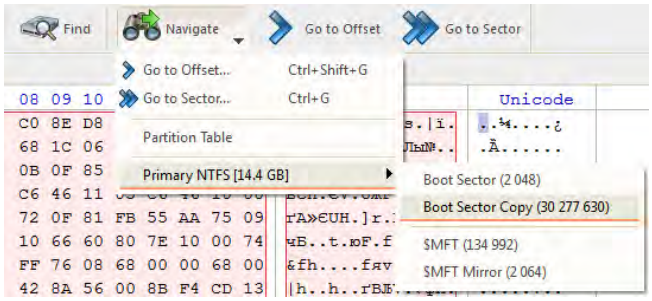


Рис. 1.89. Перехід до $MFT

Головна таблиця MFT складається з 16 службових файлів. Для виправлення «брудного біта» необхідного знайти в ній третій за порядком файл, який називається $Volume шляхом переміщення повзунка прокрутки донизу, поки у стовпці Unicode не з’явиться ім’я файлу $Volume (рис. 1.90).

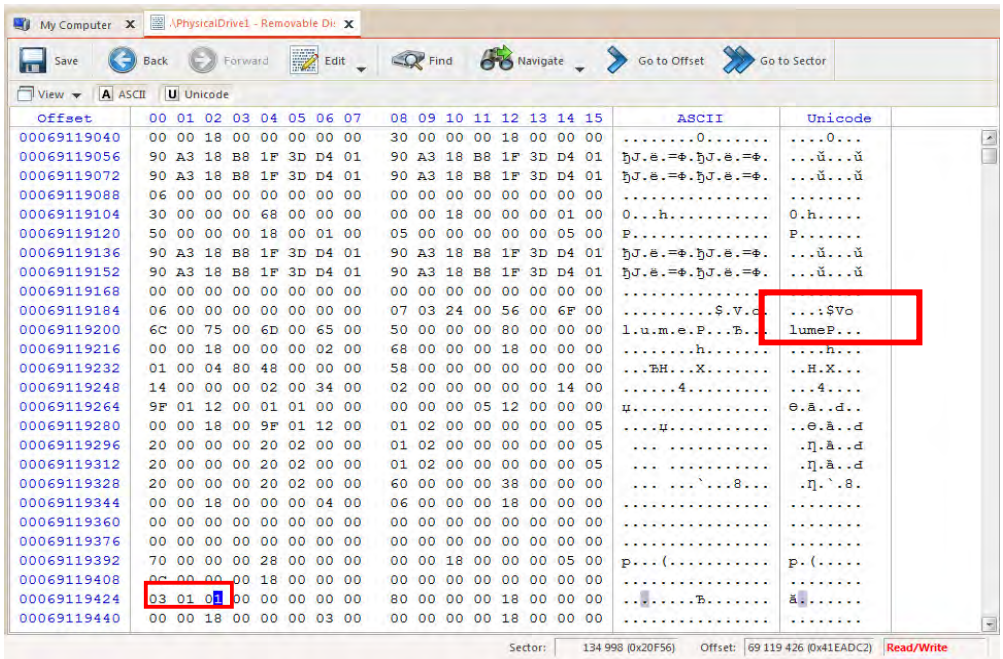


Рис. 1.90. Перехід до службового файлу $Volume

Після переходу виписують номер сектора, в якому знаходиться файл $Volume. На рис. 12 видно, що це сектор із номером 134998.

Мітка «брудного біту» включає послідовність 030101, яку видно на рис. 1.91 трохи нижче ніж вказано ім'я файлу. Для усунення «брудного біту» останню 1 необхідно змінити на 0. При спробі зробити це в редакторі Active Disk Editor виникає помилка, наведена на рис. 1.91, тому далі переходять в редактор DMDE.

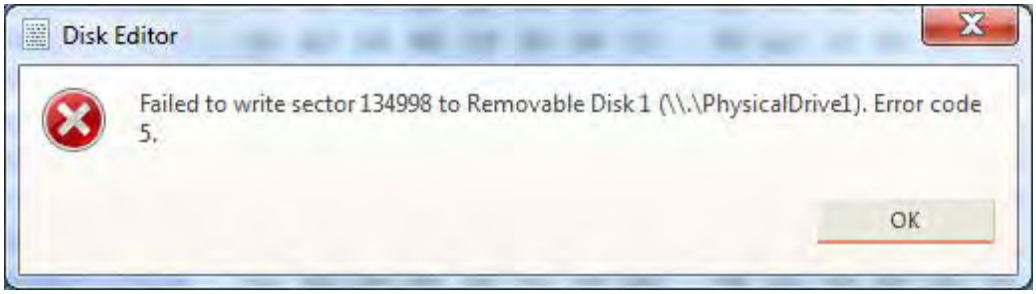


Рис. 1.92. Помилка редагування «брудного біта» в Active Disk Editor

Для того, щоб завантажити редактор DMDE переходять за посиланням https://dmde.ru/download.html і обирають «Завантажити DMDE для Windows». DMDE не потребує встановлення і запускається одразу після розпакування архіву. При відкритті редактора зі списку праворуч обирають потрібний накопичувач, а ліворуч встановлюють прапор тільки для фізичних пристроїв (рис. 1.92).

Після того, як накопичувач відкритий переходять на вкладку «Режим» і обирають першу в списку команду – «Шістнадцятковий/Текст» (рис. 1.93). Після цього структура набуває звичного шістнадцяткового виду.

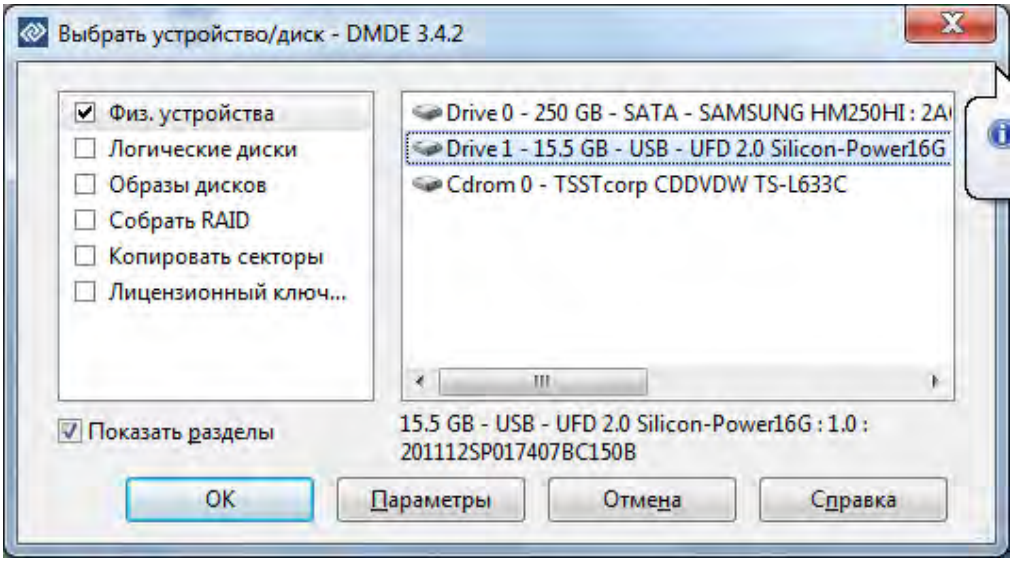


Рис. 1.92. Вибір пристроїв в редакторі DMDE

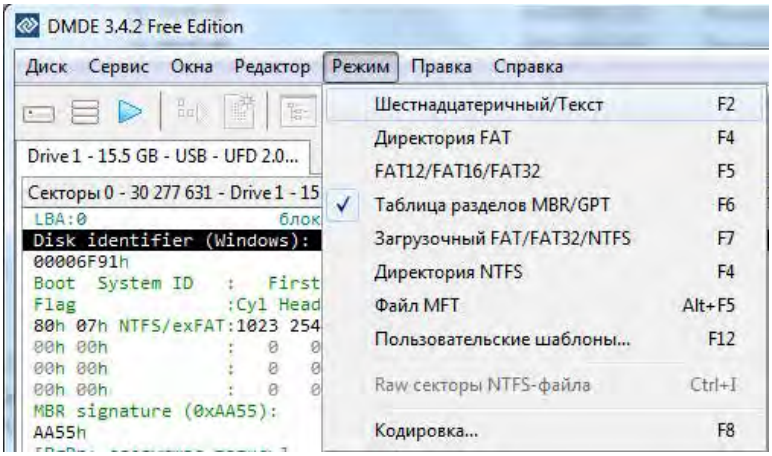


Рис. 1.93. Зміна режимів перегляду структури накопичувачів

Для переходу в сектор з «брудним бітом» обирають вкладку «Редактор», а далі – команду «Фізичні сектори». У вікні, що відкрилося, у першому рядку вводять номер раніше знайденого сектора – 134998 (рис. 1.94). Зверніть увагу на те, що перемикач Dec/Hex має бути в положенні Dec.

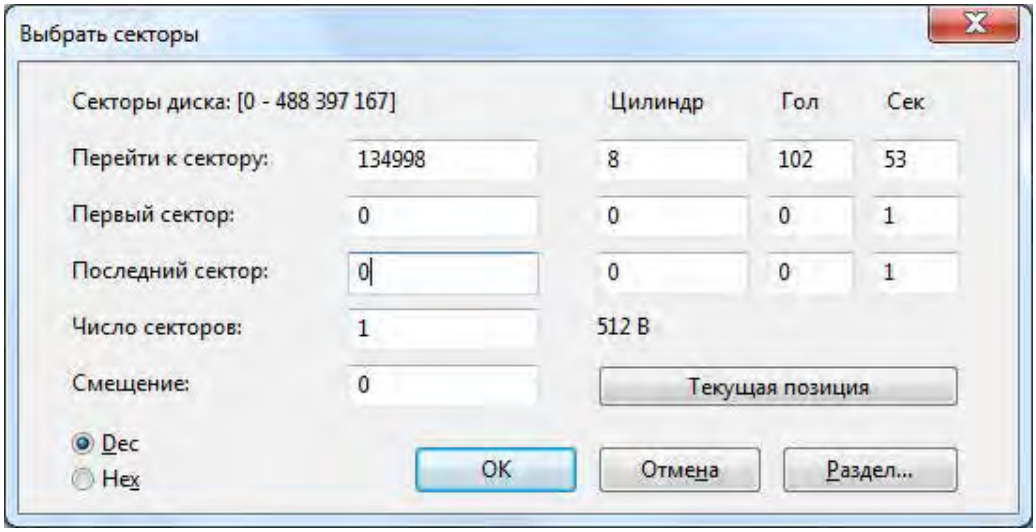


Рис. 1.94. Перехід в визначений сектор

У секторі 134998 знаходять послідовність характерну для «брудного біту» – 030101 і виправляють її. Для переходу в режим редагування використовують сполучення клавіш Ctrl+E. Під час пошуку орієнтуватися необхідно на імя файлу $Volume. Послідовність завжди знаходиться на декілька рядків нижче (рис. 1.95).

Після того, як «брудний біт» виправлено, переходять в меню «Диск» і обирають пункт «Зберегти зміни». Для того, щоб переконатися в правильності всіх дій виконують перевірку накопичувач на «брудний біт» в командному рядку (рис. 1.96).

1.8.2. Завдання для самостійного виконання

1. Підготувати флеш-накопичувач з файловою системою FAT16.

2. Змоделювати на флеш-накопичувачі помилку «брудний біт». Виконати перевірку наявності «брудного біту» на флеш-накопичувачі через командний рядок.

3. Виправити помилку і виконати перевірку відсутності «брудного біту» на флеш-накопичувачі через командний рядок.

4. Виконати пункти 1–3 для флеш-накопичувачів з файловими системами FAT32 і NTFS.

5. Оформити звіт, що містить підтвердження результатів виконання роботи у виді скріншотів.

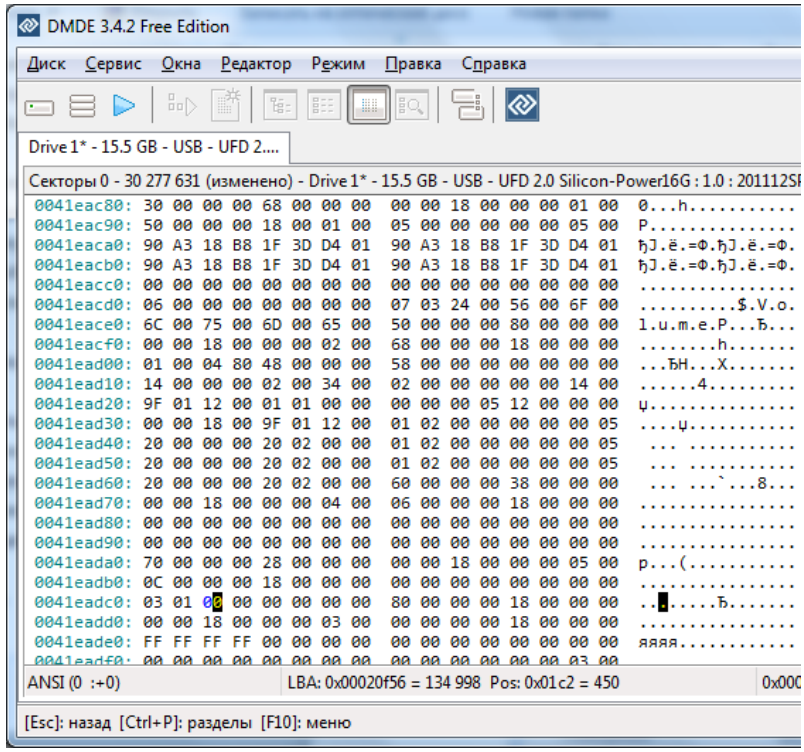


Рис. 1.95. Послідовність «Брудний біт»

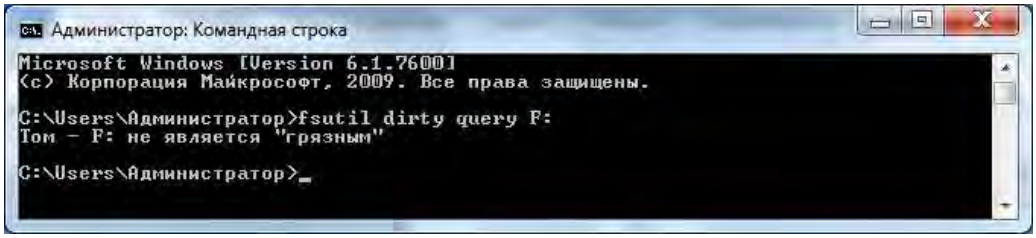


Рис. 1.96. Підтвердження виправленого «брудного біту»

1.8.3. Запитання для самоперевірки

1. Який ідентифікатор файлової системи має NTFS?
2. Дайте визначення поняття «брудний біт». Якими способами «брудний біт» може бути виправлений?
3. У якій структурній частині міститься помилка «брудний біт» для кожної з файлових систем?

4. Як перевірити наявність і відсутність брудного біту за допомогою командного рядка?
5. Поясніть причини того, що Active Disk Editor не редагує помилку «брудний біт».

Тема 1.9. Відновлення даних з використанням таблиці FAT

Мета: Навчитися визначати належні файлу кластери в таблиці FAT та відновлювати файли з даними після видалення за допомогою записів в таблиці FAT.

1.9.1. Теоретична відомості

Робота із таблицею FAT. Як відомо таблиця FAT складається з ланцюжків номерів кластерів, що належать наявним на накопичувачі файлам. Відправною точкою є номер першого кластера файла, який знаходиться в записі про файл в кореневому або створеному каталозі. Пошук номера першого кластера файла виконують в шістнадцятковому редакторі, наприклад, Active Disk Editor за відомим алгоритмом.

Для цього спочатку знаходимо в MBR-секторі перший розділ в Partition Table, а в ньому байти, які відповідають за номер першого сектора розділу (рис. 1.97).

Після переходу в 2048 сектор – PBR першого розділу знаходимо байти, що відповідають за кількість секторів на одну таблицю FAT, які згідно з логічною структурою накопичувача необхідно врахувати для переходу в кореневий каталог (рис. 1.98).

Оскільки далі працювати будемо як з кластерами, так і з секторами, тому випишемо значення відповідних байт для цього співвідношення (рис. 1.99). Останньою з PBR випишемо кількість резервних секторів (рис. 1.100).

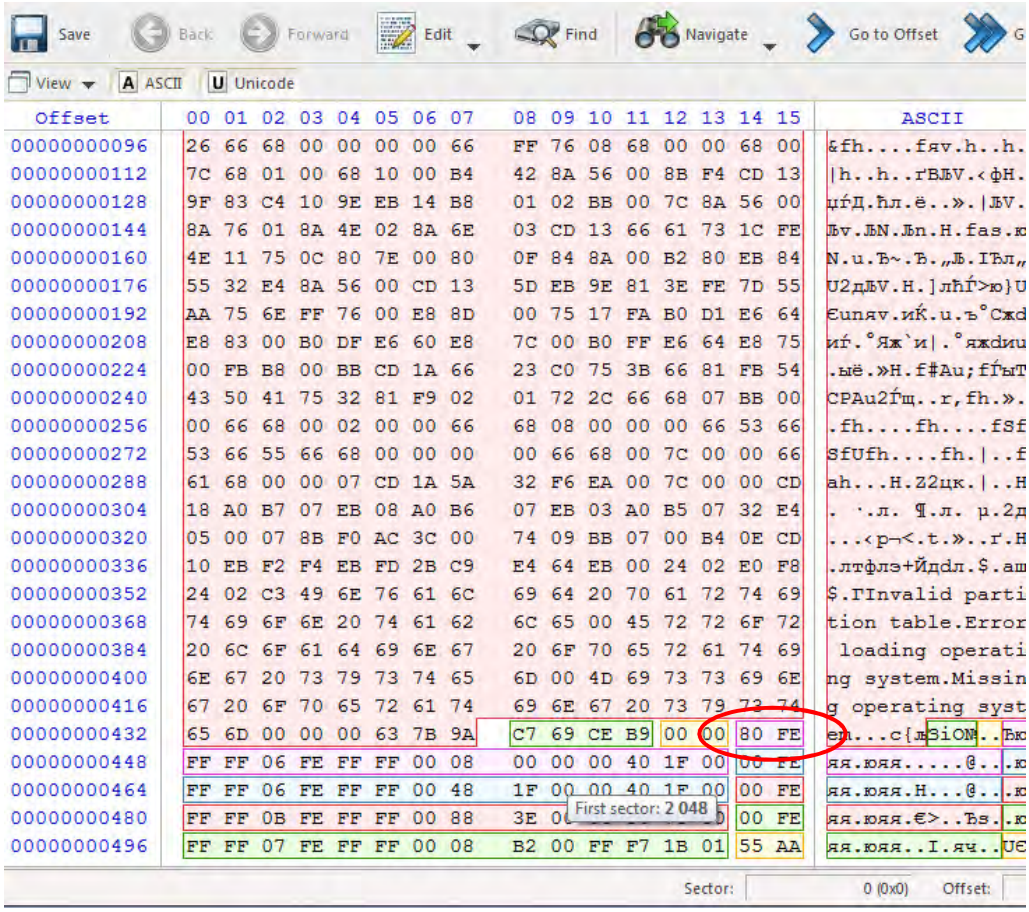


Рис. 1.97. Визначення першого сектора першого розділу накопичувача

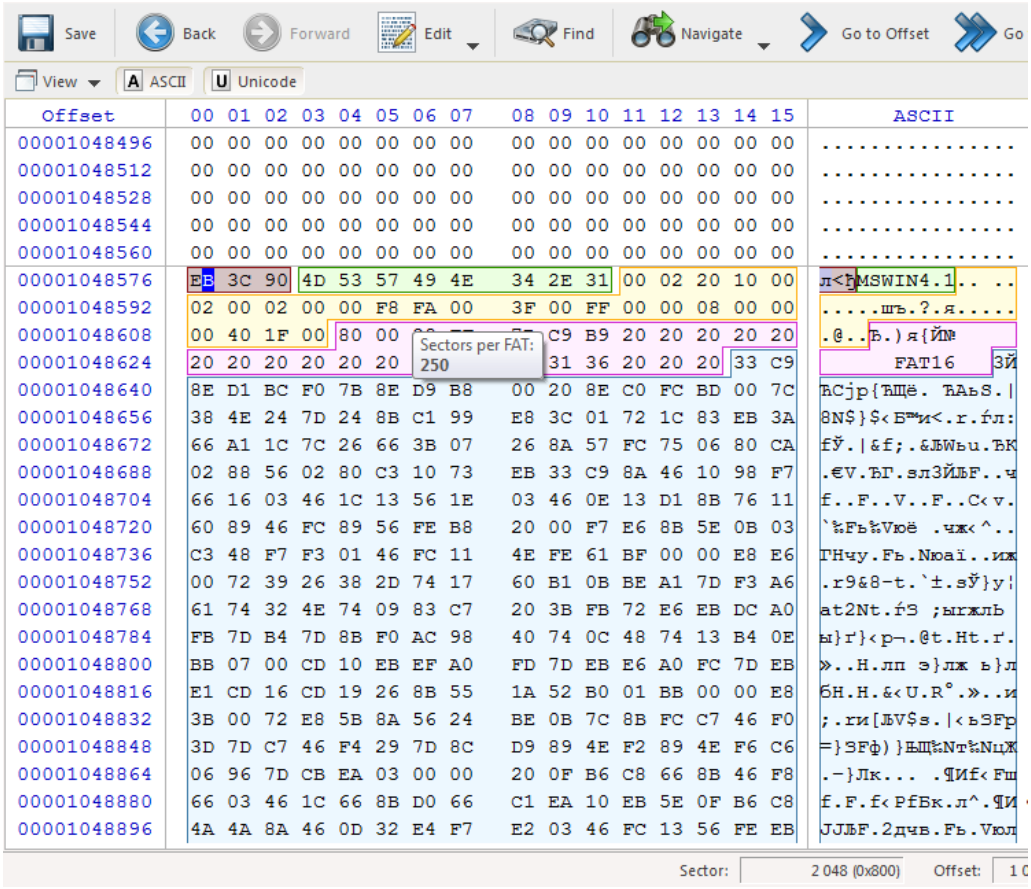


Рис. 1.98. Визначення кількості секторів на одну таблицю FAT

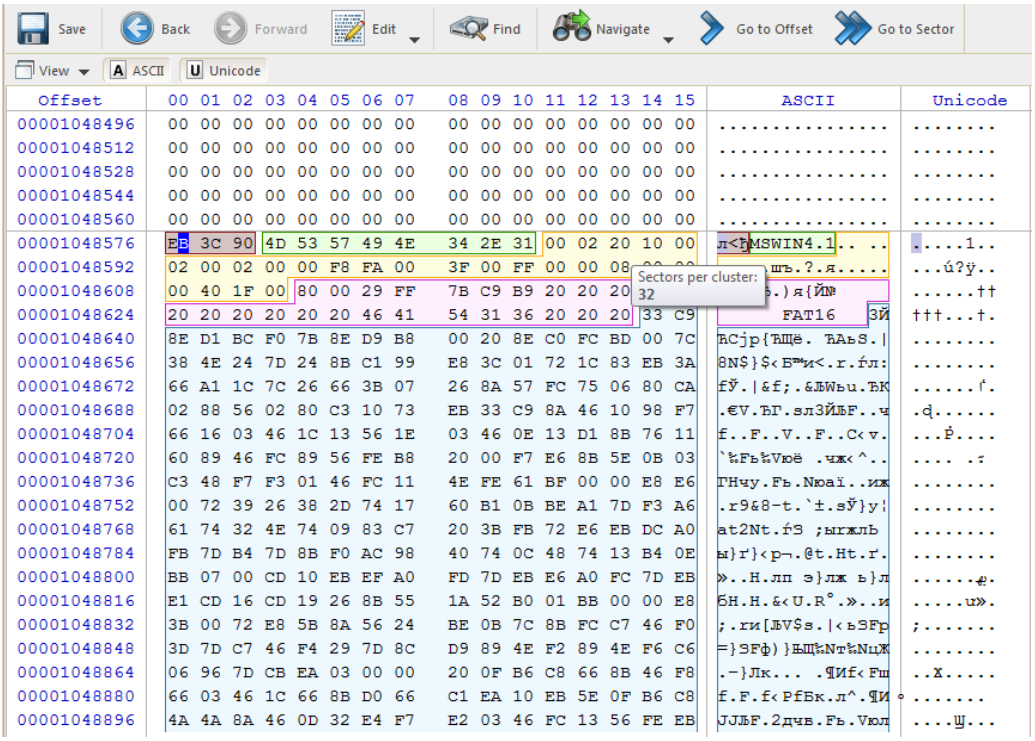


Рис. 1.99. Визначення кількості секторів на кластер

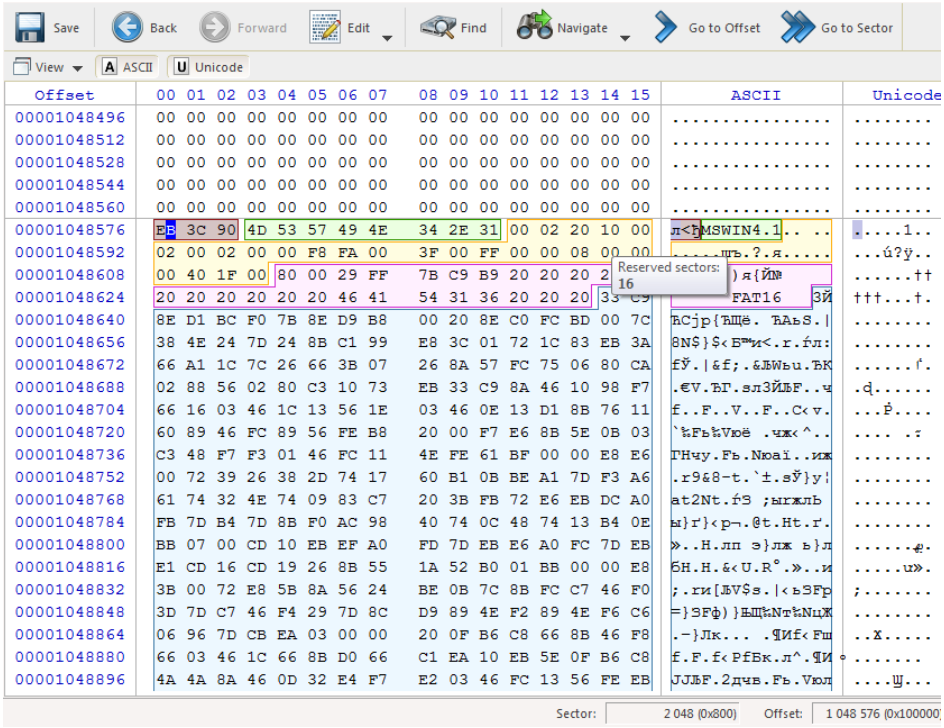


Рис. 1.100. Визначення кількості резервних секторів

Таким чином, знайдемо початковий сектор кореневого каталогу за відомою формулою і підставимо в неї всі значення: ROOT = PBR+2FAT+RESERVED = 2048+2*250+16=2564 (сектор).

Починаючи з першого сектора кореневого каталогу шукаємо в стовбці ASCII ім'я файлу Lection_NTFS.doc, взятого для прикладу. Після знайдених короткого і довгого імен файлу, визначають початковий кластер змісту файлу (рис. 1.101) і переходять в FAT1 через меню Navigate.

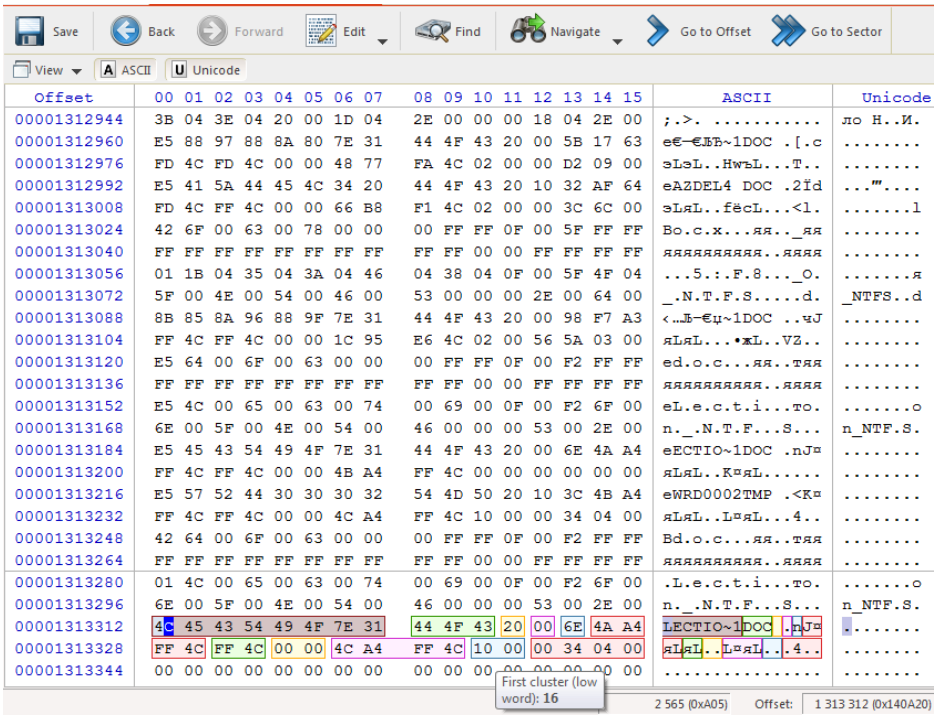
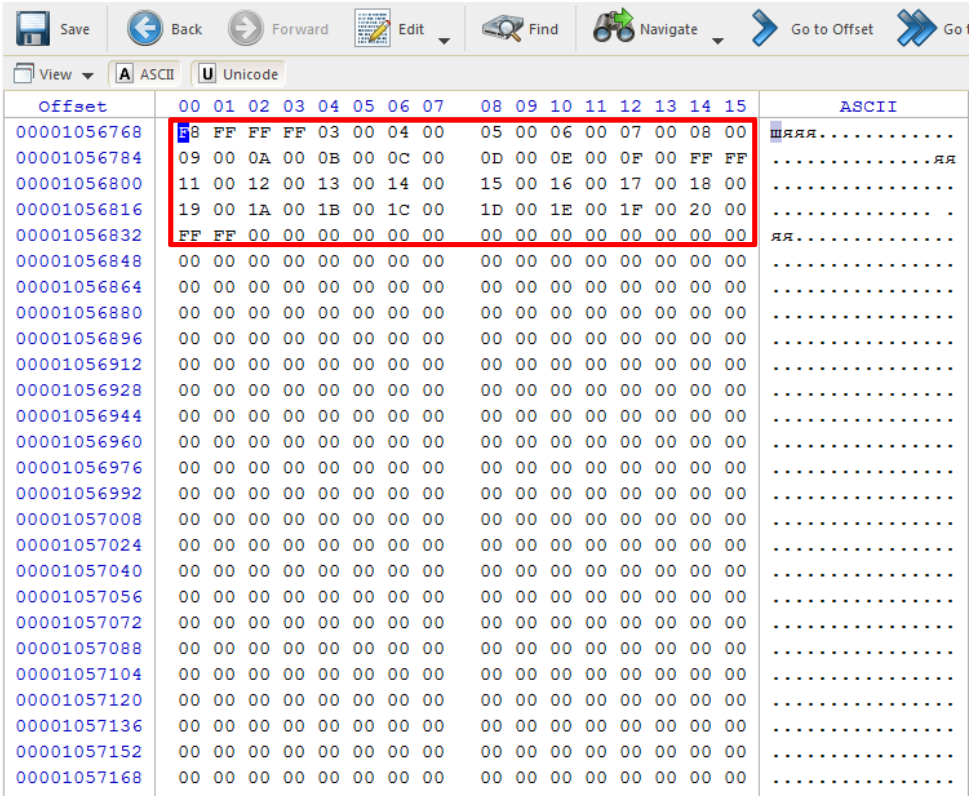


Рис. 1.101. Визначення першого кластера файла

FAT1 дозволяє виписати повний ланцюжок кластерів для досліджуваного файлу. Відомо, що перший кластер файлу – 16. У ньому за технологією зв’язних списків записані номера наступних кластерів ланцюжка. FAT1 наведена на рис 1.102. Одна з технологій відновлення даних на накопичувачі полягає в періодичному копіюванні зберіганні FAT. На даному етапі це необхідно зробити також для виконання останньої частини роботи.



Offset	00 01 02 03 04 05 06 07	08 09 10 11 12 13 14 15	ASCII
00001056768	F8 FF FF FF 03 00 04 00	05 00 06 00 07 00 08 00	шяяя............
00001056784	09 00 0A 00 0B 00 0C 00	0D 00 0E 00 0F 00 FF FF	..............яя
00001056800	11 00 12 00 13 00 14 00	15 00 16 00 17 00 18 00	....
00001056816	19 00 1A 00 1B 00 1C 00	1D 00 1E 00 1F 00 20 00	....
00001056832	FF FF 00 00 00 00 00 00	00 00 00 00 00 00 00 00	яя..............
00001056848	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056864	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056880	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056896	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056912	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056928	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056944	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056960	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056976	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001056992	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057008	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057024	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057056	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057072	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057088	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057104	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057120	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057136	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057152	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....
00001057168	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	....

Рис. 1.102. Таблиця FAT1

Випишемо з FAT1 повний ланцюжок кластерів файлу: 11_{16} – 12_{16} – 13_{16} –14_{16} – 15_{16} – 16_{16} – 17_{16} – 18_{16} – 19_{16} – $1A_{16}$ – $1B_{16}$ – $1C_{16}$ – $1D_{16}$ – $1E_{16}$ – $1F_{16}$ – 20_{16} – FFFF.

В 10-ій системі числення отриманий ланцюжок приведений в табл. 1.

Додавши раніше відомий номер першого кластера, отримаємо наступне: 16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-FF.

Кількість кластерів в ланцюжку дорівняє 17. Значить, можна розрахувати розмір файлу: $V_ф$ = 17 (кластерів) * 32 (сектора) * 512 (байт / сектор) = 278528 (байт) або 278,528 (кБайт). Правильність обчислень перевіряємо у властивостях файлу (рис. 1.103).

Таблиця 1

Ланцюг кластерів файлу (в 10-ій системі числення)

$0011_{16} = 17_{10}$	$0019_{16} = 25_{10}$
$0012_{16} = 18_{10}$	$001A_{16} = 26_{10}$
$0013_{16} = 19_{10}$	$001B_{16} = 27_{10}$
$0014_{16} = 20_{10}$	$001C_{16} = 28_{10}$
$0015_{16} = 21_{10}$	$001D_{16} = 29_{10}$
$0016_{16} = 22_{10}$	$001E_{16} = 30_{10}$
$0017_{16} = 23_{10}$	$001F_{16} = 31_{10}$
$0018_{16} = 24_{10}$	$0020_{16} = 32_{10}$
FFFF – последний кластер.	

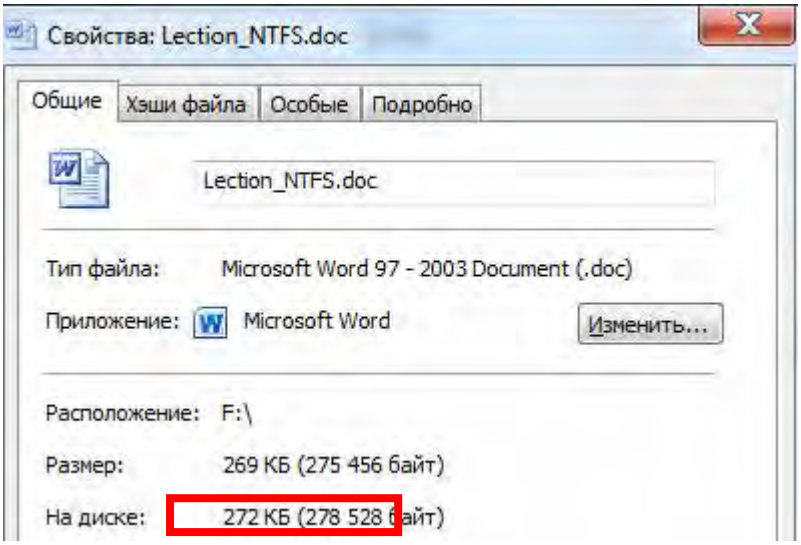


Рис. 1.103. Перевірка розміру файла

Після цього виконують перевірку ланцюжка кластерів шляхом переходу в кожний кластер.

1-ий кластер (16-ий).

Для зчитування записаної в цей кластер інформації виконують розрахунок за формулою (1):

$$NS = ROOT+(N-1)*SK, \qquad (1.1)$$

де NS – номер сектора з даними;

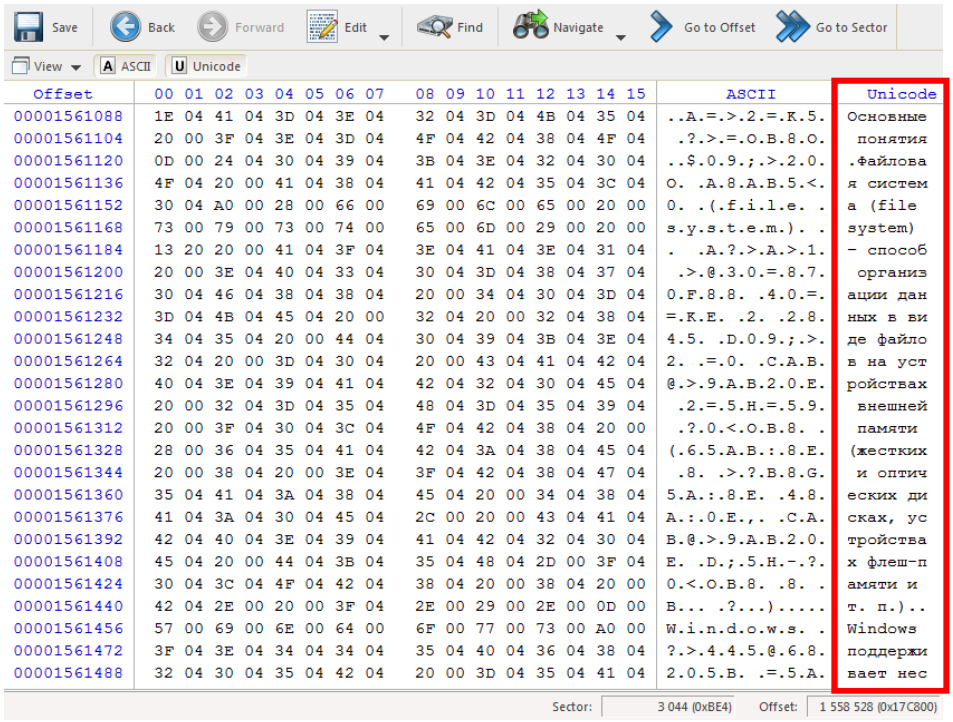
ROOT – номер сектора кореневого каталогу;

N – номер кластера ланцюжка в десятковій системі числення;

SK – кількість секторів на кластер.

Отже, отримаємо NS = 2564+(16-1)*32=3044 (сектор).

Номер сектора коректний, оскільки знайдені дані відповідають початку змісту файлу (рис. 1.104, 1.105).



Offset	00 01 02 03 04 05 06 07	08 09 10 11 12 13 14 15	ASCII	Unicode
00001561088	1E 04 41 04 3D 04 3E 04	32 04 3D 04 4B 04 35 04	..A.=.>.2.=.K.5.	Основные
00001561104	20 00 3F 04 3E 04 3D 04	4F 04 42 04 38 04 4F 04	.?.>.=.O.B.8.O.	понятия
00001561120	0D 00 24 04 30 04 39 04	3B 04 3E 04 32 04 30 04	..$.0.9.;.>.2.0.	.Файлова
00001561136	4F 04 20 00 41 04 38 04	41 04 42 04 35 04 3C 04	O. .A.8.A.B.5.<.	я систем
00001561152	30 04 A0 00 28 00 66 00	69 00 6C 00 65 00 20 00	0. .(.f.i.l.e. .	а (file
00001561168	73 00 79 00 73 00 74 00	65 00 6D 00 29 00 20 00	s.y.s.t.e.m.). .	system)
00001561184	13 20 20 00 41 04 3F 04	3E 04 41 04 3E 04 31 04	. .A.?.>.A.>.1.	- способ
00001561200	20 00 3E 04 40 04 33 04	30 04 3D 04 38 04 37 04	.>.@.3.0.=.8.7.	организ
00001561216	30 04 46 04 38 04 38 04	20 00 34 04 30 04 3D 04	0.F.8.8. .4.0.=.	ации дан
00001561232	3D 04 4B 04 45 04 20 00	32 04 20 00 32 04 38 04	=.K.E. .2. .2.8.	ных в ви
00001561248	34 04 35 04 20 00 44 04	30 04 39 04 3B 04 3E 04	4.5. .D.0.9.;.>.	де файло
00001561264	32 04 20 00 3D 04 30 04	20 00 43 04 41 04 42 04	2. .=.0. .C.A.B.	в на уст
00001561280	40 04 3E 04 39 04 41 04	42 04 32 04 30 04 45 04	@.>.9.A.B.2.0.E.	ройствах
00001561296	20 00 32 04 3D 04 35 04	48 04 3D 04 35 04 39 04	.2.=.5.H.=.5.9.	внешней
00001561312	20 00 3F 04 30 04 3C 04	4F 04 42 04 38 04 20 00	.?.0.<.O.B.8. .	памяти
00001561328	28 00 36 04 35 04 41 04	42 04 3A 04 38 04 45 04	(.6.5.A.B.:.8.E.	(жестких
00001561344	20 00 38 04 20 00 3E 04	3F 04 42 04 38 04 47 04	.8. .>.?.B.8.G.	и оптич
00001561360	35 04 41 04 3A 04 38 04	45 04 20 00 34 04 38 04	5.A.:.8.E. .4.8.	еских ди
00001561376	41 04 3A 04 30 04 45 04	2C 00 20 00 43 04 41 04	A.:.0.E.,. .C.A.	сках, ус
00001561392	42 04 40 04 3E 04 39 04	41 04 42 04 32 04 30 04	B.@.>.9.A.B.2.0.	тройства
00001561408	45 04 20 00 44 04 3B 04	35 04 48 04 2D 00 3F 04	E. .D.;.5.H.-.?.	х флеш-п
00001561424	30 04 3C 04 4F 04 42 04	38 04 20 00 38 04 20 00	0.<.O.B.8. .8. .	амяти и
00001561440	42 04 2E 00 20 00 3F 04	2E 00 29 00 2E 00 0D 00	B... .?...).....	т. п.)..
00001561456	57 00 69 00 6E 00 64 00	6F 00 77 00 73 00 A0 00	W.i.n.d.o.w.s. .	Windows
00001561472	3F 04 3E 04 34 04 34 04	35 04 40 04 36 04 38 04	?.>.4.4.5.@.6.8.	поддержи
00001561488	32 04 30 04 35 04 42 04	20 00 3D 04 35 04 41 04	2.0.5.B. .=.5.A.	вает нес

Рис. 1.104. Зміст першого кластеру файлу (фрагмент)

Основные понятия

Файловая система (*file system*) – способ организации данных в виде файлов на устройствах внешней памяти (жестких и оптических дисках, устройствах флеш-памяти и т. п.).

Windows поддерживает несколько файловых систем для различных внешних устройств:

- NTFS – основная файловая система семейства Windows NT;
- FAT (File Allocation Table – таблица размещения файлов) – простая файловая система используемая Windows для устройств флеш памяти, а также для совместимости с другими операционными системами при установке на диски с множественной загрузкой. Основным элементом этой файловой системы является таблица размещения файлов FAT (по имени которой названа вся файловая система), необходимая для определения расположения файла на диске. Существует три варианта FAT, отличающихся разрядностью идентификаторов, указывающих размещение файлов: FAT12, FAT16 и FAT32;
- exFAT (Extended FAT – расширенная FAT) – развитие файловой системы FAT, использующее 64 разрядные идентификаторы. Применяется в основном для устройств флеш-памяти;
- CDFS (CD ROM File System) – файловая система для CD дисков, объединяющая форматы ISO 9660[1] и Joliet[2];
- UDF (Universal Disk Format – универсальный формат дисков) – файловая система для

Рис. 1.105. Порівняння інформації в першому кластері і в файлі

Аналогічно по формулі (1) виконаємо розрахунки для наступних кластерів файлу.

2-ий кластер ($11_{16}=17_{10}$)

NS = 2564+(17-1)*32=3076 (сектор).

Інформація, записана в 2-ий кластер файлу, наведена на рис. 1.106.

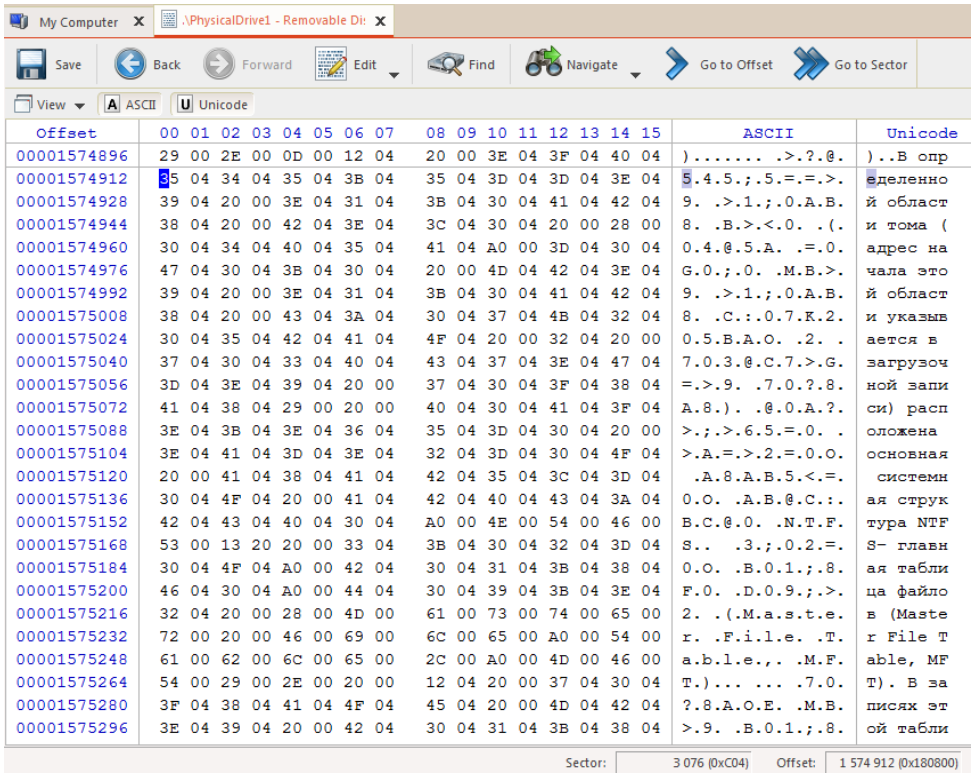


Рис. 1.106. Інформація 2-го кластеру файлу (фрагмент)

3-ій кластер ($12_{16}=18_{10}$)

NS = 2564+(18-1)*32=3108 (сектор).

Інформація, записана в 3-ій кластер файлу, наведена на рис. 1.107.

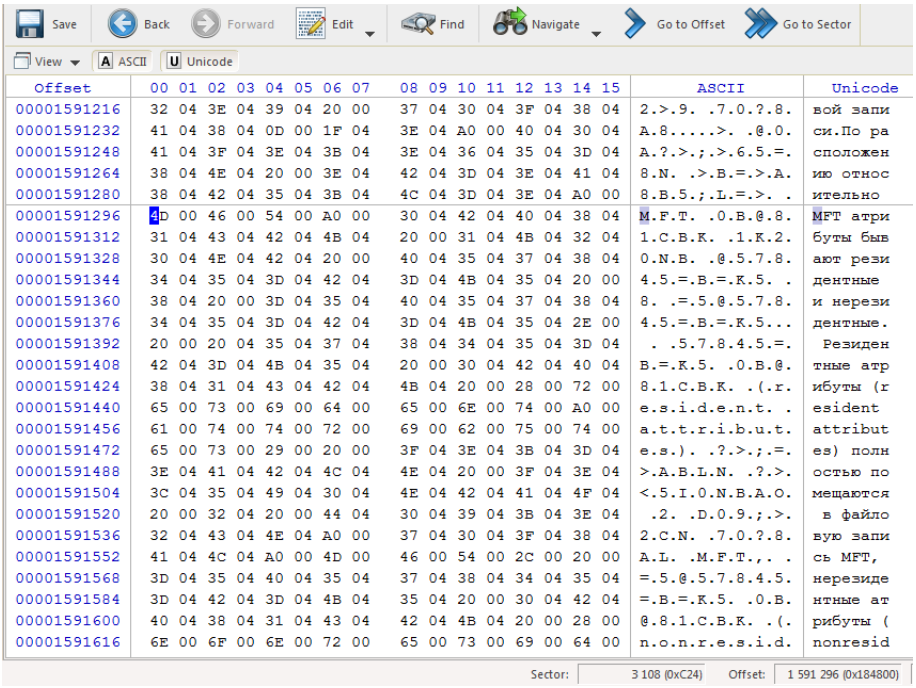


Рис. 1.107. Інформація 3-го кластеру файлу (фрагмент)

Для наступних кластерів алгоритм буде аналогічним, тому розрахунки і рисунки не наведені.

Відновлення видалених файлів. Далі розглянемо випадкове, не навмисне, видалення файлів з інформацією, яку треба відновити. Перше правило після помилкового видалення даних – нічого не записувати і не видаляти з накопичувача, тому що це змінить таблицю FAT.

Припустимо, що файл Lection_NTFS.doc був помилково видалений. Про це свідчить мітка E5 в кореневому каталозі (рис. 1.108).

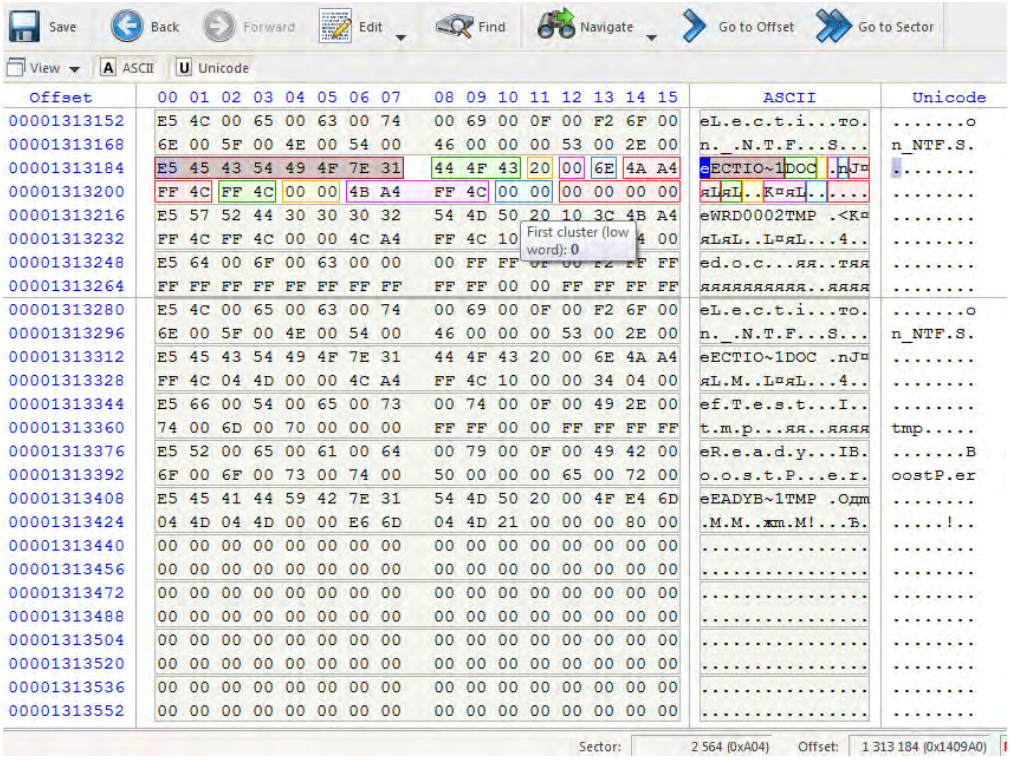


Рис. 1.108. Видалений файл з міткою E5

Для видалених файлів характерно спотворення їх імен в стовпці ASCII і заповнення.

Для відновлення даних переходимо в режим Read/Write і виправляємо ім'я файлу. Після цього мітка E5 заміщується відповідним кодом букв в ASCII (рис. 1.109).

Після цього переходять в FAT1, в якій тепер замість записів про номери кластерів файлу Lection_NTFS.doc знаходяться нулі (рис. 1.110).

Оскільки, заздалегідь була зроблена копія FAT1, то відновлюють записи в ній згідно з номерами кластерів (рис. 1.111).

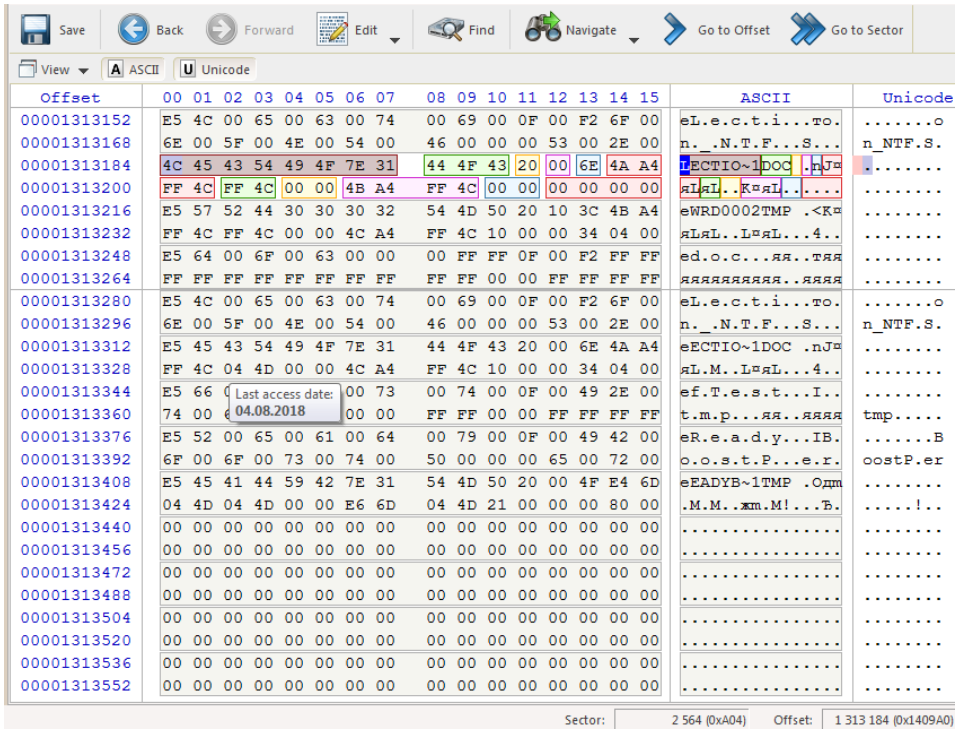


Рис. 1.109. Відновлення імені файлу

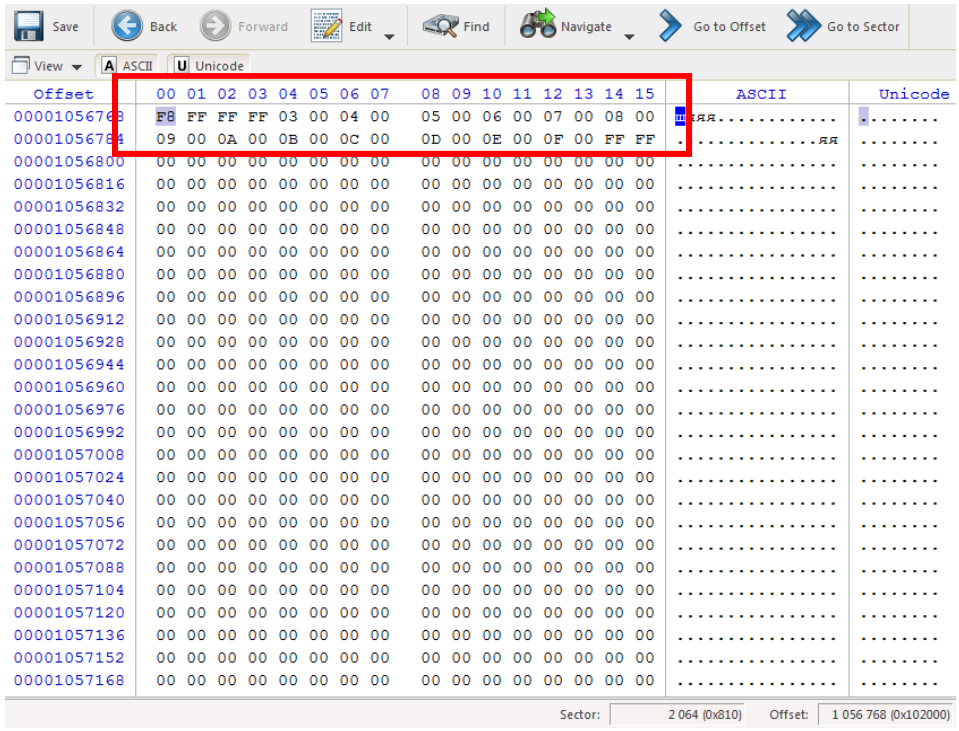


Рис. 1.110. Порожні кластери видаленого файлу в FAT1

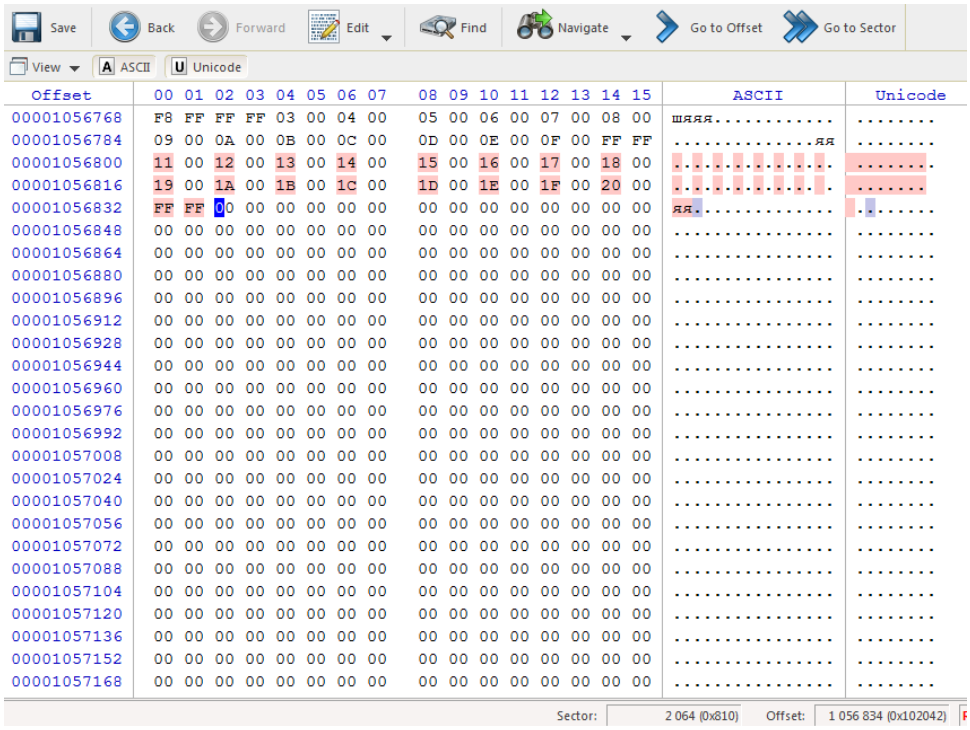


Рис. 1.111. Відновлена FAT1

Після цього переходять у FAT2 і відновлюють її також. Після збереження результатів файл і інформацію в ньому повністю відновлюються на накопичувачі. Із розглянутої методики відновлення слідують такі висновки:

– необхідно давати максимально унікальні імена файлам, щоб їх було простіше знаходити;

– періодично робити копії таблиці FAT1, особливо після запису/створення файлів або директорій;

– періодично робити дефрагментацію накопичувачів для того, щоб у тому числі таблиці FAT були заповнені без пробілів.

1.9.2. Завдання до самостійного виконання

1. Створити текстовий документ із розміром в декілька кластерів.

2. Визначити ланцюжок кластерів, що відповідають файлу у таблиці FAT. Привести відповідні розрахунки.

3. Визначити за кількістю кластерів у ланцюжку розмір файлу. Навести розрахунки і підтвердження їх результату.

4. Перевірити на відповідність змісту файла кожен кластер ланцюжка. Привести відповідні розрахунки і скріншоти.

4. Видалити файл і виконати його відновлення.

1.9.3. Запитання для самоперевірки

1. Чому на вкладці «Властивості файлу» завжди вказано декілька значень його розміру?

2. Які особливості запису номерів кластерів в таблиці FAT?

3. Як розрахувати розмір файлу за кількістю зайнятих байт в таблиці FAT?

4. Що міститься в кластерах файлу окрім записаної інформації?

5. Як зміниться ланцюжок кластерів файлу при видаленні з файлу інформації розміром в декілька кластерів?

6. Залежно від кількості вільного місця на накопичувачі назвіть можливі варіанти для місць запису кластерів в таблицю FAT після додавання інформації в файл.

7. Які дії необхідно виконати для відновлення файлу при його випадковому видаленні?

8. Які дії не можна виконувати після випадкового видалення файлу для можливого його відновлення?

9. Назвіть запобіжні заходи, які необхідно виконувати для надійного відновлення випадково видалених файлів та директорій.

Тема 1.10. Структура файлової системи NTFS

Мета: Ознайомитися зі структурою завантажувального сектора і навчитися обчислювати розмір запису MFT.

1.10.1. Теоретичні відомості

Файлова система NTFS (New Technology File System) була розроблена і запропонована для використання фірмою

Microsoft в 90-і роки минулого сторіччя. Проте, довго вона не знаходила широкого розповсюдження через вимогливість до системних ресурсів: потужності процесора, об'єму оперативної пам'яті і дискового простору. Вирішення проблеми системних ресурсів на початку 2000-х забезпечило широке розповсюдження NTFS, яка на сьогоднішній день залишається найбільш надійною і швидкою файловою системою.

Загальний підхід до використання простору на логічному диску полягає в тому, що на диску не має нічого, крім файлів. Концепція архітектури NTFS полягає в тому, що всі службові структури файлової системи організовані в виді файлів у загальному просторі логічного диска. На відміну від інших файлових систем в NTFS відсутні жорстко задані структури. Увесь дисковий простір виступає областю даних, і будь-який кластер може бути виділений файлу.

В основі логічного диска NTFS лежить Master File Table (MFT) – службовий файл, у якому містяться записи про всі файли і каталоги логічного диску. Під зону MFT спочатку виділяється 12,5% від всього дискового простору накопичувача. Виділений надмірний простір забезпечує де фрагментацію файлу MFT при збільшенні його розміру. Якщо простору не вистачатиме, то зона MFT буде зменшена до фактичного розміру файлу MFT.

Тільки завантажувальний сектор має фіксоване місцеположення, і знаходиться в нульовому секторі. Так само, як і в файлових системах сімейства FAT, він закінчується сигнатурою 55AA.

Найбільш важливою інформацією завантажувального сектора є наступна:

– номер початкового кластера MFT;

– номер початкового кластера файлу – копії перших системних записів MFT;

– розмір індексного блоку.

Структура завантажувального сектора наведенв в Додатку 2. Особливістю поля «Розмір запису MFT», що займає 1 байт, є представлення довжини запису в байтах або в кластерах.

Оскільки інформація в завантажувальному секторі зберігається в шістнадцятковій системі числення, то необхідно додатково визначити розмірність запису.

При знаходженні в відповідному байті додатнього числа, розмір запису MFT представлено в кластерах. Для отримання значення в байтах вираз обчислюють за формулою (1.2):

$$N_{байт} = N_{кл} \times N_{сек/кл} \times N_{б/сек}, \quad (1.2)$$

де $N_{байт}$ – кількість байт, що займає запис MFT;

$N_{кл}$ – кількість кластерів, що займає запис MFT;

$N_{сек/кл}$ – кількість секторів на кластер;

$N_{б/сек}$ – кількість байт в секторі, яка дорівнює 512 за замовчанням.

При знаходженні в відповідному байті від'ємного числа, розмір запису MFT представлено в байтах. Записане значення є ступенем, до якого підносять 2, щоб отримати довжину запису в байтах за формулою (1.3):

$$N_{байт} = 2^n, \quad (1.3)$$

де N – довжина запису MFT в байтах;

n – додатній еквівалент коду запису MFT.

За правилами шістнадцяткової аріфметики додатній еквівалент наведеного кода обчислюється за формулою (1.4):

$$n = 0 – L_{MFT}, \quad (1.4)$$

де n – додатній еквівалент коду запису MFT;

L_{MFT} – код розміру запису MFT.

Наприклад, поле «Розмір запису MFT» містить код 04, а один кластер складається з 32 секторів.

04 – це додатне число, отже, розмір запису вказаний в кластерах, і дорівнює чотирьом кластерам. Для розглядуваного прикладу за формулою (1) отримаємо розмір запису MFT в байтах:

$$N_{байт} = N_{кл} \times N_{сек/кл} \times N_{б/сек} = 4 \times 32 \times 512 = 65536 \text{ (байт)}$$

Розглянемо приклад, коли поле «Розмір запису MFT» містить код E5. Запис є від'ємним числом, оскільки перший біт одиничний. Отже, розмір буде розрахований в байтах.

За формулою (3) обчислюють додатній еквівалент коду запису MFT:

$$n = 0 - L_{MFT} = 00 - E5 = 1B_{16} = 27_{10}.$$

Розмір запису MFT обчислюють за формулою (1):

$$N_{байт} = 2^n = 2^{27} = 134217728 \text{ (байт)} \approx 134 \text{ (Мбайт)}.$$

Інформацію про логічний диск і файлову систему можна отримати не тільки із завантажувального сектора, але і за допомогою утиліт командного рядка. Починаючи з Windows XP, у командний рядок вбудована утиліта **Fsutil**, яка також визначає інформацію про логічний диск і файлову систему NTFS.

Утиліта містить набір власних команд, які можна переглянути, набравши в командному рядку **fsutil fsinfo** (рис. 1.112).

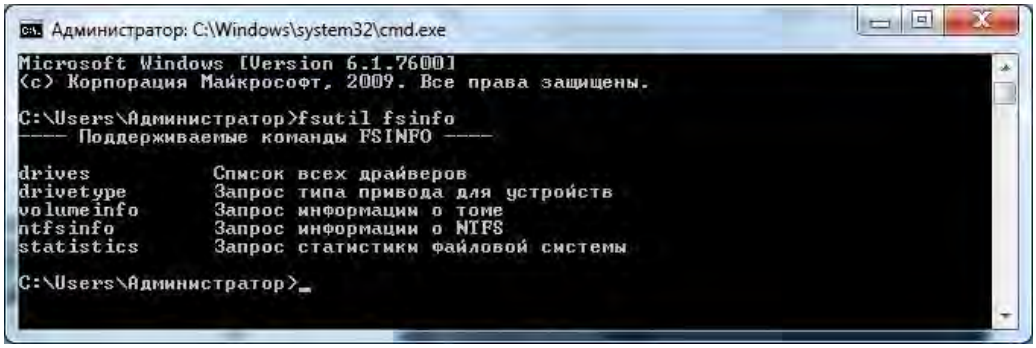


Рис. 1.112 Команди утиліти fsutil

Для того, щоб переглянути відомості про файлову систему, використовують команду **ntfsinfo** (рис. 1.113).

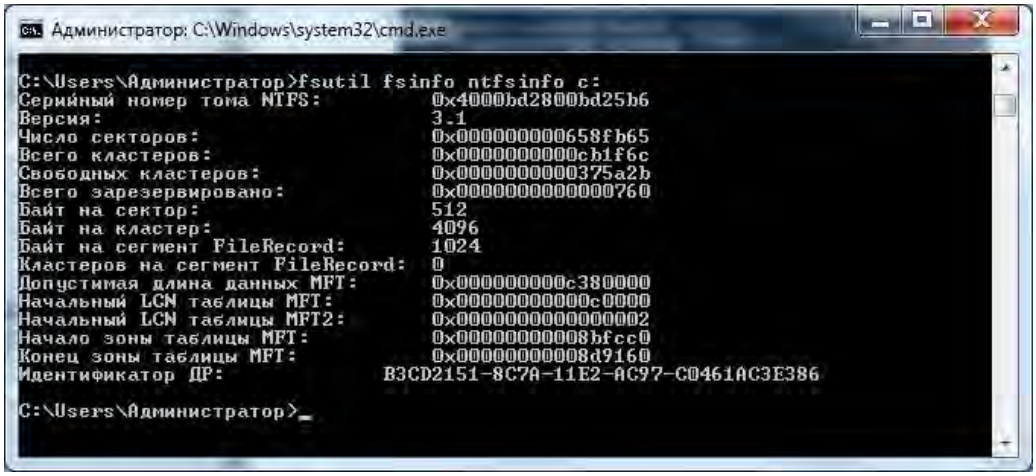


Рис. 1.113. Результат команди ntfsinfo

Для того, щоб отримати відомості про логічний диск використовують команду **volumeinfo** (рис. 1.114).

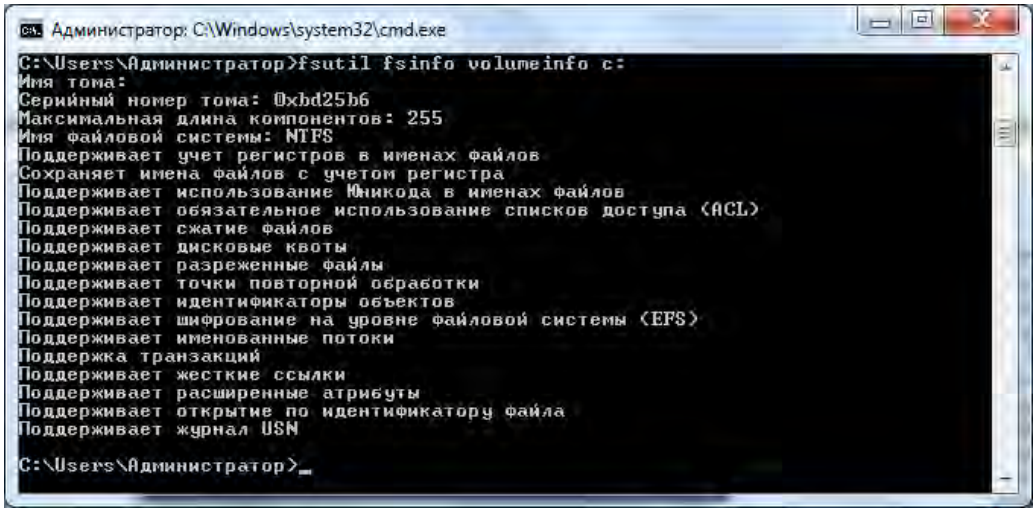


Рис. 1.114. Результат команди volumeinfo

Для отримання докладної статистики файлової системи на дисках використовують команду **statistics** (рис. 1.115).

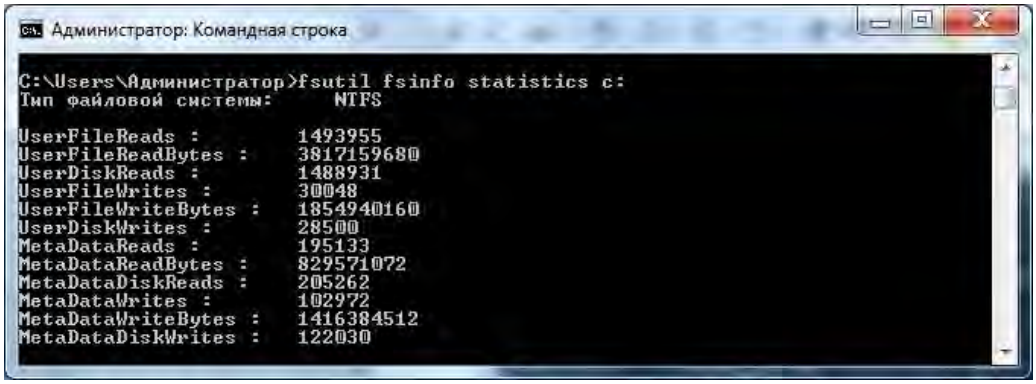


Рис. 1.115. Результат команди statistics

1.10.2. Завдання для самостіного виконання

1. Завантажити редактор Active Disk Editor і відкрити в ньому завантажувальний диск з файловою системою NTFS.

2. Створити таблицю параметрів завантажувального сектора, в якій привести їх значення в десятковій формі. Таблицю створити за зразком:

Зсув	Розмір поля	Назва параметра	Шістнадцятковий код параметра	Десятковий код параметра

3. Визначити розмір запису MFT, якщо відповідний байт завантажувального сектора містить:

Для парних варіантів	Для непарних варіантів
03	01
02	05
F5	F7
E7	EE

4. В командному рядку перевірити правильність визначених параметрів завантажувального сектора за допомогою утиліти Fsutil і її команд.

1.10.3. Запитання для самоперевірки

1. З якої причини файлова система NTFS була впроваджена набагато пізніше ніж розроблена?
2. Яка структура лежить в основі логічного диска NTFS? Який об’єм спочатку виділяється під неї?
3. Назвіть особливості визначення розміру запису MFT.
4. Що таке додатній еквівалент коду запису MFT?
5. Яка утиліта командного рядка використовується для перевірки параметрів завантажувального сектора? Назвіть її основні команди.

Тема 1.11. Визначення стандартних атрибутів і місцезнаходження об’єктів у файловій системі NTFS

Мета: Навчитися визначати стандартні атрибути об’єктів у файловій системі NTFS і їх місцезнаходження.

1.11.1. Теоретичні відомості

Згідно з концепцією файлів в NTFS, файл складається з атрибутів. **Атрибутом** називають самостійний інформаційний потік (послідовність байтів), збережений на логічному диску. Атрибут має своє найменування, розмір і місце розташування.

Файл завжди складається з декількох атрибутів – декількох окремих інформаційних потоків, які в сукупності дають повну характеристику даного файлу. Строго кажучи, NTFS зчитує або записує не файл, а окремі потоки атрибутів файлу.

Атрибути файлів і каталогів

Типовими атрибутами для звичайних файлів є $STANDART INFORMATION, $ FILE NAME і $DATA.

Атрибут $STANDART INFORMATION містить стандартні атрибути «тільки для читання», «прихований», «архівний» та інші, час створення і модифікації файлу, число каталогів, що посилаються на цей файл.

Атрибут $FILE NAME містить ім'я файлу в кодуванні Unicode завдовжки до 255 символів. Файл може мати кілька атрибутів імені. Наприклад, другий атрибут імені може містити коротку назву у форматі 8.3 для сумісності з іменами в MSDOS.

Атрибут $DATA є власне вмістом файлу. Файл може мати кілька атрибутів даних. Додаткові атрибути даних повинні мати власні імена.

Усі атрибути діляться на **резидентні** і **нерезидентні.**

Резидентним називається атрибут, значення якого вміщується в один запис MFT. Резидентними можуть бути атрибути невеликого інформаційного розміру, оскільки **стандартний розмір запису MFT становить 1, 2 або 4 Кбайт**.

Атрибути $STANDART INFORMATION і $FILE NAME завжди резидентні. Якщо файл має невеликий обсяг даних, то атрибут $DATA може бути **резидентним.**

Резидентний атрибут складається з двох полів: «Заголовок атрибута» і «Значення атрибута». Заголовок атрибута містить ознаку резидентного атрибута, зміщення від початку заголовка до поля значення атрибута і довжину цього значення в байтах.

Наприклад, формат резидентного атрибута «Ім’я файлу» для файлу license.txt має наступний вид:

«Ім’я файлу» (резидентний атрибут)

Заголовок	Значення атрибуту
«resident»	license.txt
Зміщення: 8h	
Довжина: 16h	

Довжина атрибуту «Ім’я файлу» дорівнює кількості символів імені в двобайтному коді Unicode.

У файлі license.txt міститься текст: «Доброго дня!», тому формат резидентного атрибута «Дані» має наступний вид:

«Дані» (резидентний атрибут)

Заголовок	Значення атрибуту
«resident»	Доброго дня!
Зміщення: 8h	
Довжина: Ch	

Значення атрибуту записано в коді ASCII, 1 символ кодується 1 байтом, отже, довжина атрибута складає 12 байт.

Нерезидентними називають атрибути, значення яких зберігається не в MFT, а окремо на логічному диску. Для таких атрибутів запис MFT містить інформацію про їх розміщення.

У більшості випадків дані файлу занадто великі, щоб поміститися в одному записі MFT. Для них будуть виділятися окремі кластери за межами MFT. Тобто, атрибут «Дані файлу» буде нерезидентним. Заголовок нерезидентного атрибута (non-resident) містить інформацію, необхідну для пошуку кластерів, де розміщуються дані. Заголовок представлений у виді таблиці з переліком займаних екстентів.

Екстент – це безперервна послідовність кластерів, займаних об'єктом. Поняття «екстент» використовується з метою зменшення кількості записів розміщення.

Розглянемо приклад для файлу, фрагменти якого розміщені в наступних кластерах: з 1008 по 1018, з 1824 по 1830, 2000, з 2001 по 2004 кластери. Інформація про розміщення в заголовку нерезидентного атрибута «Дані» буде виглядати так:

Початковий кластер екстента	Кількість кластерів
1008	11
1824	7
2000	4

Таким чином, вміст файлу займає 22 кластери. При відомій кількості кластерів/сектор можна порахувати це значення в байтах.

Перегляд атрибутів і секторів розміщення файлів будемо здійснювати за допомогою утиліти nfi, не прив'язуючись до конкретного шістнадцяткового редактора, оскільки операційна система блокує запити редакторів, використаних в попередніх лабораторних роботах курсу.

Утиліта nfi з пакета засобів підтримки Windows Support Tools дозволяє отримати текстовий дамп всього вмісту головної файлової таблиці. Завантажити перевірену робочу утиліту nfi можна за посиланням: https://ab57.ru/syssuite.html. Після завантаження архіву, його необхідно розпакувати в директорію system32 або system64 в залежності від розрядності операційної системи.

Зміст заголовку і значення атрибуту «Дані» мають вид:

«Дані» (нерезидентний атрибут)

Заголовок		Значення атрибуту
«non-resident»		
Зміщення: Ch		
Початковий кластер екстента	Кількість кластерів	
1008	11	
1824	7	
2000	4	

Nfi – консольна утиліта, запуск якої здійснюється через командний рядок із правами адміністратора. У результаті виконання команди **nfi drive: |more** здійснюється посторінковий вивід всіх записів MFT логічного диска **drive :**. Для кожного запису MFT утиліта показує ім'я об'єкта, його атрибути та інформацію про займані сектори на логічному диску (рис. 1.116).

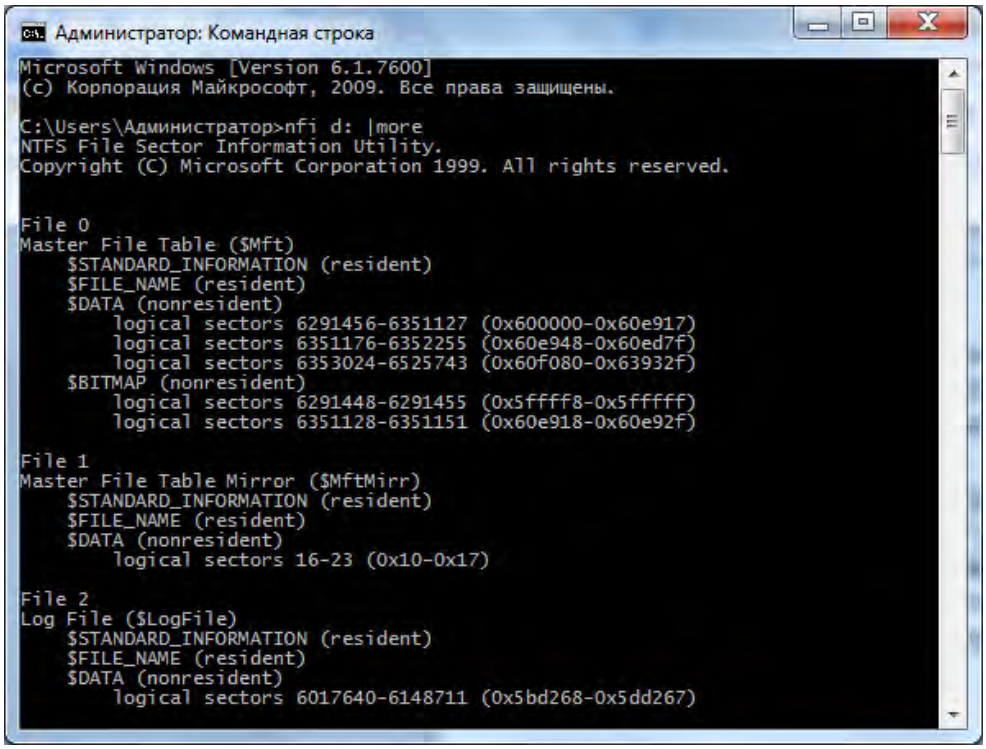


Рис. 1.116. Результат виконання команди nfi d: |more

Для переходу до наступного рядку запису натискають Enter, а для виводу наступної сторінки із даними – пробіл. Для збереження цієї інформації результат виконання записують в текстовий файл (створювати його заздалегідь не треба) за допомогою команди: **nfi drive: > drive \report.txt.**

Утиліта nfi також дозволяє отримати відомості про розміщення конкретного файлу або каталогу: **nfi drive:\шлях**

до об'єкта (рис. 1.117) або навпаки дізнатися який об’єкт розміщений в заданому секторі (рис. 1.118).

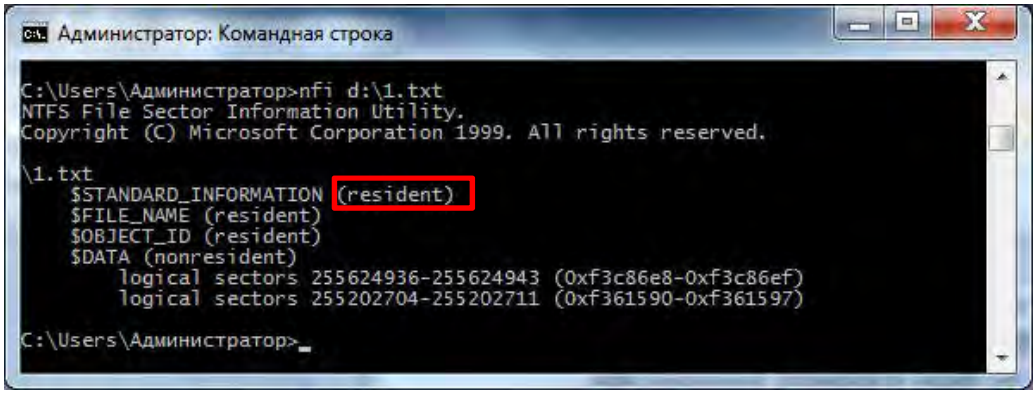


Рис. 1.117. Відомості про файл 1.txt на диску d

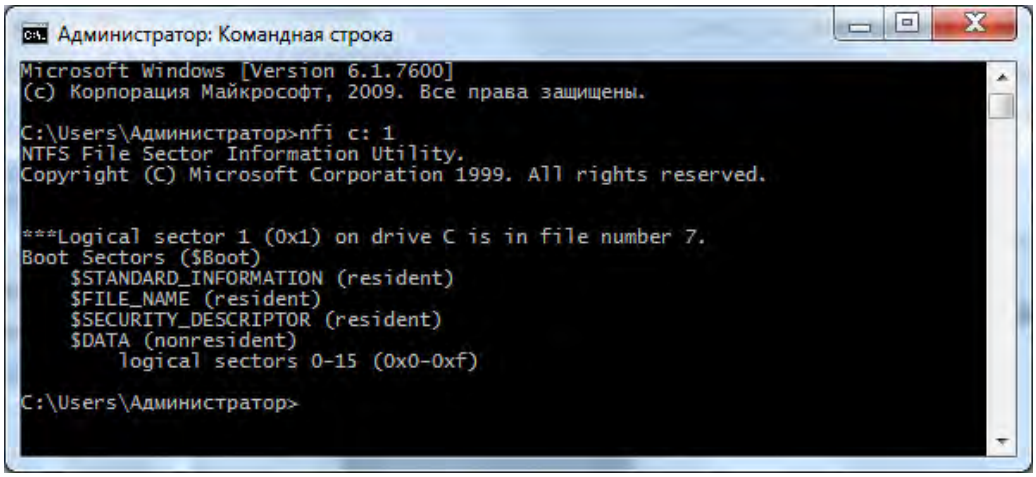


Рис. 1.118. Відомості про файл, що займає перший сектор на диску c:

Для того, щоб дізнатися зміст першого сектора файлу 1.txt у шістнадцятковому редакторі відкривають носій як логічний диск (рис. 1.119) і переходять в сектор із відповідним номером (рис. 1.120).

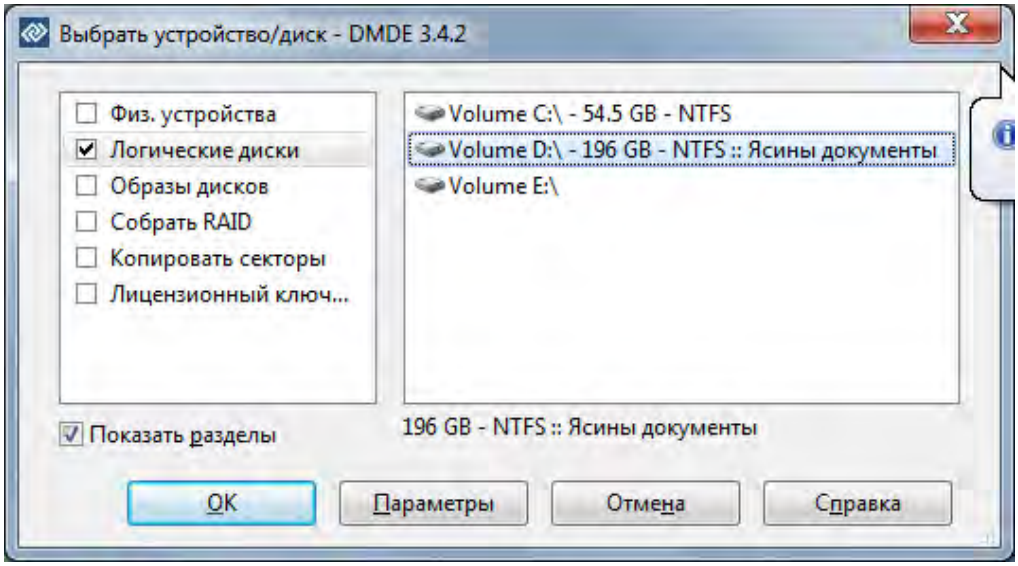


Рис. 1.119. Відкриття носія як логічного диска

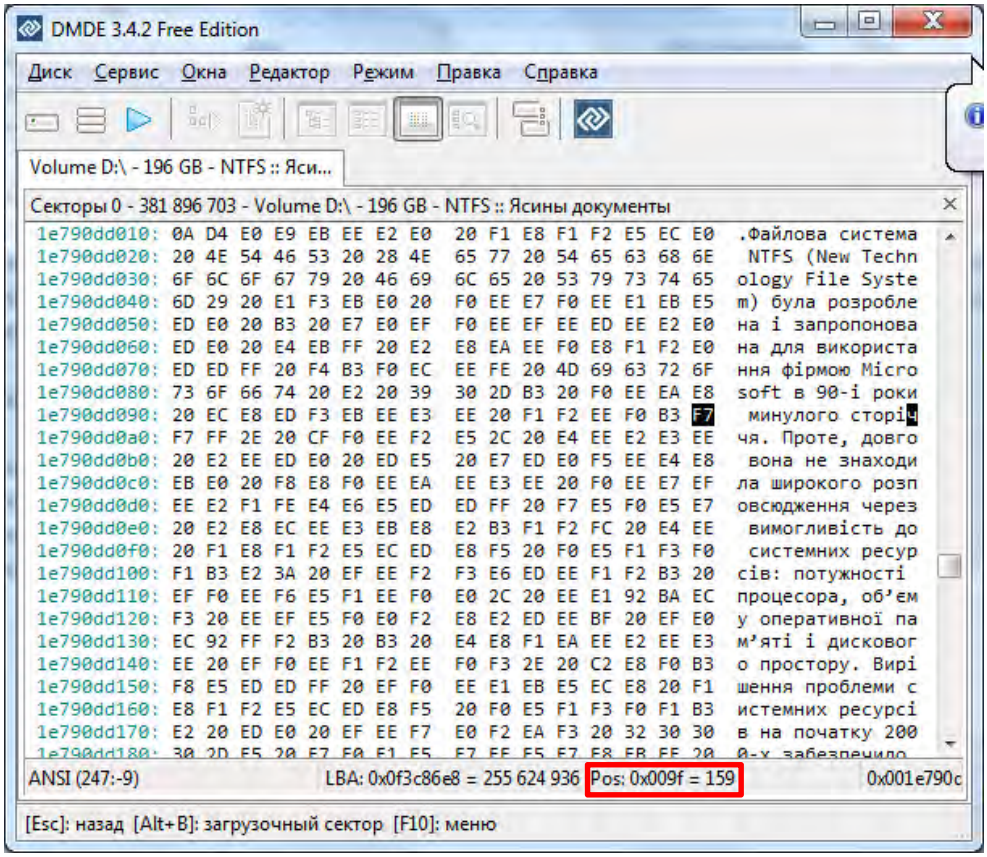


Рис. 1.120. Зміст першого кластера файлу 1.txt

1.11.2. Завдання для самостійного виконання

1. Визначити кількість екстентів, які займають наступні файли:

Для парних варіантів	Для непарних варіантів
Файл Lab4.doc фрагментований і займає наступні кластери: 1, 8, 9, 10, 22, 28, 32, 33, 34, 36, 38.	Файл Lab4.doc фрагментований і займає наступні кластери: 10, 18,19, 20, 24, 25, 30, 32, 35, 36, 37.
Файл Lab5.doc фрагментований і займає наступні кластери: 3, 18, 20, 21, 22, 23, 24, 30, 31, 32, 44,45.	Файл Lab5.doc фрагментований і займає наступні кластери: 21, 22, 23, 40, 41, 42, 45, 46, 50, 52, 53.
Файл Lab6.doc фрагментований і займає наступні кластери: 55, 56,80, 82, 83, 84, 85, 90, 93, 94.	Файл Lab6.doc фрагментований і займає наступні кластери: 60, 61, 62, 63, 65, 66, 70, 71, 75, 76.
Файл Lab7.doc фрагментований і займає наступні кластери: 91, 95,96, 97, 98, 101, 102, 120, 121, 125.	Файл Lab7.doc фрагментований і займає наступні кластери: 104, 108,109, 110, 111, 115, 116, 120, 121.
Файл Lab8.doc фрагментований і займає наступні кластери: 112, 122, 124, 125, 126, 127, 129, 130, 131, 133.	Файл Lab8.doc фрагментований і займає наступні кластери: 114, 116,117, 118, 119, 121, 124, 128, 129.

Підрахувати загальну кількість екстентів для кожного файлу і розмір файлу в байтах, якщо кількість секторів/кластер = 8.

2. Розписати для перших трьох файлів атрибути «Ім’я файлу» і «Дані» в стандартному виді.

3. Створити на накопичувачі з файловою системою NTFS по 2 txt-файли з резидентним і нерезидентним атрибутом «Дані». За допомогою утиліти nfi знайти і навести екстент(и) їх розміщення.

4. Для створених файлів представити в стандартному виді їх атрибути «Ім’я файлу» і «Дані».

5. Для файлів з нерезидентним атрибутом «Дані» в шістнадцятковому редакторі перейти в перший сектор і навести його зміст.

6. За допомогою утиліти nfi знайти і навести зміст 3 конкретних секторів накопичувача (номера секторів обрати самостійно).

1.11.3. Запитання для самоперевірки

1. Які атрибути відносяться до стандартних атрибутів файлу?

2. Які атрибути називаються резидентними, а які – нерезидентними?

3. Що називається екстентом? Як визначається розмір файлу за кількістю зайнятих екстентів?

4. Яка особливість запису значень резидентного і нерезидентного атрибуту «Дані»? У якому випадку значення атрибуту «Дані» може бути резидентним?

5. Яка особливість запису значень атрибуту «Ім’я файлу»?

6. Чи буде зміщення для значення атрибуту «Ім’я файлу» завжди постійним? Відповідь поясніть.

7. З чим пов’язане використання логічної адресації для пошуку змісту файлів в NTFS?

РОЗДІЛ 2. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ОС WINDOWS

Тема 2.1. Перевірка реєстру ОС Windows на наявність вірусів

Мета: Ознайомлення зі структурою реєстру і роботою утиліти «Редактор реєстру». Отримати навики виправлення помилки в файлах реєстру і перевірити їх на наявність вірусів типу Trojan.

2.1.1. Теоретичні відомості. Будова реєстру Windows

Реєстр операційної системи Windows – це база даних, де зберігається інформація про налаштування системи. Цією інформацією користується як сама операційна система, так і інші програми. У деяких випадках відновити працездатність системи після збою можна, завантаживши працездатну версію реєстру. Для цього необхідно мати копію реєстру. Основним засобом для перегляду і редагування записів реєстру служить спеціальна утиліта «Редактор реєстру».

Файл редактора реєстру знаходиться в папці Windows і називається regedit.exe. Запуск утиліти: **Пуск -> Виконати -> regedit**. Реєстр організований в ієрархічну структуру розділів, підрозділів і параметрів. Після запуску з'явиться вікно редактора реєстру. Зліва у вікні розташований список з 5 кореневих розділів реєстру (рис. 2.1).

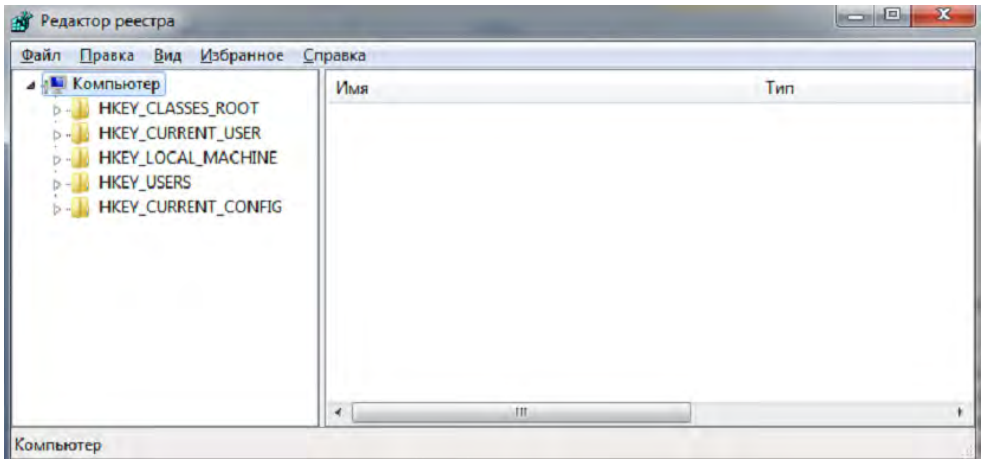


Рис. 2.1. Вікно утиліти «Редактор реєстру»

Короткий опис кожного з розділів наведено в табл. 2.1.

Робота з розділами реєстру аналогічна роботі з директоріями в Провіднику Windows. У реєстрі Windows системна інформація розбита на так звані кущі (hive). Термін «кущ» описує деревоподібну структуру розділів, підрозділів і параметрів, що виходить з вершини ієрархії реєстру. Кущ міститься в окремому файлі і має окремий журнал, які знаходяться в папках Windows\System32\Config або C:\Documents and Settings\ім'я.

Кожен кущ в реєстрі Windows пов'язаний з набором стандартних файлів. Імена стандартних кущів і файлів показані в табл. 2.

Кожен розділ або підрозділ реєстру може містити дані – параметрами або ключі. Деякі параметри зберігають відомості для конкретних користувачів, інші зберігають відомості, які застосовуються до всіх користувачів комп'ютера. Параметр реєстру має ім'я, тип даних і значення. Табл. 3 містить список типів даних, визначених і використовуваних системою.

Таблиця 2.1.

Кореневі розділи реєстру

Ім’я кореневого розділу	Опис
HKEY_LOCAL_MACHINE	Відомості про локальний комп'ютер, включаючи дані про обладнання і операційну систему, такі як тип шини, системна пам'ять, драйвери пристроїв і параметри завантаження.
HKEY_USERS	Відомості о завантажених профілях користувачів та профіль, використовуваний за замовчуванням. Сюди включені відомості, також з'являються в піддереві HKEY_CURRENT_USER. Віддалені користувачі не мають профілів в цьому розділі сервера; їх профілі знаходяться в реєстрах

	власних комп'ютерів.

Продовження табл. 2.1.

HKEY_CLASSES_ROOT	Відомості, які використовуються різними технологіями OLE, і дані про зіставлення типів файлів. Певний розділ або параметр існують в HKEY_CLASSES_ROOT, якщо відповідний розділ або параметр існують в HKEY_LOCAL_MACHINE \ SOFTWARE \ Classes або HKEY_CURRENT_USER \ SOFTWARE \ Classes. Якщо розділ або параметр є в обох місцях, в HKEY_CLASSES_ROOT з'явиться значення з HKEY_CURRENT_USER.
HKEY_CURRENT_USER	Профіль користувача, який увійшов в систему локально (на відміну від віддаленого користувача), включаючи змінні середовища, параметри робочого столу, здійснювати підключення до мережі, принтерів і додатків. Це піддерево є псевдонімом піддерева HKEY_USERS і вказує на HKEY_USERS\обліковий_код_ поточного_користувача.

Продовження табл. 2.1.

HKEY_CURRENT_CONFIG	Відомості про конфігурацію обладнання, що використовується локальним комп'ютером при запуску системи. ці відомості використовуються для настройки завантажуються драйверів і дозволу дисплея. Це піддерево є частиною поддерева HKEY_LOCAL_MACHINE і відповідає HKEY_LOCAL_MACHINE \ SYSTEM \ Current ControlSet \ Hardware Profiles \ Current.

Таблиця 2.2.

Зв'язок стандартних кущів з файлами

Кущ реєстру	Імена файлів
HKEY_LOCAL_MACHINE\SAM	Sam и Sam.log
HKEY_LOCAL_MACHINE\SECURITY	Security и Security.log
HKEY_LOCAL_MACHINE\SOFTWARE	Software и Software.log
HKEY_LOCAL_MACHINE\SYSTEM	System и System.log
HKEY_CURRENT_CONFIG	System и System.log
HKEY_CURRENT_USER	Ntuser.dat и Ntuser.dat.log
HKEY_USERS\.DEFAULT	Default и Default.log

Таблиця 2.3.

Визначені системою типи даних

Тип даних	Опис
REG_BINARY	Необроблені двійкові дані. Більшість відомостей про устаткування зберігається в вигляді двійкових даних і виводиться в редакторі реєстру в шістнадцятковому форматі.
REG_DWORD	Дані, представлені цілим числом (4 байта). Багато параметри служб і драйверів пристроїв мають цей тип і відображаються в двійковому, шістнадцятковому або десятковому форматах.
REG_EXPAND_SZ	Рядок даних змінної довжини. Цей тип даних включає змінні, які обчислюються при використанні програма або служба.
REG_MULTI_SZ	Багаторядковий текст. Цей тип, як правило, мають списки і інші записи в форматі, зручному для читання. Окремі значення розділяються пробілами, комами або іншими символами.
REG_SZ	Текстовий рядок фіксованої довжини.
REG_FULL_RESOU RCE_DESCRIPTOR	Послідовність вкладених масивів, розроблена для зберігання списку ресурсів апаратного компонента або драйвера.

Список імен кущів і шляхів до каталогів, в яких вони зберігаються, розташовані в розділі HKEY_LOCAL_MACHINE \ SYSTEM \ Current Control Set \ Control \ hivelist (рис. 2.2).

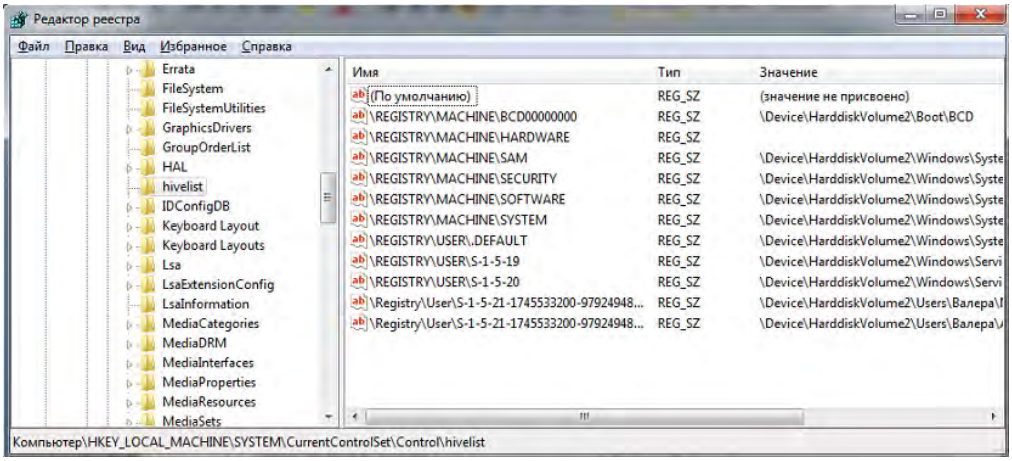


Рис. 2.2. Розділ hivelist з іменами кущів і шляхів до їх каталогів

Реєстр містить важливі дані про комп'ютер, його додатки і файли, але при видаленні програм в реєстрі залишаються їх сліди, тому необхідно проводити повну перевірку системи, видаляти невірні записи реєстру і виправляти помилки. Для цього, наприклад, можна використати програму CCleaner. CCleaner розповсюджується безкоштовно з офіційного сайту www.ccleaner.com. Для роботи із програмою переходимо в розділ «Реєстр» і виконуємо «Пошук» (рис. 2.3).

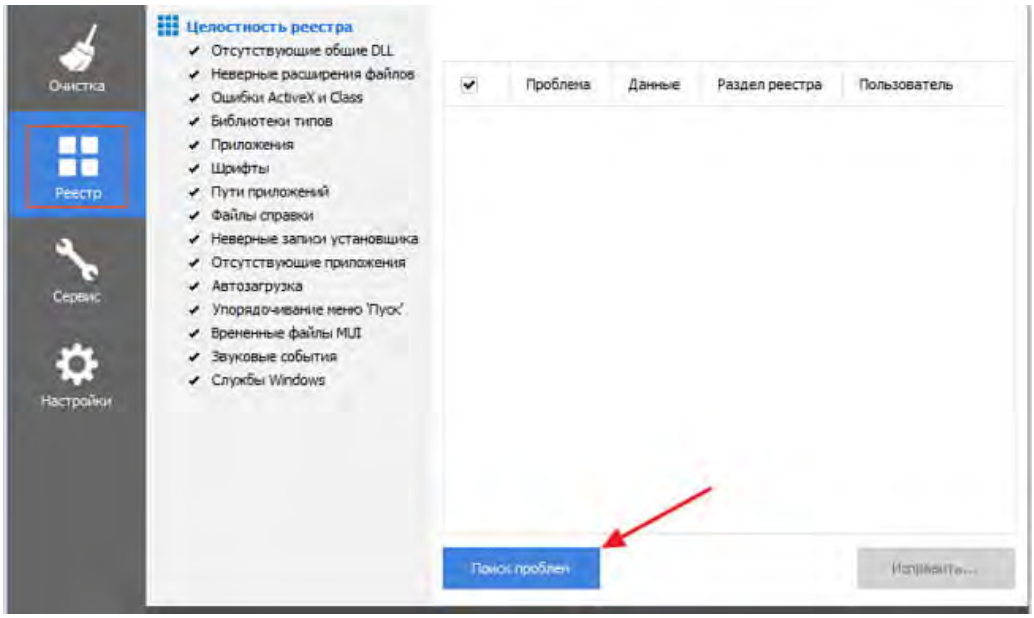


Рис. 2.3. Пошук проблем в реєстрі в CCleaner

Після того, як відобразиться перелік помилок, натискаємо «Виправити» (рис. 2.4).

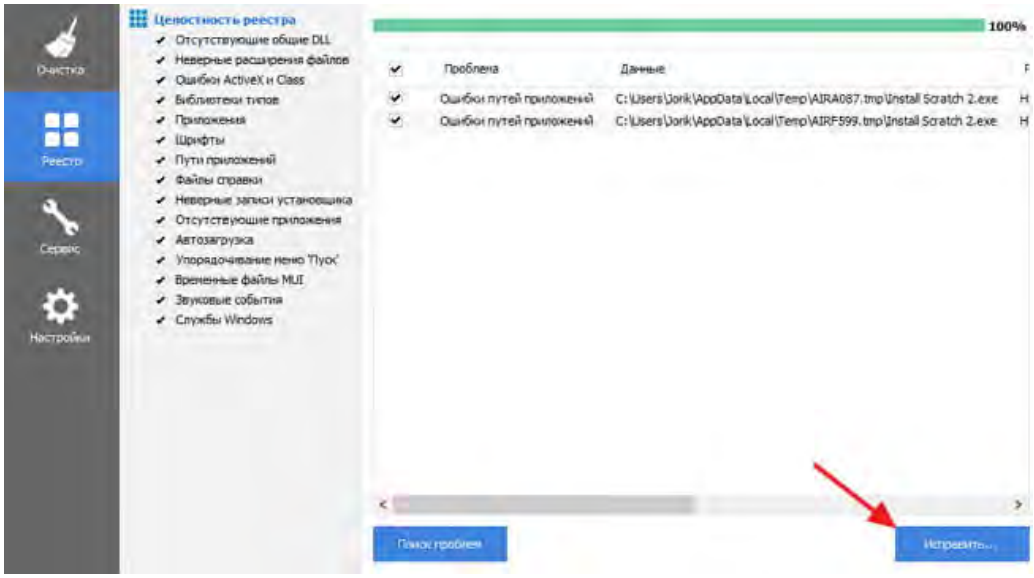


Рис. 2.4. Виправлення помилок реєстру в CCleaner

На цьому кроці програма запропонує попередньо створити бекап. Рекомендовано погодитися і зберегти початковий варіант, щоб у разі падіння ОС, була можливість її відновити (рис. 2.5).

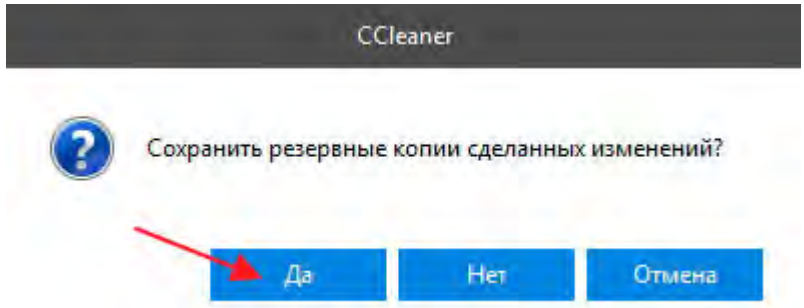


Рис. 2.5. Створення бекапу в CCleaner

Далі натискаємо на «Виправити відмічені» (рис. 2.6). Працює як на Windows 10, так і на Xp. Після виконання

нескладних дій, реєстр буде очищений від загроз. Спосіб ефективний також і при видаленні вірусів в браузері.

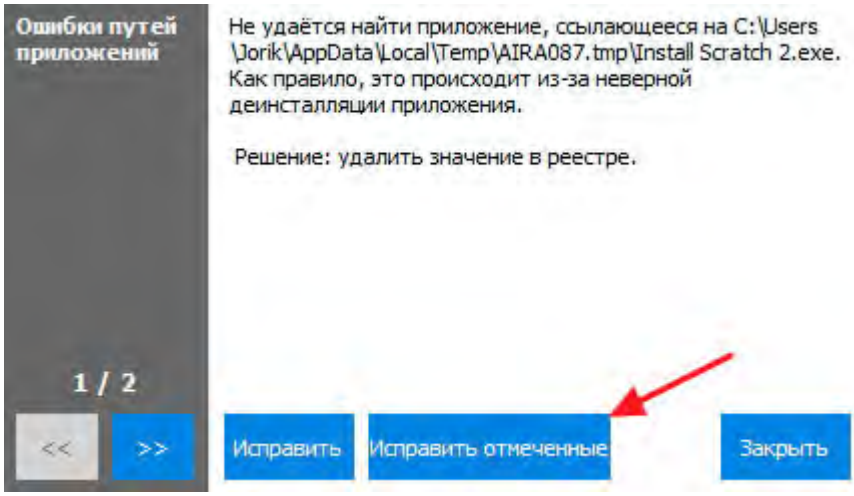


Рис. 2.6. Видалення записів з реєстру в CCleaner

Зловмисник може скористатися реєстром для нанесення серйозного збитку комп'ютеру. Дуже важливо підтримувати високий рівень безпеки реєстру і перевіряти його на наявність вірусів.

Один з найбільш поширених типів вірусів – троянські програми. «Троянський кінь» – це програма, отримана шляхом явної зміни або додавання команд в призначену для користувача програму. «Троянська програма» може заважати роботі користувача, шпигувати за користувачем, використовувати ресурси комп'ютера для незаконної діяльності і т.д.

Потенційними місцями записів «троянських програм» в системному реєстрі є:

– розділи, що описують програми;

– розділи, що запускаються автоматично при завантаженні операційної системи від імені користувача та системи.

Для перевірки на наявність троянської програми необхідно:

1. Перевірити вміст параметра HKEY_LOCAL_MACHINE \ SOFTWARE \ Microsoft \ Windows NT \ CurrentVersion \

Winlogon. За замовчуванням цей параметр має значення C: \ Windows \ system32 \ userinit.exe. Якщо в значенні містяться додаткові записи, то це можуть бути «троянські програми». В цьому випадку проаналізуйте місце розташування програми, зверніть увагу на час створення файлу.

2. Перевірити розділ автозапуску Run: HKEY_LOCAL_MACHINE \ SOFTWARE \ Microsoft \ Windows \ CurrentVersion \ Run.

При перегляді розділу аналізують які програми автоматично запускаються при завантаженні Windows і виділяють записи, що викликають підозри.

Побачити віруси можна також за допомогою диспетчера задач. Запустити диспетчер можна комбінацією клавіш "Ctrl + Shift + Esc" або викликати за допомогою стандартного "Ctrl + Alt + Del".

На які підозрілі процеси потрібно звертати увагу:

1. Процеси, імена яких схожі з іменами головних системних процесів (з різницею в одну-дві букви).

Список головних системних процесів складається з:

– csrss.exe;

– explorer.exe;

–lsass.exe;

– svchost.exe;

– system;

– wininit.exe;

– winlogon.exe.

Наприклад: CSRSS.EXE – оригінальний процес, а CSRCS.EXE – підозріла підробка. SVCHOST.EXE – системна програма, а SCVHOST.EXE – ймовірний вірус (рис. 2.7).

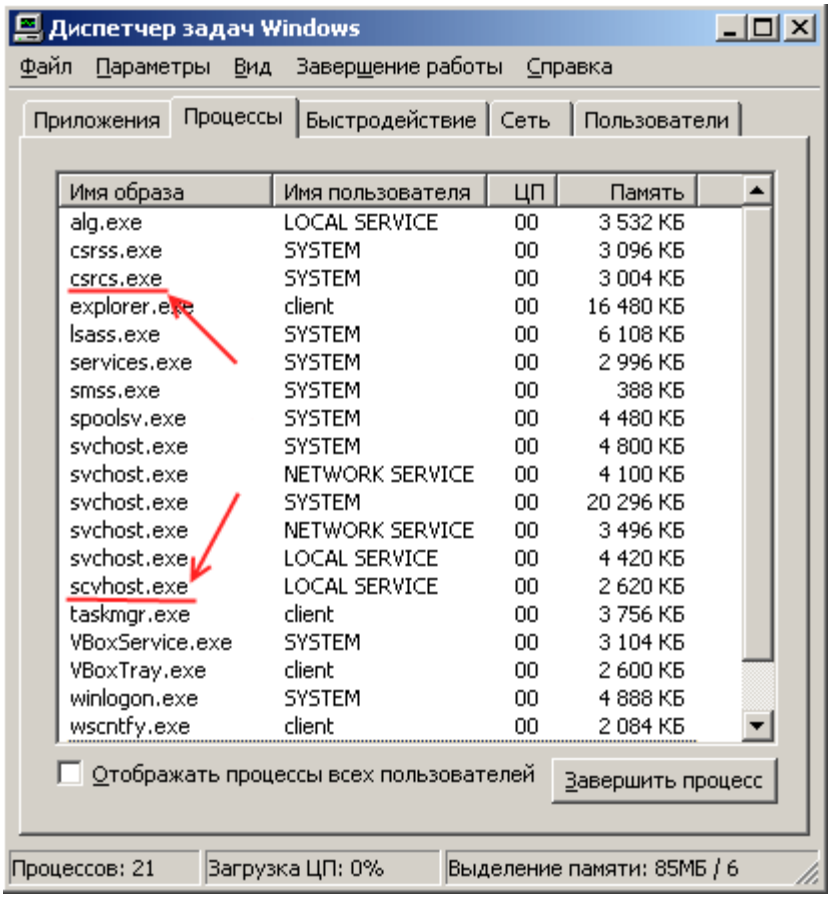


Рис. 2.7. Виявлення схожих на оригінальні процеси в диспетчері задач Windows

2. Процеси з підозрілими іменами (рис. 2.8).

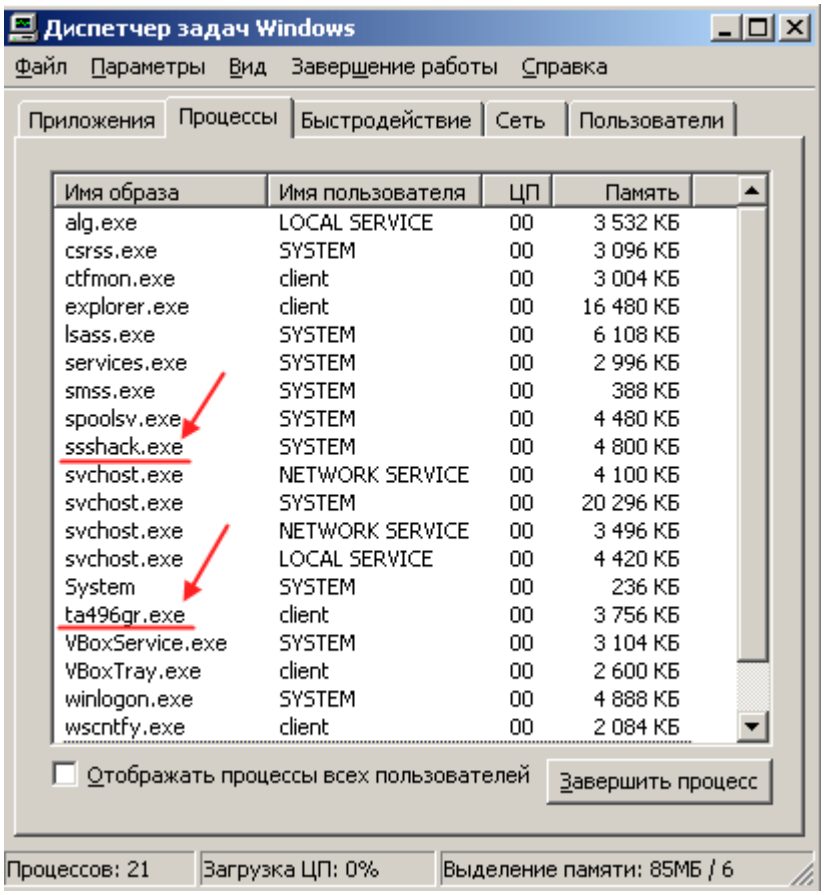


Рис. 2.8. Виявлення підозрілих процесів в диспетчері задач Windows

3. Процеси з підозрілим описом або без опису зовсім.

Всі знайдені підозрілі назви файлів потрібно буде перевірити в базі даних відомих процесів. Такі бази можна знайти за наступними адресами:

http://wiki.compowiki.info/ProcessyWindows (російськомовний);

http://www.what-process.com (англомовний);

http://www.tasklist.org (англомовний).

Також можна скористатися звичайною пошуковою системою і ввести ім'я підозрілого процесу. Якщо Ви запідозрили невідомі процеси, але вони позначені в базах як нешкідливі, то можна їх залишити.

Зверніть увагу на процеси, які не знайшлися на вищеописаних сайтах. По-перше, визначте поточну директорію запуску підозрілої програми. Для цього клікніть по процесу правою кнопкою миші і перейдіть на вкладку «Властивості». Якщо файл зберігається у вашій папці Windows \ System32, Ви можете бути впевнені, що не маєте справу з вірусом.

Найчастіше місце розташування вірусів – директорія C:\Windows\Temp. Отже, очистка всіх файлів в директорії Temp комп'ютера зменшить ризик його зараження. Для цього викликають команду «Виконати» (Win + R) і вводять в рядку «temp» (рис. 2.9). Це відкриє папку Temp на диску з встановленою операційною системою (рис. 2.10).

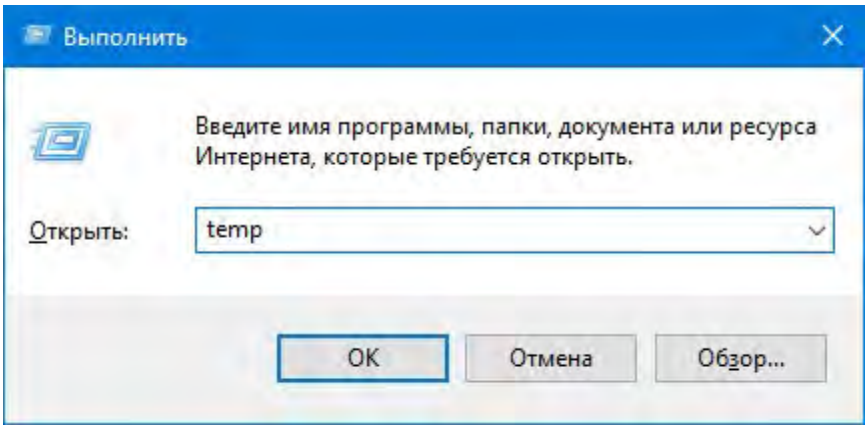


Рис. 2.9. Виклик відкриття директорії Temp

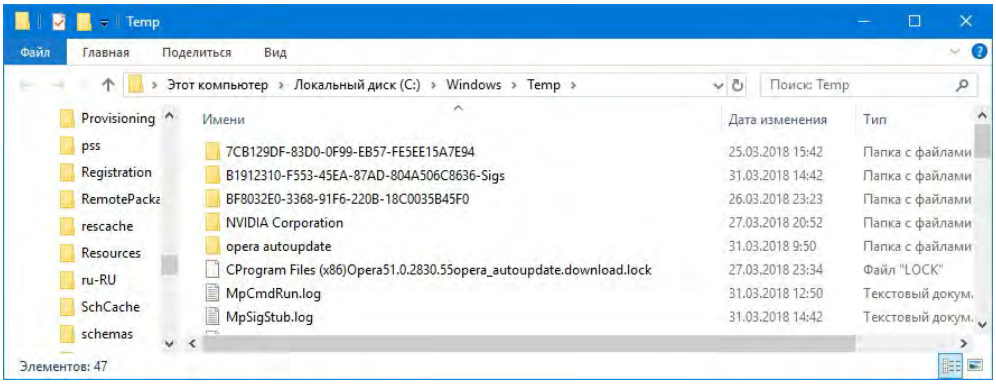


Рис. 2.10. Тимчасові файли директорії Temp

За останні роки найвідоміші віруси Petya і Petya.A пошкодили сотні тисяч комп'ютерів по всьому світу, вимагаючи від власників даних гроші за їх роз шифровку. Petya відноситься до завантажувальних вірусів і замість оригінального сектора MBR записує його змінену копію.

Модифікація Petya.A радикально відрізняється від версії Petya, для якої програмісти оперативно створили алгоритм розшифровування файлів і розблокування операційної системи. У новій версії використаний криптографічно стійкий шифр зі створенням особливого ключа для кожного диска і його подальшим видаленням. Це унеможливило створення алгоритмів розшифровування файлів. Якщо в пошуку по файлах ви виявили файли «C: \ Windows\perfc.dat» або «C:\Windows\dllhost.dat» – це і є свідчення про зараження вірусом. Однак творці вірусу також читають публікації в Інтернет і можуть перейменувати ці файли.

При зараженні менш уразливим вірусом, що не блокує доступ до ПК, доцільно перевіряти сектор MBR на наявність стандартних сигнатур і цілісність Partition Table.

Функція запобігання зараженню та блокуванню операційної системи реалізована правами доступу. За замовчуванням адміністратору надається повний доступ до всього реєстру, в той час як інші користувачі в основному мають повний доступ до розділів, що належать їх обліковим записам (у тому числі HKEY_CURRENT_USER) і читання розділів, що належать комп'ютеру і його програмному забезпеченню. Користувачі не мають доступ до розділів, що належать обліковим записам інших користувачів. Користувачі, що мають відповідні дозволи доступу до розділу, можуть змінювати дозвіл на доступ до цього розділу і будь-яким розділам, що містяться в ньому. Отже, виходити в Інтернет з облікового запису адміністратора не рекомендовано, оскільки це додатковий ризик повного зараження ПК.

Коректна робота багатьох програм, що працюють під управлінням ОС, забезпечується завдяки .NET Framework. Для того, щоб забезпечити працездатність встановлених додатків,

необхідно встановити декілька версій фреймворка, оскільки більш нова не заміщує стару і включає окремий набір функцій. Для визначення версії .NET Framework на комп'ютері є два способи – приблизний і точний. Приблизну версію .NET Framework можна дізнатися через реєстр за шляхом: **HKLM\SOFTWARE\Microsoft\.NET Framework**. Шлях також може закінчуватися на **.NET Framework Setup**. Далі необхідно переглянути список ключів, показаних на рис. 2.13.

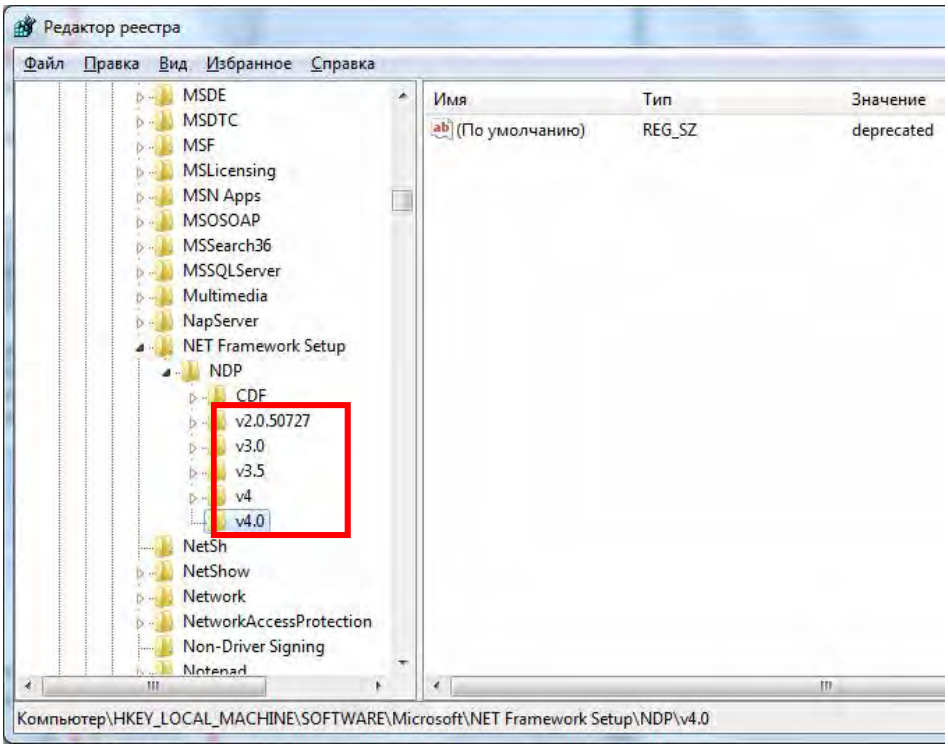


Рис. 2.13. Базові версії .NET Framework

Серед ключів наведені тільки базові версії, які можна уточнити, перейшовши до системної директорії Microsoft.NET за шляхом: %SystemRoot%\Microsoft.NET\Framework.

Назва директорій із встановленими версіями фреймворків має загальний вид vX.X.XXX. У назві вказані загальні версії так само, як і в реєстрі. Для уточнення версії необхідно зайти в директорію з іменем, що відповідає останній загальній версії

фреймворка (рис. 2.14) і знайти файл **mscorlib.dll,** у властивостях якого буде вказана точна версія (рис. 2.15).

Версія .NET Framework 4 і її підверсії додають близько 3000 нових ключів у реєстр та 6000 нових значень даних, що приблизно збільшує його на 1%. Це досить велика кількість, проте вона не впливає на продуктивність системи, оскільки, наприклад при встановленні MS Office кількість нових ключів і значень параметрів становить в кілька разів більше.

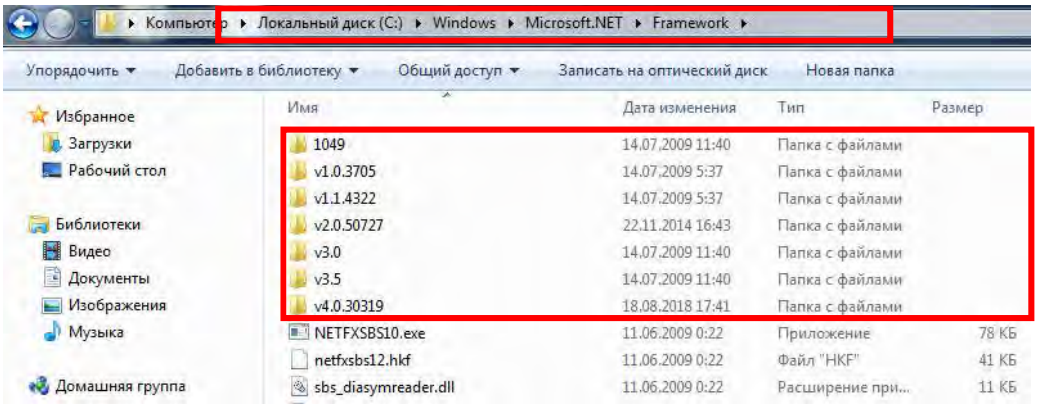


Рис. 2.14. Системна директорія Microsoft.NET

Згідно із даними наведеними в мережі, встановлення .NET Framework версії 4.6 додасть 1,7 секунди до часу завантаження Windows.

Разом із .NET Framework встановлюється служба оптимізації, яка запускає процедури оптимізації в бібліотеках .NET щоразу, коли в системі з'являється оновлення. Це відбувається при першому встановленні нової версії .NET. Під час оптимізації в диспетчері завдань з'явиться файл **Mscorsvw.exe**.

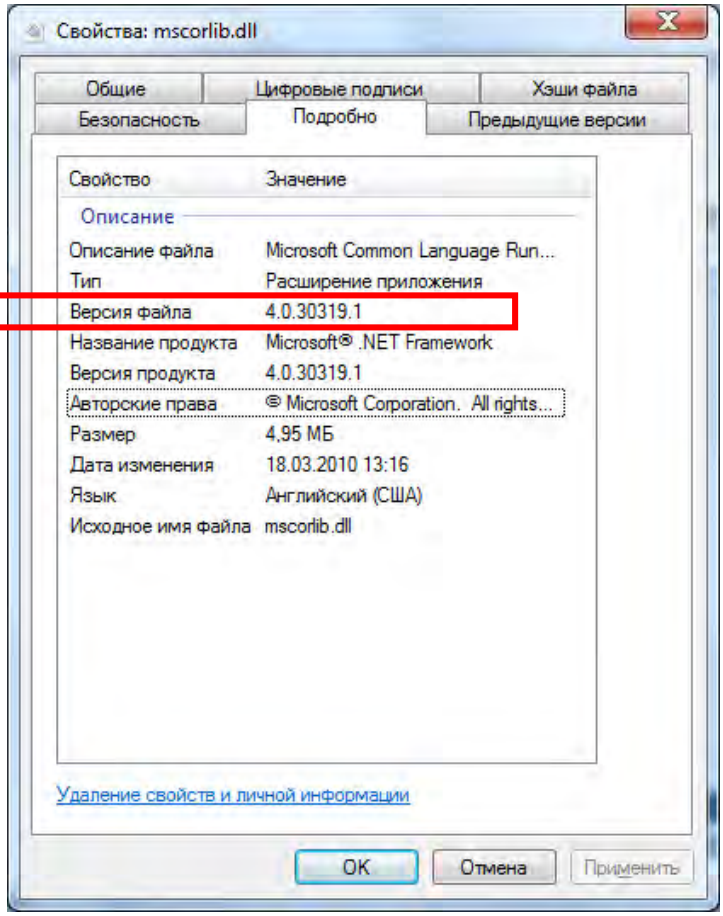


Рис. 2.15. Визначення точної версії .NET Framework

Хоча процес mscorsvw.exe (рис. 2.16) запускається у фоновому режимі, він може запускати цикли процесора під час роботи сервісу. Це відбувається одноразово і займає лише кілька хвилин. На це можна не зважати, оскільки проблеми пов'язані з .NET Framework, спричиняють уповільнення та надмірно високий рівень використання процесора.

З огляду на вказані результати можна зробити висновок, що .NET Framework 4.6 не буде значно уповільнювати роботи ОС. Він додає тисячі ключів реєстру, сервісу та потребує сотень мегабайт на системному диску, але при звичайних умовах роботи користувач може лише помітити, що тільки після встановлення або оновлення виникає затримка на кілька хвилин.

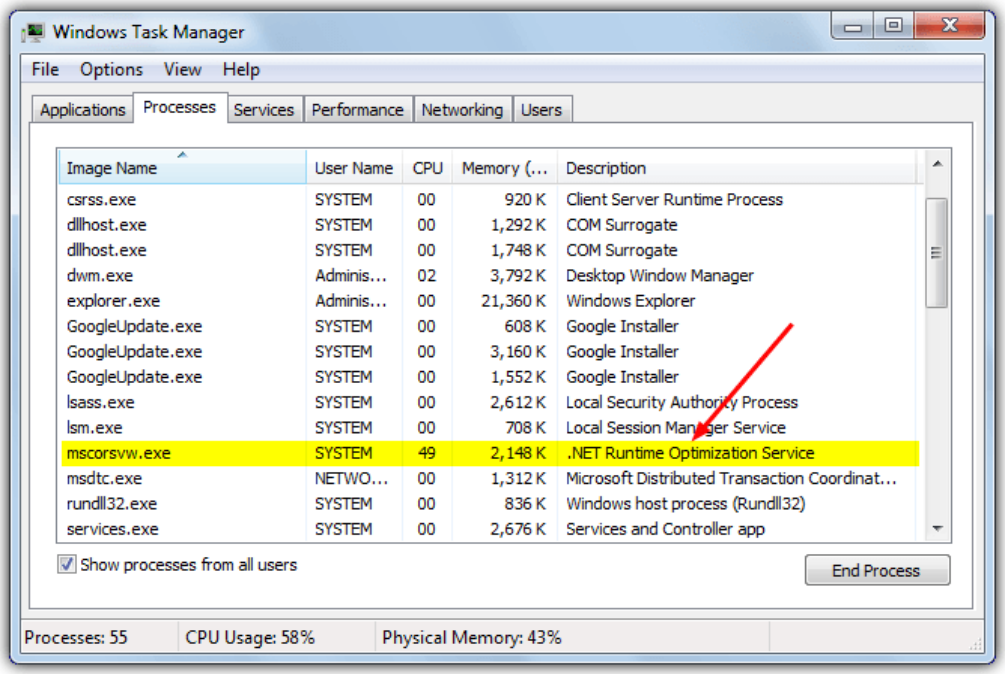


Рис. 2.16. Виконувальний файл Mscorsvw.exe процесу оптимізації

2.1.2. Завдання для самостійного виконання

1. Перевірити в програмі CCleaner цілісність реєстру, видалити записи про помилки при їх наявності.

2. Перевірити потенційні місця записів «троянських програм» в системному реєстрі операційної системи Windows.

3. Перевірити запущені на ПК процеси в диспетчері задач. Виділити основні системні, відмітити підозрілі процеси і відслідкувати місцезнаходження відповідного файлу.

4. Перейти в директорію тимчасових файлів ОС Windows. Перевірити її на наявність підозрілих об’єктів і очистити її зміст.

5. Визначити версію .NET Framework двома способами, навести відповідні скріншоти.

6. Знайти в мережі відомості про останню версію .NET Framework. Зробити висновки про необхідність обновлення поточної версії на ПК.

2.1.3. Запитання для самоперевірки

1. Що таке системний реєстр Windows?
2. Наведіть кореневу структуру реєстру.
3. Поясніть особливості «троянських програм».
4. Чому профілактика «троянських програм» пов'язана з системним реєстром?
5. Які розділи і ключі є потенційними місцями записів «троянських програм »?
6. Поясніть основи роботи антивірусних програм.
7. Як можна знайти віруси самостійно? Де найчастіше вони розташовуються віруси і які профілактичні міри необхідно приймати?
8. Які процеси відносяться до головних системних? Поясніть їх функції.

Тема 2.2. Системне обслуговування ОС Windows при аварійних ситуаціях

Мета: Навчитися зчитувати дамп помилки BSOD і визначати причину її виникнення.

2.2.1. Теоретичні відомості

У даній лабораторній роботі розглянемо таку поширену системну помилку як синій екран смерті або BSOD (the blue screen of death). BSOD з'являється, коли Windows виявляє "STOP-помилку", яку система не в змозі виправити самостійно (рис. 2.11). Таке критичне падіння призводить до зупинки роботи системи Windows. У цьому випадку залишається тільки примусово вимкнути комп'ютер і перезавантажити його. Дана процедура може привести до втрати не збережених даних. В ідеальному сценарії програми повинні регулярно зберігати прогрес роботи, щоб BSOD або інші помилки не привели до втрати даних. Однією з причин перезавантаження комп’ютера без явних причин може бути синій екран.

Рис. 2.11. Синій екран, що викликаний STOP-помилкою

При появі синього екрану смерті Windows автоматично створює та зберігає на диску файл пам'яті "minidump", який містить інформацію про критичний збій. Користувачі можуть переглядати інформацію в дампах, що допоможе ідентифікувати причину падіння ОС.

Помилка «Синій екран смерті» містить в собі наступну інформацію:

1. Назва помилки (важлива інформація).
2. Рекомендації по її усуненню (є стандартним текстом для певних груп помилок).
3. Шістнадцятковий код помилки.
4. Параметри помилки (для деяких помилок є важливою інформацією).
5. Назва драйвера, що викликав помилку (важлива інформація вказується не завжди).
6. Адреса місця, в якому виникла помилка (вказується не завжди).

У випадку відсутності дампу помилки на комп’ютері необхідно її змоделювати. Перед цим переходять у **Властивості комп’ютера → Додаткові параметри системи → Завантаження і відновлення (параметри)**. Після цього знімають галочку з пункту «Виконувати автоматичне

перезавантаження» (рис. 2.12). Тут також можна обрати дані для запису у дамп пам’яті. Доступні такі варіанти:

– малий дамп пам’яті;

– дамп пам’яті ядра;

– повний дамп пам’яті.

На цьому етапі також визначають директорію для запису дампів пам’яті. Для наведеного на рис. 2 прикладу директорія збереження – Windows\Minidump.

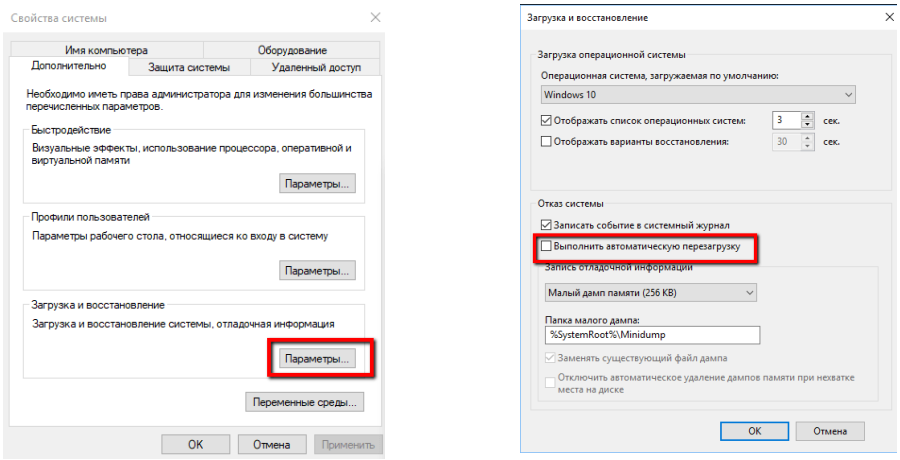


Рис. 2.12. Відміна автоматичного перезавантаження при виникненні STOP-помилок

На наступному кроці викликають диспетчер задач (Ctrl+Alt+Del) і переходять на вкладку «Процеси», де необхідно відобразити процеси всіх користувачів і відшукати серед системних процесів критичні для продовження роботи ОС. Прикладами критичних системних процесів є **csrss.exe, winlogon.exe, wininit.exe** (детальний виклик BSOD через диспетчер задач можна подивитися у навчальному відео до цієї лабораторної роботи).

Змоделювати помилку BSOD також можна, змінивши запис у реєстрі про параметр клавіатури за наступним алгоритмом:

1. Запустити редактор реєстру (за допомогою команди regedit).

2. Відкрити розділ реєстру HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\Services\kbdhid\Parameters (параметри USB клавіатур).

3. Створити DWORD-параметр CrashOnCtrlScroll зі значенням 1.

4. Відкрити розділ реєстру HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\Services\i8042prt\Parameters (параметри клавіатур, підключених через PS/2).

5. Створити DWORD-параметр CrashOnCtrlScroll зі значенням 1.

6. Закрити редактор реєстру і перезавантажити комп'ютер.

Для даної лабораторної роботи я моделювала помилку BSOD шляхом завершення системного процесу wininit.exe, що відповідає за ініціалізацію середовища ОС для виконання програм.

Треба звернути увагу на те, що моделювання помилки BSOD може бути також корисним, наприклад, для отримання дампа пам'яті комп'ютера в заданий момент або для екстреної зупинки комп'ютера аналогічно кнопці Reset.

Після моделювання помилки BSOD необхідно зафіксувати код STOP-помилки, її параметри та виконати перезавантаження ОС.

В останніх версіях Windows вбудований інструмент аналізу причин виникнення помилки BSOD (рис. 2.13).

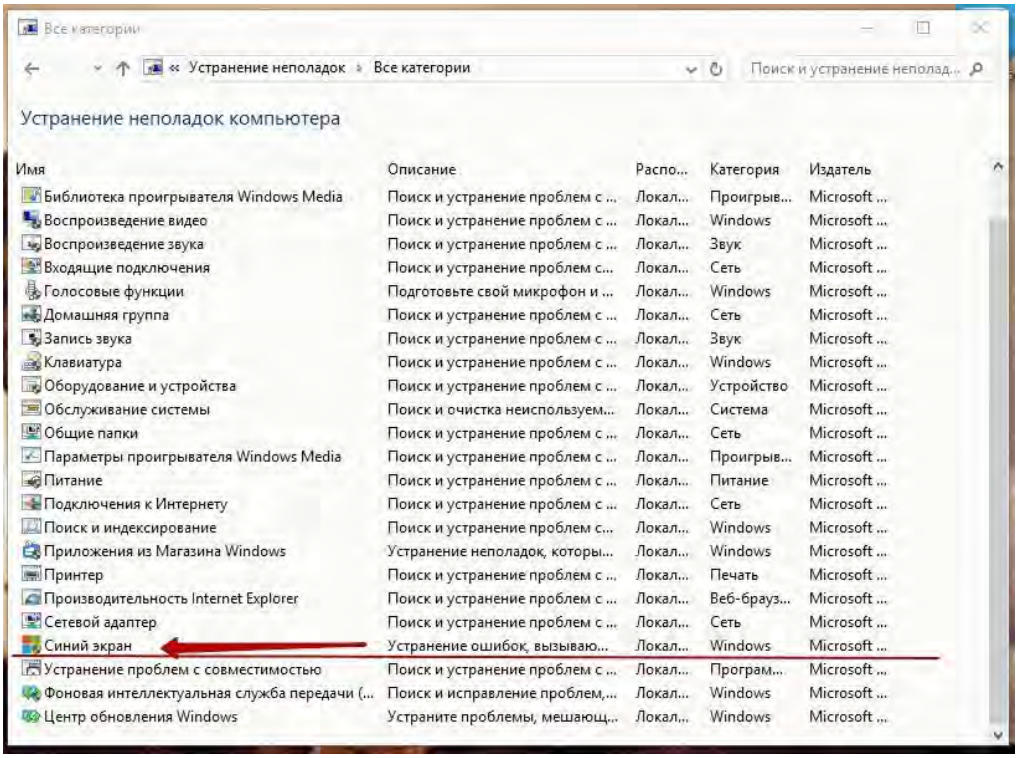


Рис. 2.13. Вбудований інструмент аналізу помилки BSOD

За замовчуванням інструмент буде автоматично застосовувати виправлення, як у випадку з усіма пакетами для усунення неполадок Windows. Якщо необхідно запустити інструмент без автоматичного виправлення, натисніть кнопку «Додатково» і зніміть прапорець «Автоматично застосовувати виправлення». Після цього натисніть кнопку «Далі».

Інструмент інтерпретує код помилки, і повідомляє, чим викликаний синій екран. Можливі варіанти причин помилки:

– несправне обладнання;
– помилки диска;
– шкідливе ПО;
– збій пам'яті;
– сервіси;
– драйвери пристроїв.

Інструмент робить запити про BSOD, які відбулися за останні 7 днів, в журналі подій Windows – Microsoft–Windows–WER–SystemErrorReporting.

У випадку більш ранньої версії Windows переходять у панель управління і обирають категорію «Усунення помилок» (рис. 2.14).

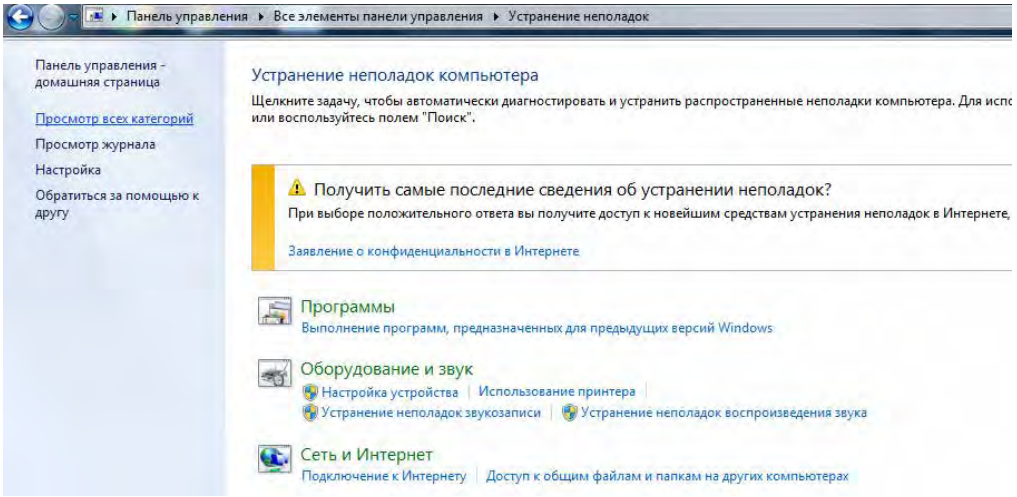


Рис. 2.14. Системна категорія інструментів з усунення помилок

При перегдяді всіх інструментів категорії обирають «Обслуговування системи» і запускають інструмент діагностування і усунення помилок (рис. 2.15).

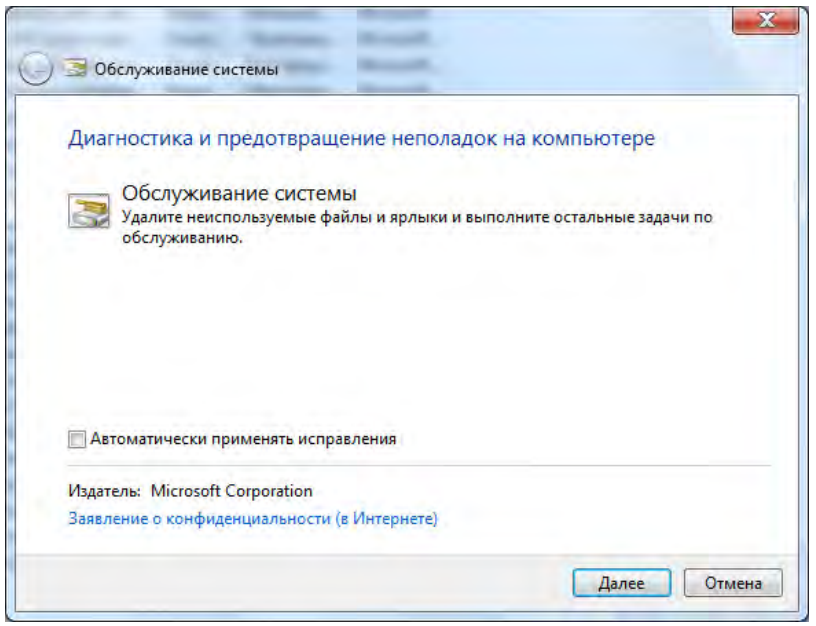


Рис. 2.15. Запуск інструменту діагностування помилок

Після завершення процесу перевірки інструмент надає звіт про виправленні помилки за умови їх наявності (рис. 2.16).

Проаналізувати дампи помилок можна також за допомогою спеціалізованого програмного забезпечення, наприклад Blue Screen View. Програма є повністю безкоштовною і в рази пришвидшує процес визначення проблеми. При запуску Blue Screen View виконує автоматичний аналіз наявності дампу(ів) помилок і виводить їх на головний екран (рис. 2.17).

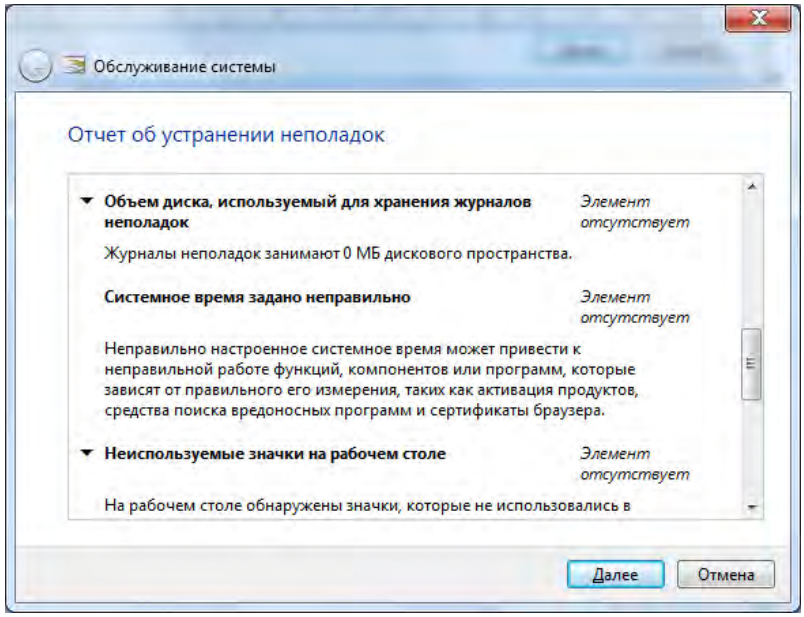


Рис. 2.16. Звіт з виправлення помилок інструменту «Обслуговування системи»

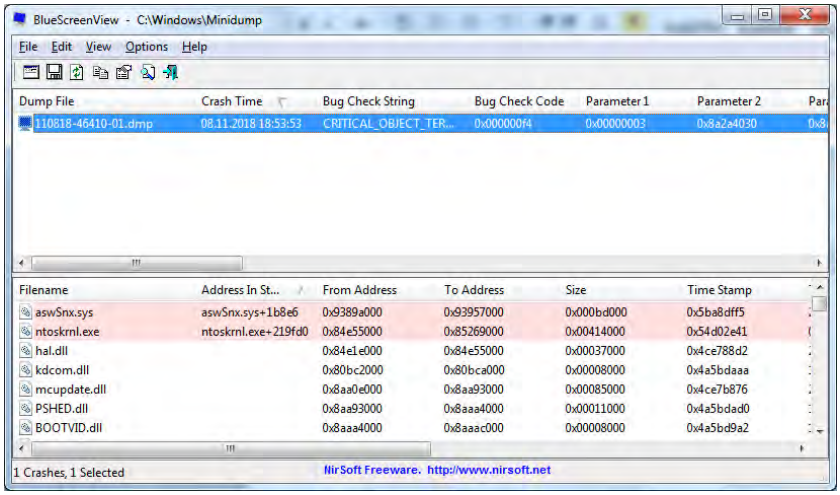


Рис. 2.17. Визначення причин BSOD за допомогою Blue Screen View

Для виявлення безпосередньої причини помилки переходять в меню File → Google Search – Bug Check + Parametr1 (рис. 2.18).

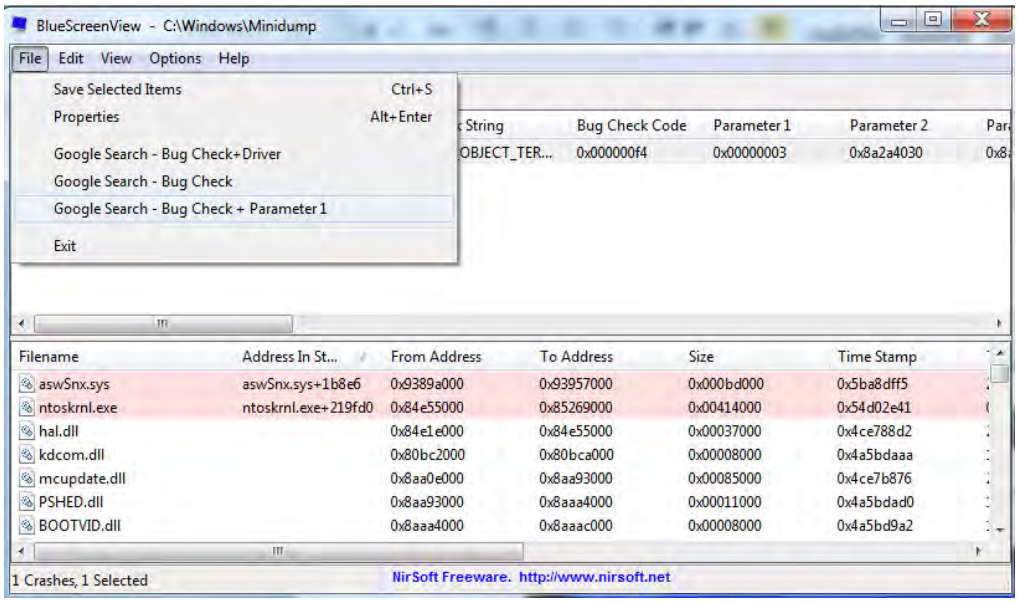


Рис. 2.18. Перехід в Google для визначення причин STOP–помилки

Після переходу за одним із посилань, я отримала інформацію, наведену на рис. 2.19, що повністю відповідає причині виникнення помилки BSOD, змодельованої завершенням системного процесу wininit.exe. Далі на сторінці наведено більш докладний опис помилки, який тут не приводиться.

За замовчанням Blue Screen View шукає дамп помилки в системній директорії Windows/Minidump. Якщо директорія запису дампа відрізняється від зазначеної, переходять на вкладку Options → Advanced Options і прописують коректний шлях.

Іноді аналіз дампа пам'яті вказує на системний драйвер, який навряд чи є причиною проблеми (наприклад, win32k.sys). У цьому випадку знадобиться серйозний аналіз дампа, що вимагає додаткових знань і досвіду в цій галузі. Проте, можна самостійно виконати перевірку драйверів за допомогою вбудованого в операційну систему засобу перевірки Verifier.exe.

Рис. 2.19. Причина STOP–помилки і її опис

У меню Пуск – Виконати вводять verifier і натискають Enter. Запускається засіб перевірки драйверів. Обирають пункт «Створити нестандартні параметри (для коду програм)» і натискають кнопку «Далі» (рис. 2.20).

На наступному кроці обирають пункт «Вибрати окремі параметри з повного списку» і натисніть кнопку «Далі».

Далі ставлять всі прапорці окрім «Імітація браку ресурсів» і натискають кнопку «Далі» (рис. 2.21).

На наступному кроці обирають пункт «Автоматично вибирати непідписані драйвери» і натискають кнопку «Далі». Якщо непідписаних драйверів не виявлено, переходять до вибіркової перевірки драйверів. Якщо непідписані драйвери виявлені, Ви побачите їх список. На цьому етапі не закривають вікно засобу перевірки драйверів і не натискають кнопку «Далі». Драйвери можуть належати як до пристроїв, так і до додатків. У цьому випадку виконують пошук оновлених драйверів і їх встановлення або видалення драйверів.

Пошук оновлених драйверів.

Спочатку перевіряють, чи є оновлені драйвери. Якщо в списку є драйвер додатку, відвідують сайт його виробника – можливо, додаток оновилося. Якщо оновленої версії немає, можна спробувати видалити додаток (завжди можна встановити його заново пізніше). Якщо критичні помилки припиняться, причина була виявлена правильно.

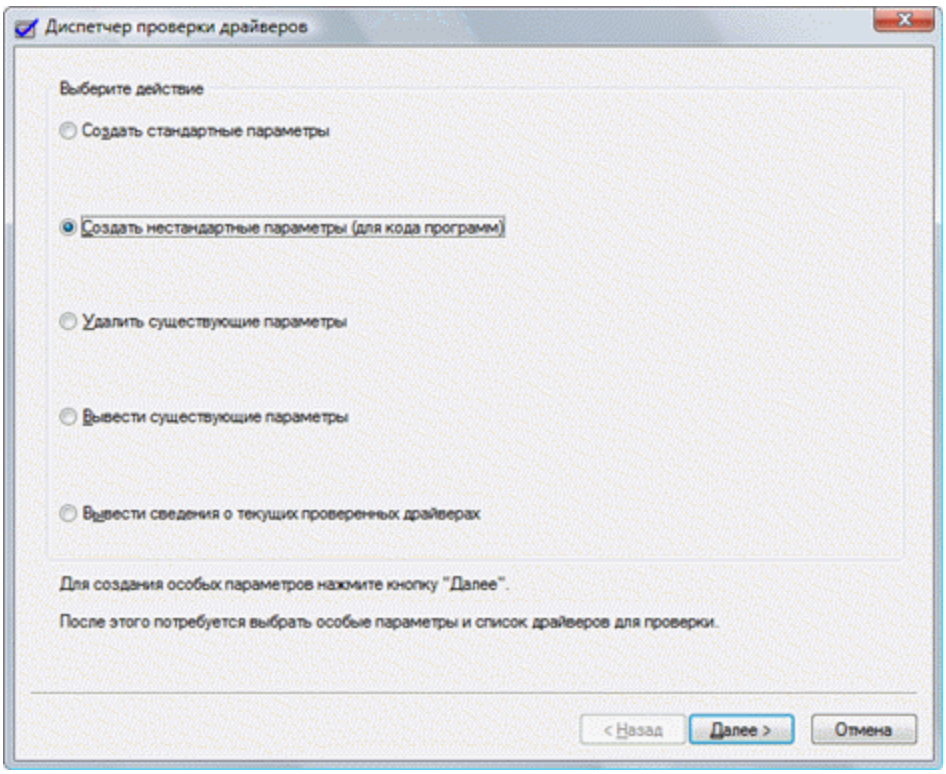


Рис. 2.20. Аналіз працездатності драйверів системним засобом Verifier

Якщо в списку драйвер пристрою, використовують центр оновлення Windows для пошуку нових драйверів. В панелі управління обирають службу Windows Update і виконують перевірку на предмет наявності оновлень для драйвера Вашого пристрою. Якщо драйвер знайдений, встановлюють його.

Якщо Windows Update не запропонує Вам нових драйверів, відвідайте сайт виробника пристрою. Можливо, нові драйвери доступні там.

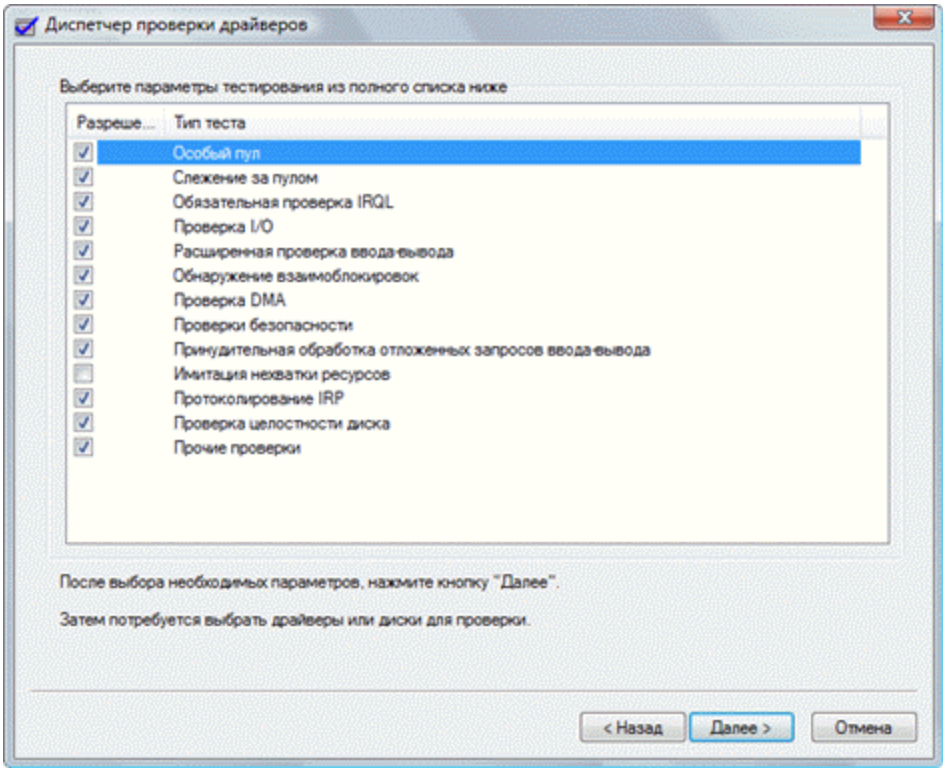


Рис. 2.21. Вибір умов для перевірки драйверів системним засобом Verifier

Після оновлення програми або драйвера закрийте вікно засобу перевірки драйверів, натиснувши кнопку «Відміна», а не «Далі». Перезавантажте комп'ютер і продовжуйте роботу в операційній системі. Якщо критична помилка більше не виникає, проблему вирішено оновленням драйвера.

Видалення драйверів. Якщо нові драйвери не виявлені, драйвер видаляють.

Увага! Видалення драйверів призводить до непрацездатності пристроїв. Після перезавантаження в кращому випадку операційна система встановить відповідний драйвер із сховища драйверів. Якщо Ви не впевнені в тому, чи потрібно видаляти той чи інший драйвер, не видаляйте його.

У диспетчері пристроїв (Пуск – Пошук / Виконати – devmgmt.msc – ОК) знайдіть пристрій, клацніть по ньому правою кнопкою миші і виберіть з контекстного меню пункт

«Властивості». Потім перейдіть на вкладку «Драйвер» і натисніть кнопку «Видалити».

Перевірка непідписаних драйверів

Увага! Після перевірки непідписаних драйверів система може не завантажитися (нижче описано, як діяти в такій ситуації).

Якщо Ви не хочете видаляти драйвер і/або хочете виконати перевірку непідписаних драйверів, у вікні засобу перевірки драйверів натисніть кнопку «Далі». Вам буде запропоновано вибрати фізичний диск.

Виберіть диск, на якому встановлена операційна система, і натисніть кнопку «Готово», після чого перезавантажте комп'ютер. Якщо після перезавантаження ви побачите синій екран з помилкою, проблемний драйвер визначено – його назву буде включено в повідомлення про помилку. Перезавантажте комп'ютер і увійдіть в безпечний режим, натиснувши F8 при завантаженні. Після входу в систему скиньте всі параметри перевірки драйверів, ввівши в Пуск – Пошук/Виконати команду verifier.exe/reset.

Якщо система завантажилася в звичайному режимі, перевірка непідписаних драйверів завершилася успішно – вони не є джерелом проблем. Ви можете побачити список перевірених драйверів, запустивши verifier.exe і вибравши на першому кроці пункт «Вивести відомості про поточні перевірені драйвери».

2.2.2. Завдання для самостійного виконання

1. Відмінити автоматичне перезавантаження ОС при виникненні помилки BSOD. Зафіксувати такі налаштування системи як записувана інформація у дамп пам’яті і директорія розміщення дампу.

2. Змоделювати помилку BSOD одним із способів.

3. Зафіксувати код помилки і її парметри. Виконати перезавантаження системи.

4. Виконати аналіз помилки за допомогою вбудованого системного інструменту «Синій екран» (при наявності).

5. Виконати аналіз помилки за допомогою вбудованого системного інструменту діагностування помилок (при наявності).

6. Виконати аналіз помилки за допомогою програми BlueScreen. Навести аналіз причини і рекомендації по виправленню.

7. Виконати перевірку встановлених драйверів пристроїв за допомогою засобу перевірки Verifier.exe.

2.2.3. Запитання для самоперевірки

1. До якого типу помилок відноситься BSOD? Наведіть приклади причин виникнення помилки BSOD.
2. Назвіть відомі способи моделювання помилки BSOD.
3. Яку інформацію можна записати в дамп помилки BSOD? Поясніть кожний варіант.
4. У чому полягає аналіз дампу помилки BSOD?
5. Які інструменти для аналізу вбудовані в операційну систему Windows? Назвіть функціональні особливості кожного інструменту.

Тема 2.3. Драйвери пристроїв

Мета: навчитися шукати, встановлювати, оновлювати і видаляти драйвери пристроїв.

2.3.1. Теоретичні відомості

Пошук і оновлення драйверів.

Драйвери – це системне програмне забезпечення, що забезпечує стабільну і коректну роботу всіх пристроїв і компонентів, підключених до комп'ютера. Розробники постійно випускають нові версії драйверів з виправленням раніше допущених помилок, тому рекомендується періодично перевіряти наявність оновлень для вже встановлених драйверів.

Встановити або оновити драйвери на Windows 10 можна як за допомогою сторонніх програм, так і стандартними системними утилітами. У роботі розглянемо другий варіант з

використанням диспетчеру пристроїв. Перейти у диспетчер пристроїв можна через «Панель управління» або за допомогою інструменту пошуку.

Існує два способи встановлення та оновлення драйверів: вручну і автоматично. Якщо обрати другий варіант, то комп'ютер сам знайде всі необхідні драйвери і встановить їх, але йому знадобиться стабільний доступ в Інтернет. Цей варіант не завжди працює, оскільки комп'ютер часто не знаходить драйвери.

Установка вручну вимагає самостійно знайти, скачати і встановити драйвери. Шукати їх рекомендовано на сайтах виробників пристроїв, орієнтуючись на назву, унікальний номер і версією драйверів.

Для перегляду унікального номеру необхідно знайти пристрій у диспетчері пристроїв і переглянути його властивості (рис. 2.22).

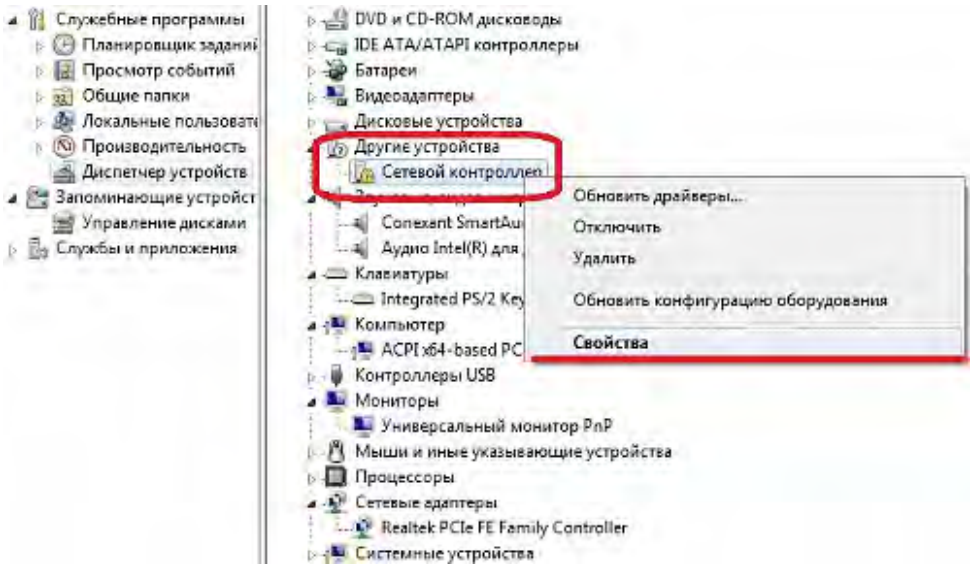


Рис. 2.22. Пошук пристрою в диспетчері

У вікні властивостей необхідно перейти на вкладку «Відомості», на якій у блоці «Властивості» встановити параметр «ІД обладнання» (рис. 2.23). Знайдені цифри є унікальним номером пристрою. Використовуючи їх, можна

визначити, що це за пристрій на сайті розробників, і там же завантажити потрібні драйвери.

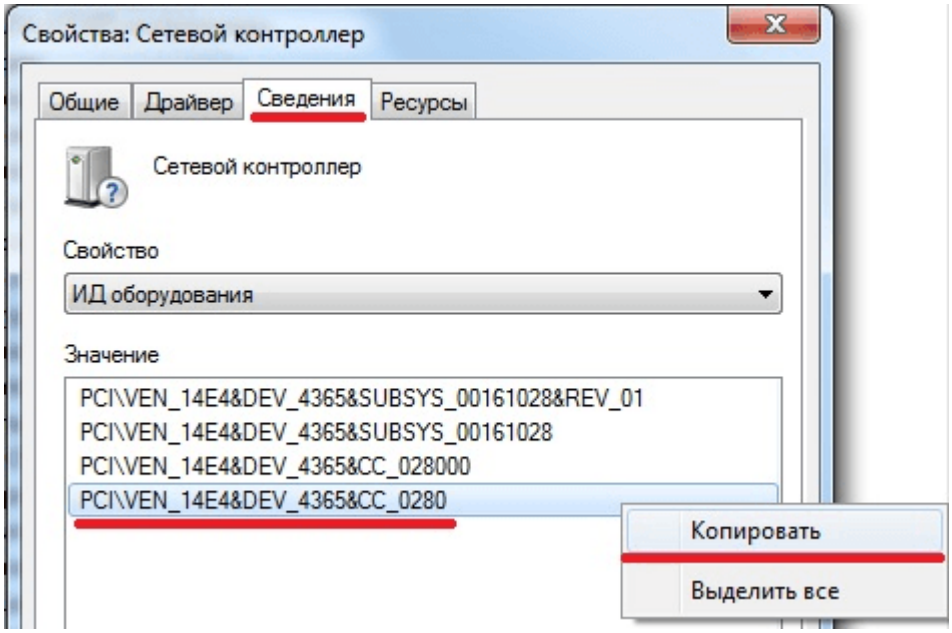


Рис. 2.23. Унікальний номер мережевого контролеру

Знайдений ІД обладнання необхідно скопіювати і знайти відповідний до нього драйвер в Інтернеті. Наприклад, на сайті Microsoft (https://www.catalog.update.microsoft.com/home.aspx). Приклад пошуку, наведений на рис. 2.24.

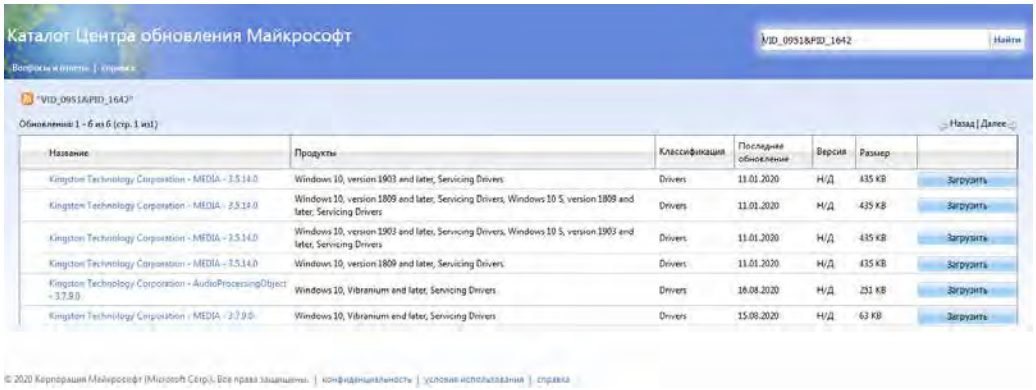


Рис. 2.24. Пошук драйверів

Установка нових драйверів проводиться поверх старих, тому оновлення і установка драйверів – одне і те ж. У разі оновлення або встановлення драйверу через некоректну роботу пристрою, спочатку варто видалити стару версію драйвера. Для цього у властивостях обладнання обирають вкладку «Драйвер» і команду «Видалити» на ній (рис. 2.25).

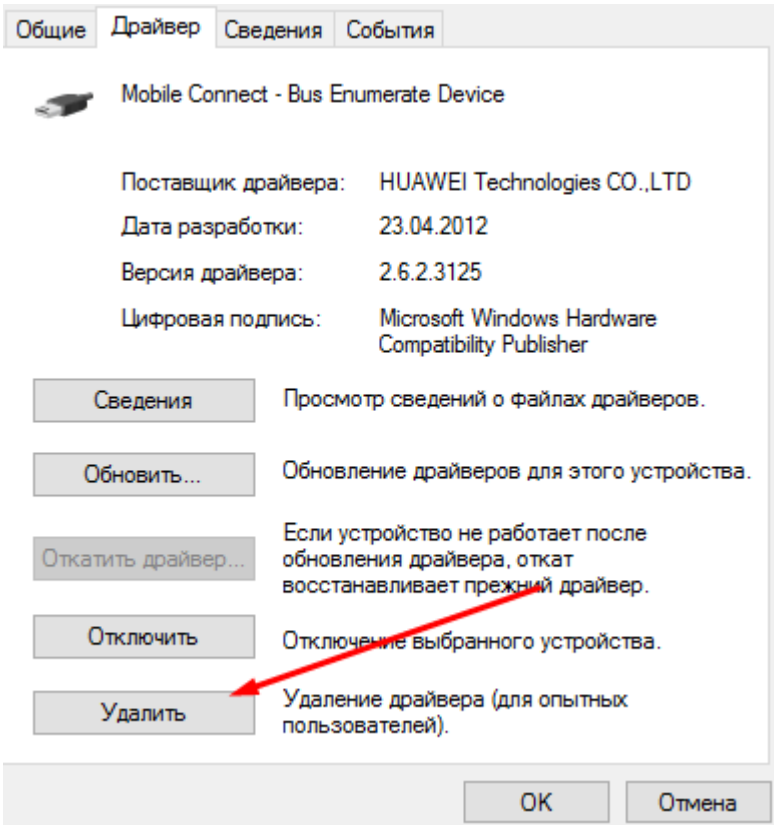


Рис. 2.25. Видалення драйверу пристрою

Після цього необхідно повернутись до головного списку у диспетчері, відкрити контекстне меню для пристрою і обрати пункт «Оновити драйвери». Система запропонує два способи оновлення – автоматичний і вручну. У разі автоматичної перевірки від користувача буде потрібно тільки підтвердження установки знайдених драйверів. При виборі установки вручну необхідно вказати шлях до драйверів, скачаних в одну з директорій жорсткого диска. Після успішного оновлення

драйверів комп'ютер перезапускають для того, щоб внесені зміни вступили в силу.

Відключення перевірки підписів драйверів.

Кожен драйвер має свій сертифікат, який підтверджує його справжність. Якщо система виявить, що у встановлюваного драйвера немає підпису, то роботу з ним буде заборонена. Найчастіше підписів немає у неофіційних драйверів, тобто скачаних не з офіційного сайту розробника пристрою. Але бувають випадки, коли сертифікат драйвера невиявлений в списку ліцензійних з іншої причини. При відключенні перевірки підписів необхідно врахувати, що неофіційні драйвери можуть привести до некоректної роботи пристрою.

Якщо необхідно встановити драйвер без підпису, отриманий з надійного джерела, то спочатку комп'ютер перезавантажують у безпечному режимі роботи. Після завантаження системи запускають командний рядок з правами адміністратора, у якому вводять команду **bcdedit.exe / set nointegritychecks X**, де X – on, щоб деактивувати перевірку, і off, щоб активувати перевірку знову.

Після цього комп'ютер завантажують у звичайному режимі і встановлюють необхідні драйвери.

Управління процесом оновлення драйверів.

За замовчуванням Windows самостійно шукає драйвери і їх нові версії для вбудованих і деяких сторонніх компонентів, але відомо, що не завжди нова версія драйверів краща за стару: іноді оновлення приносять більше шкоди, ніж користі. Тому за оновленням драйверів необхідно стежити вручну, а автоматичну перевірку варто деактивувати.

Як і у випадку з пошуком драйверів спочатку необхідно визначити ІД обладнання і скопіювати його. Після цього переходять у реєстр операційної системи за допомогою сполучення клавіш **Win+R** і команди **regedit** (рис. 2.26).

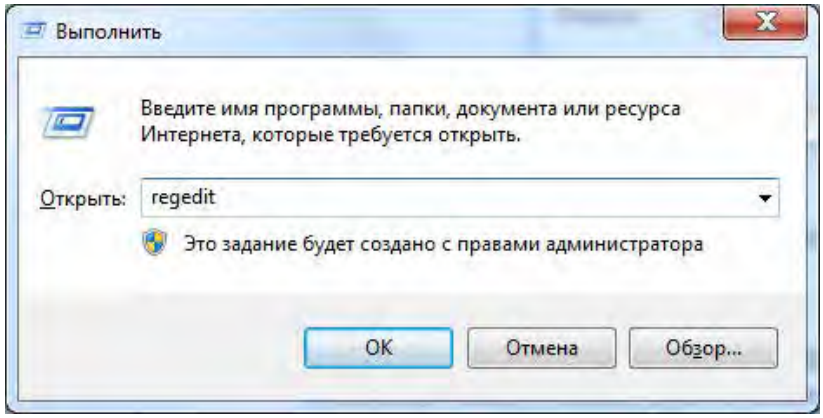


Рис. 2.26. Команда переходу до реєстру

У реєстрі переходять в **HKEY_LOCAL_MACHINE\ SOFTWARE\ Policies\ Microsoft\ Windows\ DeviceInstall\ Restrictions\ DenyDeviceIDs** (рис. 2.27). Якщо на якомусь з етапів розділ буде відсутній, то його необхідно створити вручну для переходу до папки DenyDeviceIDs.

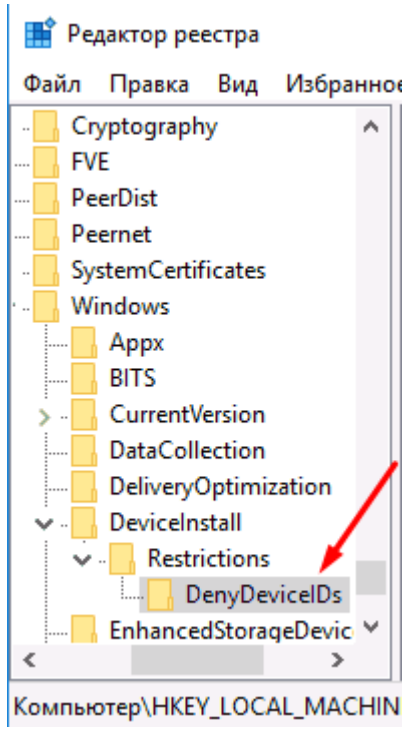


Рис. 2.27. Перехід до папки DenyDeviceIDs

У директорії DenyDeviceIDs необхідно створити окремий параметр для кожного пристрою, драйвери якого не повинні встановлюватися автоматично. Створювані параметри можна називати цифрами, починаючи з одиниці. У значеннях параметрів вказують скопійовані раніше ІД обладнання (рис. 2.28).

Після закінчення процесу реєстр закривають. Оновлення більше не будуть встановлюватися на внесені у список пристрої.

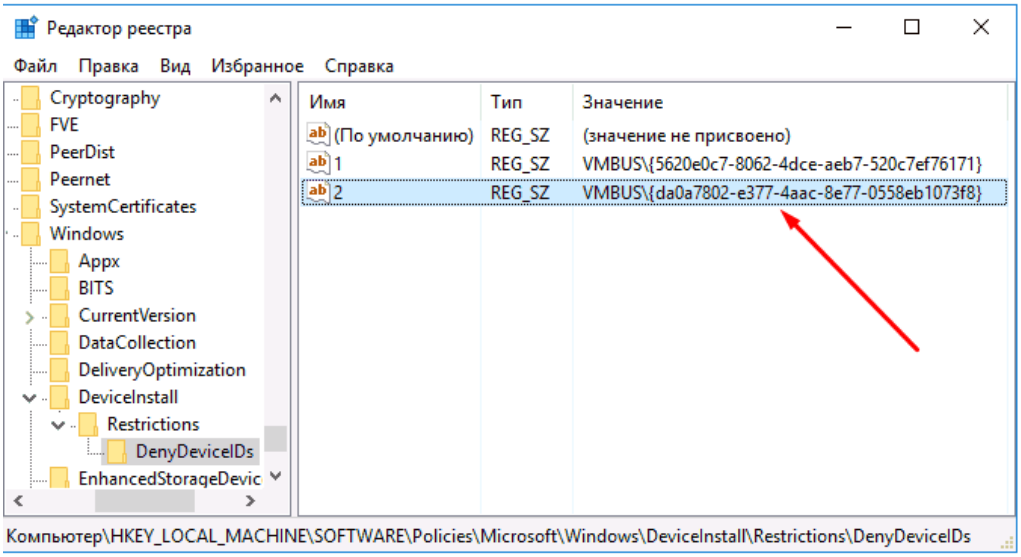


Рис. 2.28. Створення строкових параметрів зі значеннями ID обладнання

Пошук і видалення драйверів, що не використовуються.

У процесі роботи з жорстким диском на ньому відбувається накопичення непотрібних даних: зайві файли, записи в реєстрі після видалення програм, а також драйвера, що не використовуються. Усе це займає місце і призводить до скорочення вільного простору і повільної роботи операційної системи.

Просто так «залишки» драйверів не видалити, тому необхідно виконати наступні кроки:

Перейти до властивостей системи (меню Пуск → Панель управління → Система (рис. 2.29), або меню Пуск → Виконати → ввести команду control.exe/name Microsoft.System).

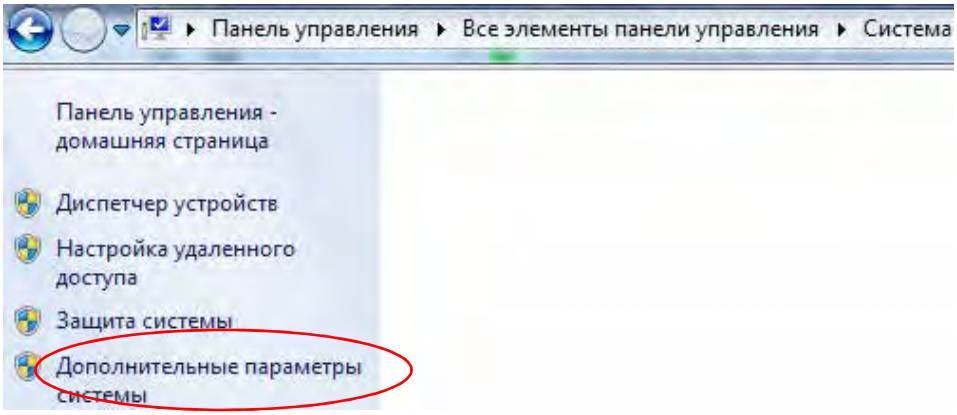


Рис. 2.29. Параметри системи

Обрати пункт «Додаткові параметри системи» і на вкладці «Додатково» натиснути кнопку «Змінні середовища» (рис. 2.30).

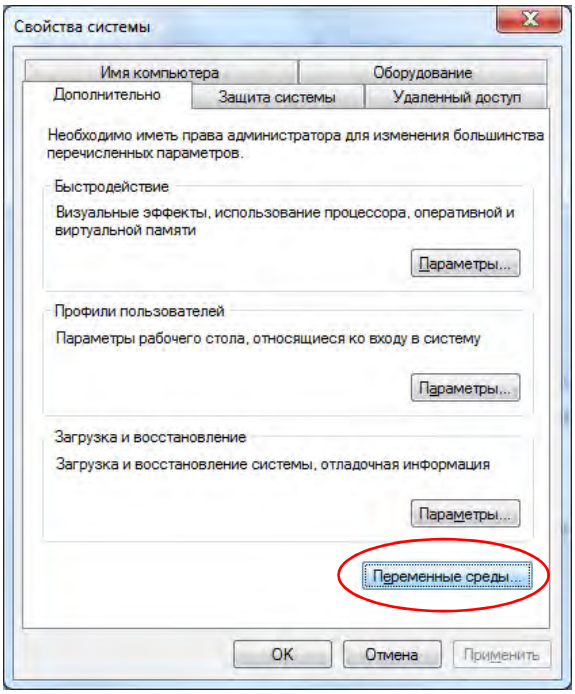


Рис. 2.30. Додаткові властивості системи

У вікні «Змінні середовища» необхідно створити нову змінну, для цього натискають кнопку «Створити». У діалоговому вікні змінній присвоюють ім’я – **devmgr_show_nonpresent_devices** і надають значення – 1 (рис. 2.31). Після вводу параметрів і натискання на кнопку «ОК» нова змінна з’явиться в списку.

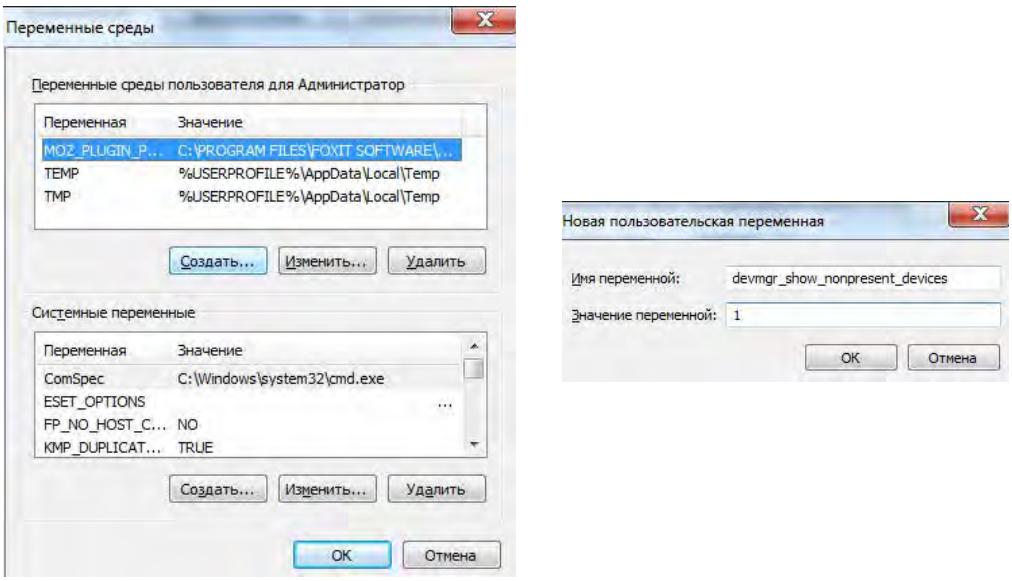


Рис. 2.31. Створення нової змінної користувача

Далі переходять на вкладку «Устаткування», де обирають команду «Диспетчер пристроїв». На виконання команди необхідний деякий час для побудови дерева пристроїв. У меню диспетчера «Вид» обирають команду «Показати приховані пристрої» (рис. 2.32)

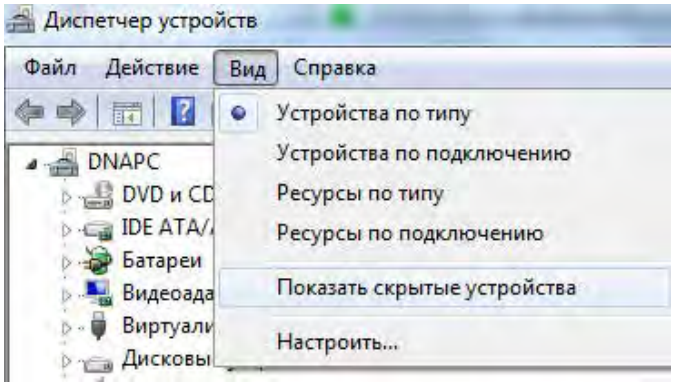


Рис. 2.32. Відображення прихованих пристроїв

Після цього в дереві пристроїв можна переглянути всі пристрої та встановлені для них драйвери, а також побачити «залишки» від видалених раніше програм. Неактивні драйвери пристроїв та компоненти будуть відображені сірим кольором. Наприклад, у розділі «Дискові пристрої» сірим відображені драйвери флеш-накопичувачів, які були під'єднані до комп'ютера за весь період його роботи (рис. 2.33).

У розділі «Несамоналаштовувані пристрої» можна побачити залишкові компоненти видалених програм (рис. 2.34).

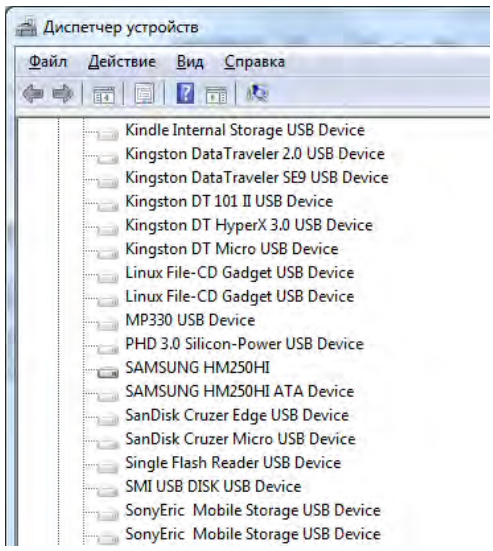


Рис. 2.33. Невикористовуванні драйвери пристроїв

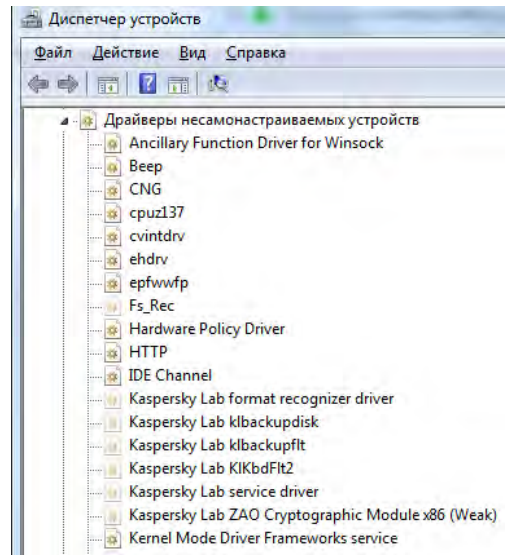


Рис. 2.34. Залишкові компоненти видалених програм

Для видалення непотрібного драйверу пристрою або компонента програми використовують команду «Видалити» з контекстного меню (рис. 2.35).

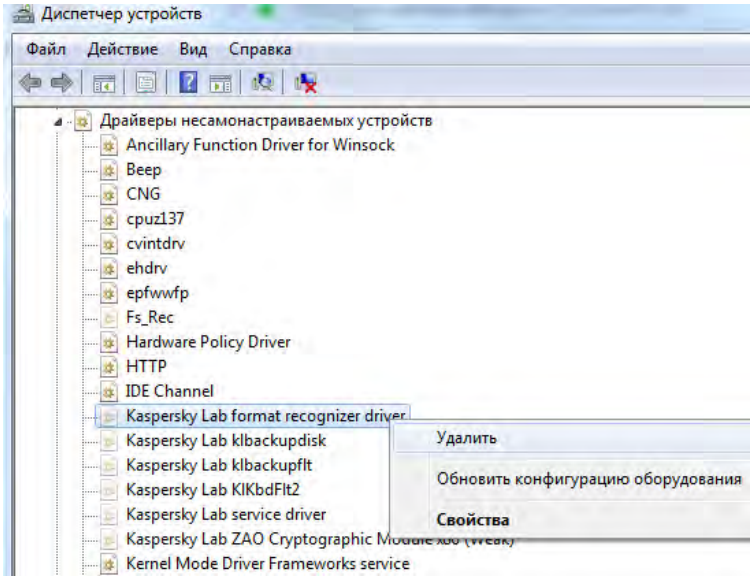


Рис. 2.35. Видалення «залишків» програми Kaspersky Antivirus

Повернення до старої версії драйверу.

Іноді після установки нових версій драйверів обладнання – відеокарти, системних пристроїв, звукової плати, можна зіткнутися з тим, що воно починає працювати неправильно. У цьому випадку одна з перших речей, які слід спробувати – повернутися до старої версії драйверу.

У Windows 10, як і в попередніх версіях ОС передбачений вбудований метод повернення драйверів пристроїв до їх попередніх версій при необхідності. Для цього у диспетчері пристроїв необхідно знайти потрібний пристрій, і обрати пункт «Властивості» з контекстного меню (рис. 2.36).

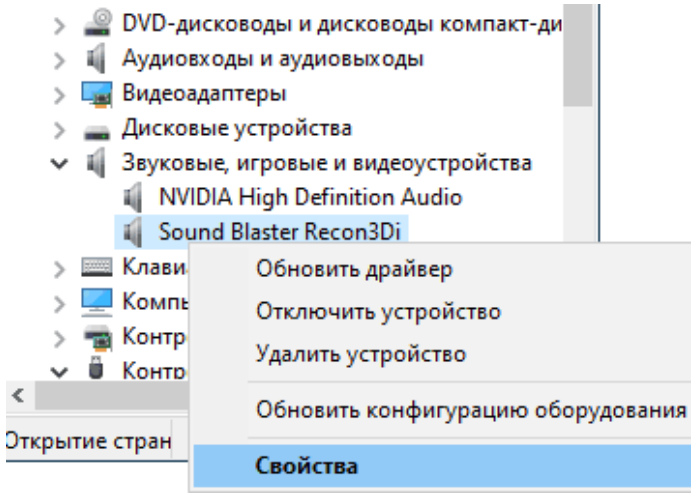


Рис. 2.36. Властивості пристрою

Потім перейти на вкладку «Драйвер» і перевірити, чи активна кнопка «Повернутися». Якщо так – натиснути її (рис. 2.37).

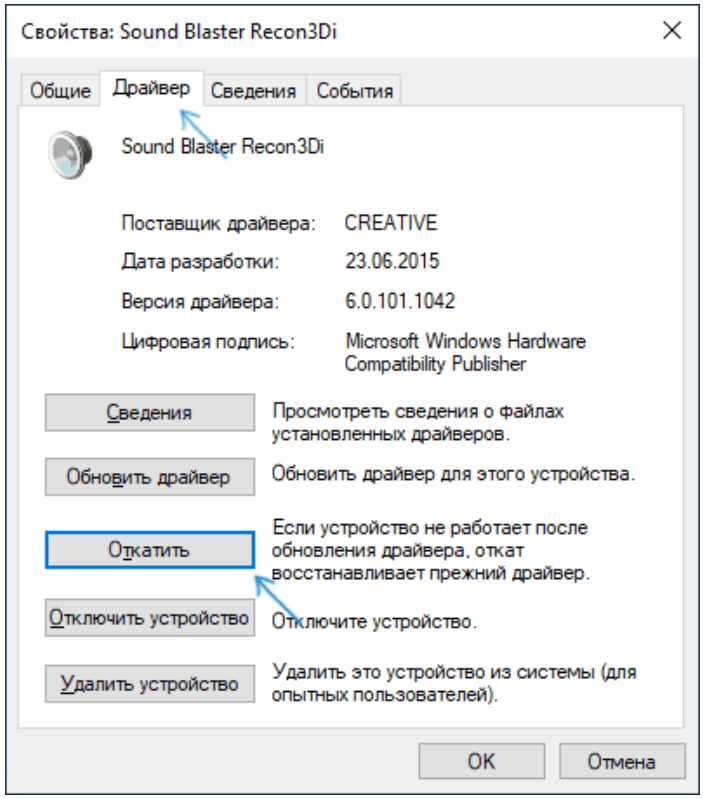


Рис. 2.37. Повернення до попередньої версії драйвера

У процесі повернення до старої версії драйверу у Windows 10 необхідно вказати його причину (рис. 2.38).

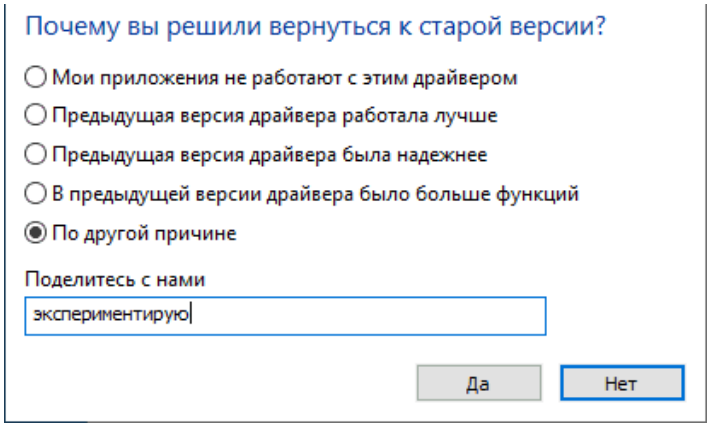


Рис. 2.38. Причина повернення до драйвера старої версії

Відразу після цього драйвер пристрою буде видалений, а замість нього встановлена попередня версія драйвера або універсальний системний драйвер. Іноді після описуваних дій також потрібно виконати перезавантаження комп'ютера, щоб старий драйвер був задіяний в якості використовуваного пристрою.

Якщо кнопка «Повернутися» у властивостях драйвера не активна, тоді необхідно знайти та завантажити драйвер потрібної версії, в ідеалі – з офіційного сайту виробника материнської плати, ноутбука або обладнання. Після цього обрати пристрій в диспетчері пристроїв, і в його контекстному меню обрати пункт «Видалити пристрій». У наступному вікні поставити галочку навпроти пункту «Видалити програми драйверів для цього пристрою» (рис. 2.39).

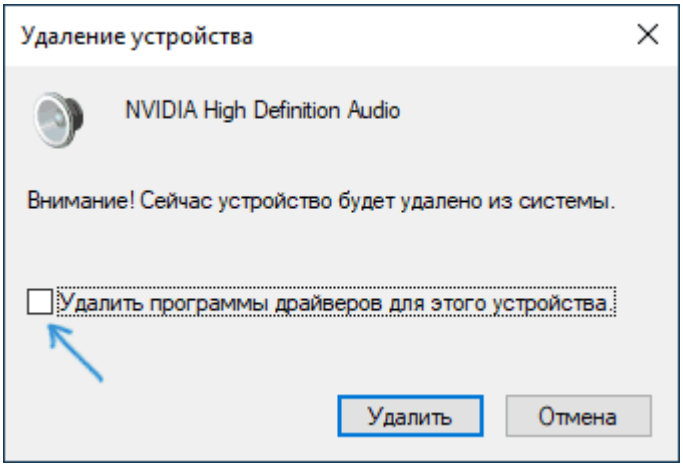


Рис. 2.39. Видалення драйвера пристрою

Після видалення пристрою необхідно встановити завантажений раніше драйвер.

2.3.2. Завдання для самостійного виконання

1. Ознайомтеся з роботою диспетчера пристроїв. Знайдіть ІД обладнання для основних компонентів комп'ютера: процесора, монітора, відеоадаптера, мережевого адаптера і т.д.
2. Перевірте актуальність встановлених драйверів для основних компонентів комп'ютера. При необхідності виконайте пошук актуальних оновлень.
3. Створіть нову змінну користувача для пошуку драйверів пристроїв, що не використовуються і компонентів видалених програм.
4. Виконайте пошук вищезазначених драйверів і компонент видалених програм і з подальшим їх видаленням.
5. Виконайте для декількох пристроїв оновлення драйверів в автоматичному режимі.
6. Налаштуйте для декількох пристроїв режим ручного оновлення драйверів за допомогою реєстру.
7. Відпрацюйте встановлення нової версії драйверу в ручному режимі для обраного компонента і повернення до його старої версії.

6.

2.3.3. Запитання для самоперевірки

1. Що таке драйвер? Для чого випускають нові версії драйверів?
2. Як визначити ІД обладнання? Наведіть приклади використання ІД обладнання при роботі з драйверами.
3. Як переглянути версію і стан драйверу пристрою?
4. Дайте визначення змінній середовища. Наведіть приклади застосування змінних.
5. Чи можна налаштувати ручний режим оновлення драйверів для всіх пристроїв?
6. Чому після видалення програм у системі можуть залишатися їх компоненти? Чи можна їх видалити?
7. Що таке цифровий підпис драйвера? Як можна відмінити його перевірку? У яких випадках це потрібно?
8. У яких випадках може бути потрібне повернення до старої версії драйверу?
9. Назвіть переваги і недоліки ручного і автоматичного режимів оновлення драйверів.
10. Назвіть етапи видалення драйвера, що не працює, і встановлення нового.

Тема 2.4. Керування процесами

Мета: навчитися керувати процесами різних категорій за допомогою диспетчера задач і командного рядка.

2.4.1. Теоретичні відомості

Розподіл ресурсів комп'ютера відбувається між процесами і задачами, які на ньому запущені. Залежно від їх кількості і об'єму необхідних ресурсів, швидкість роботи комп'ютера може змінюватися. Отже, збоку користувача важливо контролювати актуальність запущених процесів і задач.

Для того, щоб переглянути активні процеси і задачі переходять у диспетчер задач натисканням сполучення клавіш **Ctrl+Alt+Delete** або **Ctrl+ Shift+Esc**. На вкладці «Процеси» у лівому нижньому куті можна встановити показ процесів усіх

користувачів. У стовпці «Пам’ять» можна дізнатися, скільки оперативної пам’яті витрачає той чи інший процес (рис. 2.40).

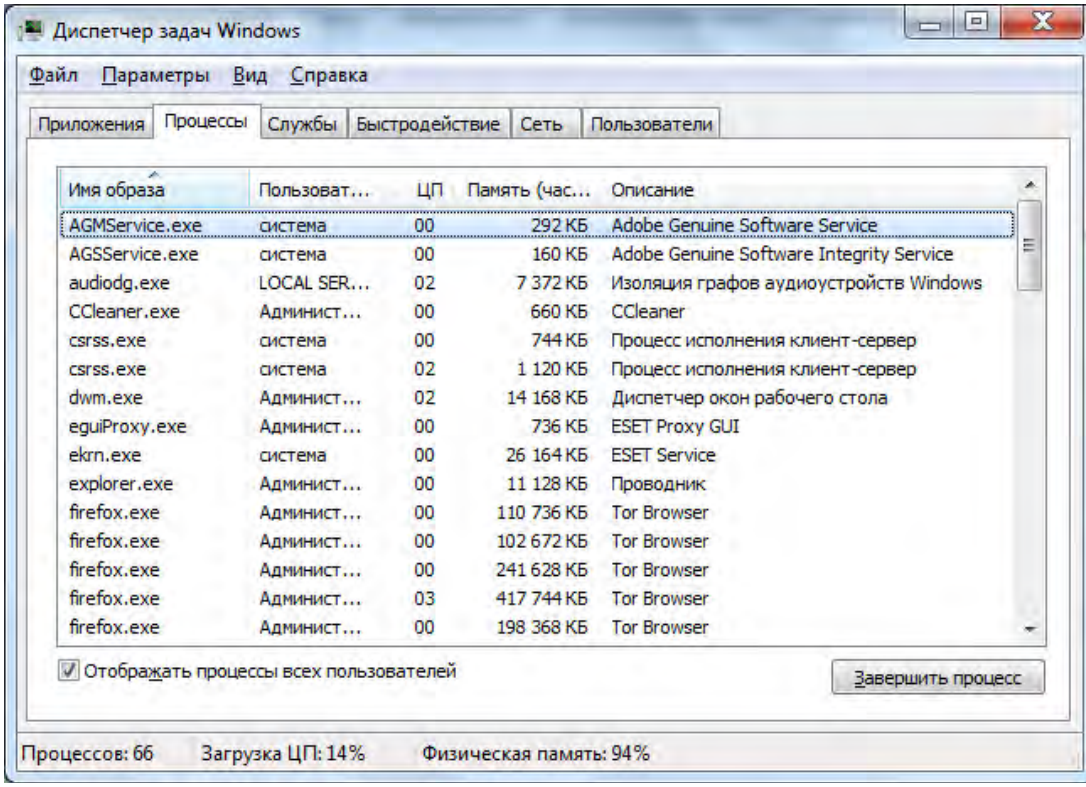


Рис. 2.40. Список процесів у диспетчері задач

Усі процеси діляться на дві категорії:

1. Системні – призначені для роботи операційної системи в цілому, і окремих її елементів.
2. Процеси користувача – запущені від імені адміністратора, тобто власника комп’ютера.

У стовпці «Користувач» можна побачити, до якої категорії належить той чи інший процес. Процеси користувача в більшості випадків необхідно завершувати. До цієї категорії відносяться програми і служби, які були встановлені самим користувачем. Однак є й деякі винятки. При спробі закрити процес explorer.exe з робочого столу пропадуть ярлики і панель задач. У табл. 2.4 наведені приклади процесів, які не можна завершувати.

Таблиця 2.4.

Приклади системних процесів

Назва	Опис
Explorer	Підтримка правильної роботи робочого столу, панелі задач.
Wininit	Автозавантаження додатків
Winlogon	Програма входу в систему
Lsass	Авторизація локальних користувачів
Lsm	Служба локальних сеансів
Services	Додаток служб Windows
SearchIndexer	Індексатор пошуку

У табл. 2.5 представлені приклади процесів, які необхідно завершити, якщо вони не потрібні користувачу в даний момент.

Таблиця 2.5.

Приклади процесів користувача

Назва	Опис
hkcmd	Супровід апаратних засобів Intel
AdobeARM	Перевірка наявності оновлень для програми AcrobatReader
reader_sl	Прискорює запуск AcrobatReader
Jusched	Перевірка наявності оновлень для Java
winampa	Процес плеєра Winamp
sidebar	Супровід гаджетів робочого столу
OSPPSVC	Платформа захисту MS Office

Для того щоб завершити процес, необхідно в диспетчері задач натиснути на ньому правою клавішею миші, обрати команду «Завершити процес», а після появи попереджуючого вікна, підтвердити свою дію. Таким чином непотрібний процес буде закритий, проте в більшості випадків при повторному завантаженні системи він буде знову запущений.

Усі задачі і процеси, незалежно від групи мають пріоритет. За замовчуванням в Windows 10 всім процесам виставляється

звичайний пріоритет. При необхідності можна змінити пріоритет окремого процесу. Зміну пріоритетів рекомендується використовувати на слабких комп'ютерах, де немає великого запасу обчислювальної потужності процесора. Саме на таких системах спостерігається підвищення продуктивності вимогливих програм.

У наявності є таке основні пріоритети:

Реального часу – всі ресурси системи будуть задіяні насамперед для виконання даного процесу. Додаток, отримавши таке значення пріоритету, буде отримувати будь-яку необхідну потужність, в деяких випадках навіть за рахунок інших процесів.

Високий – не обмежуючи інші процеси, використовує максимально доступну кількість ресурсів. Всі процеси з високим пріоритетом працюють краще в порівнянні з попередніми пріоритетами. Він здатний використовувати ресурси процесів з пріоритетом нижче.

Звичайний – більшості додатків досить цього пріоритету для нормальної роботи. Поточне значення вибирається автоматично для всіх встановлених програм.

Низький – обраний процес використовує ресурси системи, тільки коли вони повністю вільні. У результаті роботи додатків в такому режимі вони будуть відставати від реального часу.

Зміну пріоритету у попередніх версіях Windows виконують за допомогою команди «Пріоритет» контекстного меню процесу. У останніх версіях Windows переходять у розділ «Подробиці» і в контекстному меню процесу обирають команду «Призначити пріоритет» (рис. 2.41).

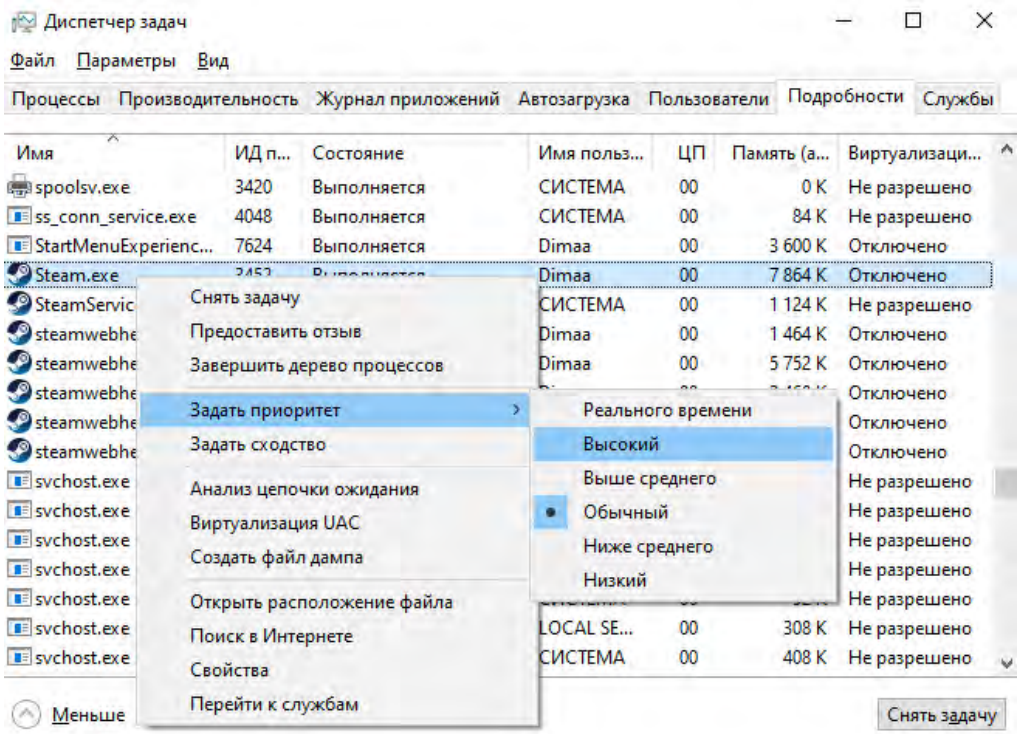


Рис. 2.41. Зміна пріоритету процесу

Після зміни пріоритету не потрібно виконувати перезавантаження комп'ютера або застосовувати зміни. Пріоритет буде змінений в режимі реального часу.

Після перезавантаження комп'ютера або при перезапуску програми встановлений пріоритет скидається, тобто Windows знову задає його автоматично. Але за допомогою налаштувань реєстру вказаний пріоритет можна зберегти. Для цього необхідно відкрити редактор реєстру за допомогою сполучення клавіш **Win+R**, і ввести команду **regedit**.

Далі необхідно відкрити наступну гілку реєстру: HKEY_LOCAL_MACHINESOFTWAREMicrosoftWindowsNTCurrentVersion Image File Execution Options (рис. 2.42).

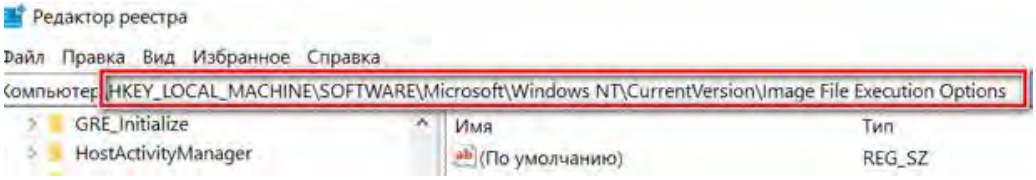


Рис. 2.42. Перехід по реєстру

Потім необхідно дізнатися точну назву виконуваного файлу програми. Для цього натискають правою кнопкою мишки по її ярлику і переходять у властивості. У розділі «Ярлик» в рядку «Об'єкт» буде вказана назва виконуваного файлу.

Після цього у відкритій гілці реєстру створюють розділ (папку) з ім'ям виконуваного файлу програми без лапок і маленькими буквами. На наступному кроці у гілці з назвою програми створюють ще один розділ (папку) з назвою «PerfOptions».

У розділі «PerfOptions» створюють параметр DWORD (32), надають йому ім'я «CpuPriorityClass» і задають значення виходячи з вимог (1 – низький пріоритет; 5 – нижче середнього; 8 – середній; 6 – вище середнього, 3 – високий). У підсумку вийде так, як показано на рис. 2.43.

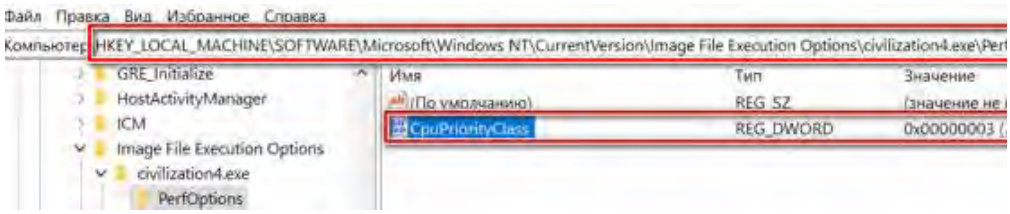


Рис. 2.43. Створення параметру DWORD (32) у розділі «PerfOptions»

У випадку, коли робота з диспетчером задач неможлива, наприклад шкідливі програми можуть блокувати його запуск, використовують командний рядок. Для управління процесами в командному рядку є дві утиліти – **tasklist** і **taskkill**. Перша показує список процесів на локальному або віддаленому комп'ютері, друга дозволяє їх завершити.

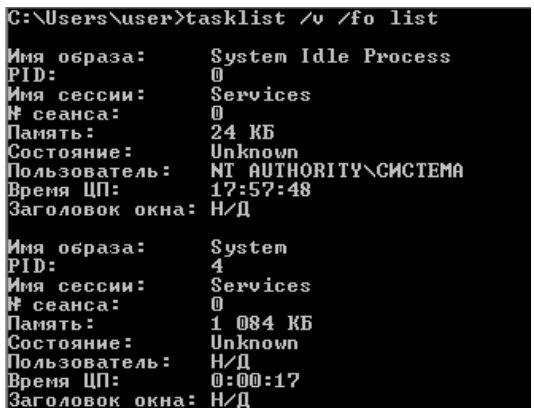
Якщо набрати команду tasklist в командному рядку, то вона видасть список процесів на локальному комп'ютері (рис. 2.44).

```
C:\Users\user>tasklist

Имя образа                     PID Имя сессии          № сеанса       Память
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0         24 КБ
System                           4 Services                   0      1 084 КБ
smss.exe                       256 Services                   0      1 200 КБ
csrss.exe                      336 Services                   0      4 272 КБ
csrss.exe                      384 Console                    1      4 584 КБ
wininit.exe                    392 Services                   0      4 544 КБ
winlogon.exe                   420 Console                    1      7 280 КБ
services.exe                   480 Services                   0      8 032 КБ
lsass.exe                      488 Services                   0     10 212 КБ
lsm.exe                        496 Services                   0      4 200 КБ
svchost.exe                    592 Services                   0      9 860 КБ
svchost.exe                    672 Services                   0      7 512 КБ
svchost.exe                    756 Services                   0     17 056 КБ
svchost.exe                    804 Services                   0     12 936 КБ
svchost.exe                    832 Services                   0     32 796 КБ
```

Рис. 2.44. Приклад роботи команди tasklist

За замовчуванням інформація виводиться у вигляді таблиці, проте ключ **/ fo** дозволяє задати вивід у виді списку або в форматі CSV, а ключ **/ v** показує більш детальну інформацію про процеси, наприклад команда **tasklist /v /fo list** виведе докладний опис всіх процесів у виді списку (рис. 2.45). Список вийде досить великий, тому спробуємо уточнити запит. Для цього використовуємо ключ / fi, який дозволяє використовувати фільтри для виведення даних, наприклад команда tasklist /fi "username eq user" /fi "memusage le 40000" виводить список процесів користувача user, які споживають не більш 40Мб пам'яті (рис. 2.46).



```
C:\Users\user>tasklist /v /fo list

Имя образа:     System Idle Process
PID:            0
Имя сессии:     Services
№ сеанса:       0
Память:         24 КБ
Состояние:      Unknown
Пользователь:   NT AUTHORITY\СИСТЕМА
Время ЦП:       17:57:48
Заголовок окна: Н/Д

Имя образа:     System
PID:            4
Имя сессии:     Services
№ сеанса:       0
Память:         1 084 КБ
Состояние:      Unknown
Пользователь:   Н/Д
Время ЦП:       0:00:17
Заголовок окна: Н/Д
```

Рис. 2.45. Приклад роботи команди tasklist з ключами

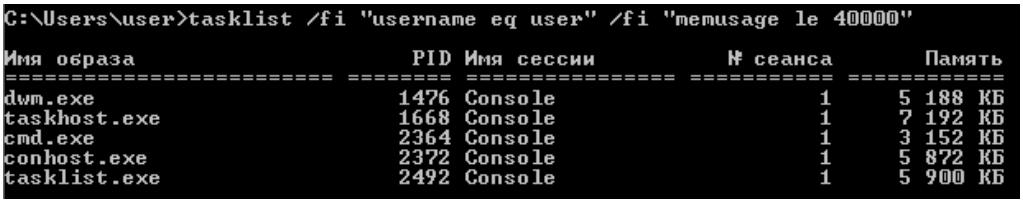


Рис. 2.46. Приклад роботи команди tasklist з ключем / fi

Знайшовши процеси, які необхідно завершити, використовують команду **taskkill**. Завершувати процеси можна по імені, ідентифікатору (PID) або задавши умови за допомогою фільтрів. Для прикладу було запущено декілька примірників програми notepad.exe – блокнот. Спробуємо завершити даний процес різними способами (рис. 2.47). Ключ /f завершує процес примусово, а /t завершує всі дочірні процеси. Довідку по командам tasklist і taskkill можна отримати, ввівши їх з ключем /?.

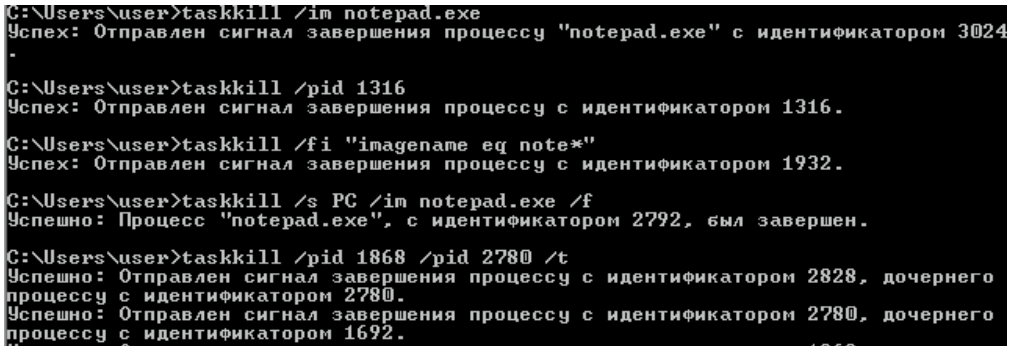


Рис. 2.47. Приклад роботи команди taskkill

2.4.2. Завдання до самостійного виконання

1. Здійснити вхід до диспетчера задач від імені адміністратора. Переглянути процеси всіх користувачів.

2. Виділити серед запущених процесів системні і процеси користувачів. Визначити скільки оперативної пам’яті витрачає кожна з категорій процесів.

3. Обрати процес, для якого змінити пріоритет у диспетчері задач. Спочатку підвищити його, потім понизити.

Після кожної зміни запустити програму і порівняти швидкість роботи в ній.

4. Переглянути список процесів за допомогою командного рядка. Залишити тільки ті процеси, які споживають не більш 50Мб пам’яті.

5. Обрати один з процесів користувача. Виконати його завершення за допомогою командного рядка.

2.4.3. Запитання для самоперевірки

1. Яким чином здійснюється вхід до диспетчера задач? Як у диспетчері переглянути процеси всіх користувачів?

2. Як можна дізнатися, скільки оперативної пам’яті витрачає той чи інший процес?

3. Назвіть категорії процесів і наведіть їх приклади відповідно до категорій.

4. Дайте визначення поняття «пріоритет процесу». Назвіть і охарактеризуйте основні пріоритети. Який пріоритет мають всі процеси за замовчуванням?

5. Який алгоритм зміни пріоритету процесу у диспетчері задач? Чому зміни у цьому випадку не зберігаються після перезавантаження ПК?

6. Який алгоритм зміни пріоритету процесу у реєстрі? Чому зміни у цьому випадку зберігаються після перезавантаження ПК?

7. За допомогою яких команд і ключів можливі перегляд і керування процесами в командному рядку?

РОЗДІЛ 3. СТИСНЕННЯ ІНФОРМАЦІЇ В КОМП'ЮТЕРНИХ СИСТЕМАХ

Тема 3.1. Вступ до алгоритмів стиснення інформації

Стиснення використовується практично всюди. Всі зображення, які ви отримуєте в Інтернеті, стискаються, як правило, в форматах JPEG або GIF, більшість модемів використовують стиснення, HDTV стискається з використанням MPEG-2, а деякі файлові системи автоматично стискають файли при збереженні, а решта роблять це за запитом користувача. Відмінна річ у задачах стиснення, які розглянемо у подальшому, складається в тому, що алгоритми, які використовуються в реальному світі, використовують широкий набір алгоритмічних інструментів, включаючи сортування, хеш-таблиці, FFT та багато чого іншого. Крім того, алгоритми з сильною теоретичної основою грають критично важливу роль в реальних додатках.

Будемо використовувати загальний термін *message* для об'єктів, які ми хочемо стиснути. Це можуть бути файли або повідомлення. Завдання стиснення полягає з двох компонентів: алгоритму *кодування,* який приймає повідомлення і генерує «стислий» зміст (як можливо з меншею кількістю бітів), і алгоритму *декодування*, який відновлює вихідне повідомлення або деяку його апроксимацію зі стисненого представлення. Ці дві компоненти зазвичай пов'язані одна з одною, оскільки вони обидві повинні розуміти загальне стисле представлення.

Розрізняють *алгоритми без втрат,* які можуть реконструювати вихідне повідомлення точно зі стисненого повідомлення, і *алгоритми з втратами,* які можуть тільки реконструювати наближення до вхідного повідомлення. Алгоритми без втрат зазвичай використовуються для тексту, а з втратами - для зображень і звуку, де невелика втрата часто не виявляється або прийнятна. Втрата однак, використовується в абстрактному сенсі і НЕ означає випадкові втрачені пікселі, а замість цього означає втрату деякої величини таких складових

як частотний компонент, або втрату шуму. Наприклад, можна подумати, що стиснення тексту з втратами було б неприйнятним, тому що вони представляють пропущені або перемикання символи. Замість цього розглянемо систему, яка спроможна перефразувати фрази в більш стандартну форму або замінити слова синонімами, щоб файл можна було краще стиснути. Технічно стиснення було б з втратами, оскільки текст змінився, але його «значення» і ясність як повідомлення можуть бути повністю збережені або навіть покращені.

Чи існує алгоритм без втрат, який може стискати все повідомлення? Була, по крайній мірі, одна заявка на патент, в якій стверджується, що вона може стискати все файли (повідомлення) - Патент 5 533 051, що має назву «Методи стиснення даних». Заявка на патент стверджувала, що, якщо б стиснення застосовувалося рекурсивно, файл можна було б зменшити майже до нуля. Але трохи подумавши, можна дійти висновку, що це неможливо, навіть якщо вихідні повідомлення можуть містити будь-яку бітову послідовність. Можна визначити це простим підрахунком аргументів. Розглянемо 1000 біт повідомлення, в якості прикладу. Є 2^{1000} різних повідомлень, які можна відправити, кожне з яких потребує чіткої ідентифікації з допомогою декодера. Повинно бути ясно, що ми НЕ можемо уявити стільки різних повідомлень, відправивши 999 або менше бітів для всіх повідомлень - 999 бітів дозволять нам відправити тільки 2^{999} окремих повідомлень. Правда в тому, що якщо яке-небудь одне повідомлення скорочується з допомогою алгоритму, то якесь інше повідомлення необхідно подовжити. Ви можете перевірити це на практиці, запустивши GZIP для файлу GIF. Фактично можна піти далі і показати, що для набору вхідних повідомлень з фіксованою довжиною, якщо одне повідомлення стисло, то середня довжина стислих повідомлень по всім можливим входам завжди буде більше, ніж вихідна. Розглянемо, наприклад, 8 можливих 3-бітних повідомлень. Якщо одне стиснуто до двох бітів, два повідомлення доведеться розширити до 4 біт, що в середньому становить 3 1/8 біт.

Оскільки не можна сподіватися на стиск усього, алгоритми стиснення повинні виходити з того, що у вхідних повідомленнях є деяка похибка, так що деякі входи більш вірогідні, ніж інші, то існує деякий незбалансоване розподіл ймовірностей по можливим повідомленнями. Більшість алгоритмів стиснення обґрунтовують цей «зсув» на структуру повідомлень - припущення, що повторювані символи більш вірогідні, ніж випадкові символи, або що на «типових» зображеннях з'являються великі білі плями. Стиснення, отже, ґрунтується на ймовірності.

При обговоренні алгоритмів стиснення важливо розрізняти два компоненти: модель і кодер. Компонент *модель* якимось чином фіксує розподіл ймовірностей повідомлень, знаючи або виявляючи щось по структурі вхідних даних. Компонент *кодера* потім використовує переваги імовірнісних похибок, згенерованих в моделі, для генерації кодів. Це досягається за рахунок ефективного подовження повідомлень з низькою ймовірністю і скорочення повідомлень з високою ймовірністю. Наприклад, модель може мати загальне «розуміння» людських осіб, знаючи, що деякі «особи» більш вірогідні, ніж інші (*наприклад,* чайник НЕ буде дуже ймовірним особою). Потім кодер зможе відправляти більш короткі повідомлення для об'єктів, які виглядають як особи. Це може добре працювати для стиснення викликів телеконференцій. Моделі в більшості сучасних алгоритмів стиснення, однак, не так складні і використовують більш буденні заходи, такі як повторювані шаблони в тексті. Незважаючи на це існує безліч різних способів розробки компонента моделі алгоритмів стиснення і величезного діапазону рівнів складності, компоненти кодера мають тенденцію бути досить загальними - в сучасних алгоритмах майже виключно вони засновані або на коді Хаффмана, або на арифметичних кодах. Слід зазначити, що грань між модельною і кодовою компонентами алгоритмів не завжди чітко визначена.

Виявляється, що теорія інформіації є сполучною ланкою між компонентами моделі і кодера. Зокрема, вона дає дуже

хорошу теорію про те, як ймовірності пов'язані з інформаційним змістом і довжиною коду. Ця теорія майже ідеально відповідає практиці, і ми можемо досягти довжини коду, практично ідентичною того, що пророкує теорія.

Інше питання по алгоритмах стиснення полягає в тому, як один оцінює якість порівняно з іншим. У разі стиснення без втрат є кілька критеріїв: час стиснення, час відновлення, розмір стислих повідомлень. В разі з втратами стиснення рішення додатково ускладняються, оскільки ми повинні турбуватися про те, як добре виконується наближення з втратами є. Як правило, існує компроміс між ступенем стиснення, часом виконання і якістю реконструкції. В залежності від додатка, одне може бути більш важливим ніж інше, і можна було б вибрати відповідний алгоритм. Найкраща спроба систематичного порівняння алгоритмів стиснення без втрат - це тест порівняння архівів Джеффа Гілхріста (Archive Comparison Test – ACT, Jeff Gilchrist). Він фіксує час і коефіцієнти стиснення для сотень алгоритмів стиснення у багатьох базах даних. Він також дає оцінку, засновану на середньозважений часі виконання і ступеня стиснення.

Алгоритми стиснення інформації широко використовуються у різних задачах обробки, зберігання та передачі даних. Основна мета процесу стиснення полягає у зменшенні ємності даних для розташування на носію інформації чи при її передачі по каналах зв'язку (відповідно і зменшенні часу обміну даними).

На рис. 3.1. представлено історично-ієрархічну структуру тільки алгоритмів стиснення без втрат інформації.

У конвеєрі обробки даних із використанням стиснення (архівації) завжди присутні два процеси – безпосередньо стиснення, при переході від початкових даних до їх зменшеної кількості, та розгортання, або інша назва – декомпресія, коли данні відновлюються із отриманого набору зменшених даних. Кількісною характеристикою цього процесу є коефіцієнт стиснення, який вказує, у скільки разів даних на виході стискача стало менше. Значення більше одиниці вказують на

зменшення даних, менше одиниці – на те що даних стало навпаки більше, рівний одиниці – відповідно ємність даних не змінилася. У останньому випадку, однак, змінюється внутрішня структура даних.

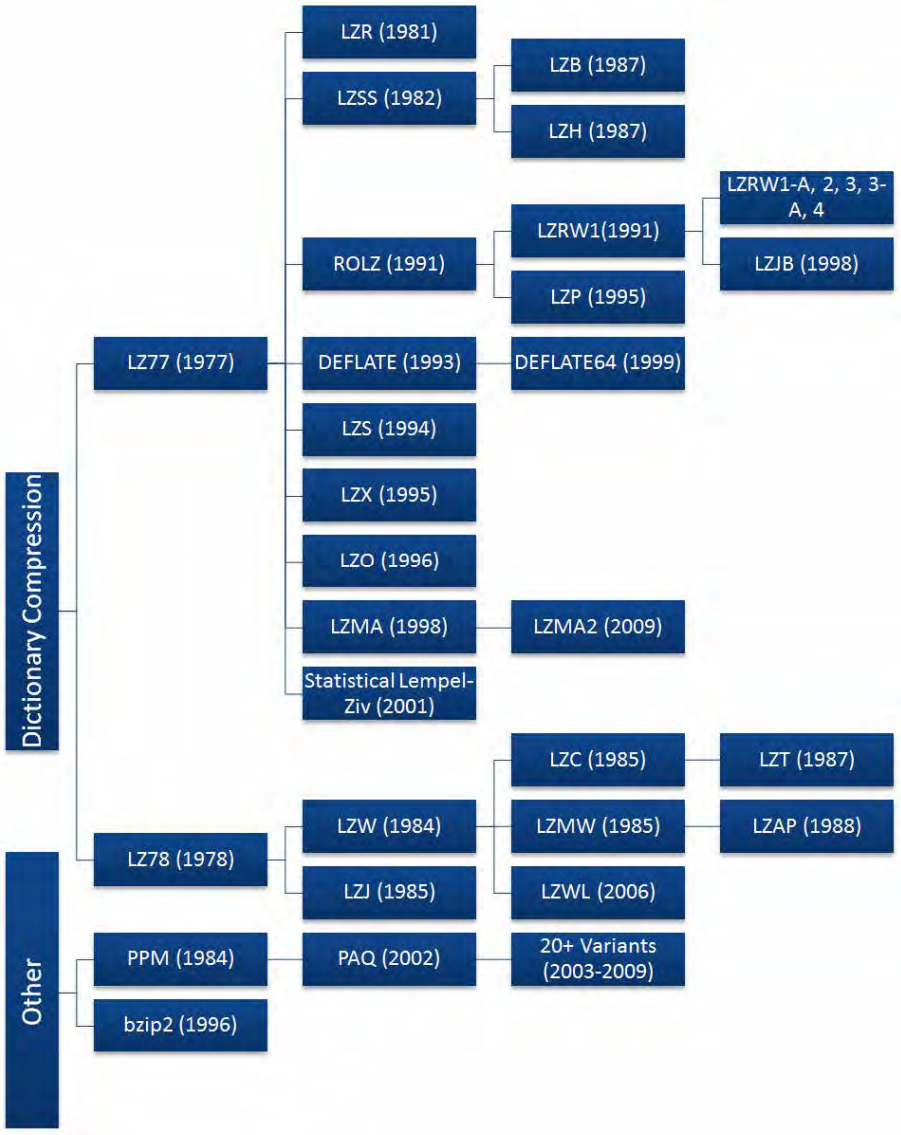


Рис. 3.1. Історично-ієрархічна структура алгоритмів стиснення без втрат

3.1.1. Історичний екскурс

Стиснення даних відігравало значну роль в обчислювальній техніці з 1970-х років, коли Інтернет ставав все більш популярним і були винайдені алгоритми Лемпеля-Зива, але у нього набагато довша історія за межами обчислювальної техніки.

Азбука Морзе, винайдена в 1838 році, є найбільш раннім прикладом стиснення даних в тому сенсі, що найбільш поширеним буквах англійською мовою, таким як «e» і «t», призначаються коротші азбуки Морзе. Пізніше, коли мейнфрейм-комп'ютери почали поширюватися в 1949 році, Клод Шеннон і Роберт Фано винайшли кодування Шеннона-Фано . Їх алгоритм привласнює коди до символів в даному блоці даних на основі ймовірності символу відбувається. Імовірність з символу відбувається обернено пропорційна довжині коду, в результаті чого більш короткий спосіб представлення даних.

Два роки по тому, Девід Хаффман вивчав теорію інформації в Массачусетському технологічному інституті і відвідував заняття з Робертом Фано . Фано дав класу можливість написати курсову роботу або скласти випускний іспит. Хаффман вибрав курсову роботу, яка повинна була знайти найбільш ефективний метод двійкового кодування. Пропрацювавши кілька місяців і нічого не придумавши, Хаффман збирався відкинути всю свою роботу і почати готуватися до останнього іспиту замість статті. Саме в цей момент у нього було прозріння, що з'ясовує дуже схожу, але більш ефективну техніку кодування Шеннона - Фано. Ключова відмінність між кодуванням Шеннона - Фано і кодуванням Хаффмана полягає в тому, що в першому дерево ймовірності будується знизу вгору, створюючи неоптимальний результат, а в другому воно будується зверху вниз.

Ранні реалізації кодування Шеннона- Фано і Хаффмана були зроблені з використанням апаратних і жорстко закодованих кодів. Тільки в 1970-х роках і появі Інтернету і онлайн-сховищ було реалізовано програмне стиснення, завдяки

якому коди Хаффмана генерувалися динамічно на основі вхідних даних. Пізніше, в 1977 році, Авраам Лемпеля і Джейкоб Зів опублікували свій революційний алгоритм LZ77, перший алгоритм, який використовує словник для стиснення даних. Зокрема, LZ77 використовував динамічний словник, який часто називають ковзним вікном. У 1978 році той же дует опублікував свій алгоритм LZ78, який також використовує словник; на відміну від LZ77, цей алгоритм аналізує вхідні дані і генерує статичний словник, а не генерує його динамічно.

Обидва алгоритми LZ77 і LZ78 швидко росли в популярності, породжуючи багато варіантів, показаних на діаграмі праворуч. Більшість цих алгоритмів припинили своє існування з моменту їх винаходу, і сьогодні їх широко використовують лише кілька, включаючи DEFLATE, LZMA і LZX. Більшість часто використовуваних алгоритмів засновані на алгоритмі LZ77. Це пов'язано не з технічним перевагою, а з тим, що алгоритми LZ78 стали обтяженими патентами після того, як Sperry запатентував похідний алгоритм LZW в 1984 році і почав пред'являти позови постачальникам програмного забезпечення, адміністраторам серверів і навіть кінцевим користувачам за використання формату GIF без ліцензії.

У той час утиліта стиснення UNIX використовувала дуже невелику модифікацію алгоритму LZW під назвою LZC і була пізніше припинено через проблеми з патентами. Інші розробники UNIX також почали відхилятися від використання алгоритму LZW на користь або з відкритим вихідним кодом. Це призвело до того, що підприємницькі кола UNIX прийняло формати gzip на основі DEFLATE і bzip2 на основі перетворення Берроуза- Уилера . У довгостроковій перспективі це стало перевагою для спільноти UNIX, оскільки формати gzip і bzip2 майже завжди досягають значно вищих коефіцієнтів стиснення, ніж формат LZW. Патентні проблеми, пов'язані з LZW, з тих пір припинилися, оскільки термін дії патенту на алгоритм LZW закінчився в 2003 році. Незважаючи на це, алгоритм LZW в значній мірі замінено і використовується тільки в стисненні GIF. З тих пір були також деякі похідні

LZW, але вони також не отримали широкого застосування, і алгоритми LZ77 залишаються домінуючими.

Ще одна юридична битва вибухнула в 1993 році по відношенню до алгоритму LZS. LZS був розроблений Stac Electronics для використання в програмному забезпеченні для стиснення дисків, такому як Stacker. Microsoft використовувала алгоритм LZS при розробці програмного забезпечення для стиснення дисків, яке було випущено з MS-DOS 6.0 і призначалося для подвоєння ємності жорсткого диска. Коли Stac Electronics виявила, що її інтелектуальна власність використовується, вона подала позов проти Microsoft . Пізніше Microsoft була визнана винною в порушенні патенту і наказала виплатити Stac Electronics збиток в розмірі 120 мільйонів доларів мінус 13,6 мільйона доларів, присуджених за зустрічним позовом, встановивши, що порушення Microsoft не було навмисним. Хоча у Stac Electronics v. Microsoft була велика думка, це не завадило розробці алгоритмів Лемпеля-Зива, як це зробив патентну суперечку LZW. Єдиний наслідок, мабуть, полягає в тому, що LZS ні розгалужений ні в які нові алгоритми.

Корпорації та інші великі організації використовували стиснення даних з тих пір, як були опубліковані алгоритми Лемпеля-Зива, оскільки вони мають постійно зростаючі потреби в зберіганні, а стиснення даних дозволяє їм задовольняти ці потреби. Однак стиснення даних не набуло широкого поширення, поки Інтернет не почав розвиватися в кінці 1980-х років, коли виникла необхідність в стисненні даних. Пропускна здатність була або обмеженою, дорогий, або і те і інше, і стиснення даних допомогло усунути ці вузькі місця. Стиснення стало особливо важливим, коли була розроблена Всесвітня павутина, коли люди стали ділитися великою кількістю зображень та інших форматів, які значно більше, ніж текст. Щоб задовольнити попит, було розроблено кілька нових форматів файлів, включаючи стиснення, включаючи ZIP, GIF і PNG.

Том Хендерсон випустив перший комерційно успішний формат архіву під назвою ARC в 1985 році через свою компанію System Enhancement Associates . ARC був особливо популярний в співтоваристві BBS, так як він був однією з перших програм, здатних як пов'язувати, так і стискати файли, а також був зроблений з відкритим вихідним кодом. Формат ARC використовує модифікацію алгоритму LZW для стиснення даних. Людина на ім'я Філ Кац помітила популярність ARC і вирішила поліпшити її, написавши процедури стиснення і розпакування на асемблері. Він випустив свою програму PKARC як умовно-безкоштовне програмне забезпечення в 1987 році, а потім Хендерсон подав на нього в суд за порушення авторських прав. Він був визнаний винним і змушений платити роялті і інші штрафи в рамках угоди про перехресне ліцензування. Він був визнаний винним, тому що PKARC був явною копією ARC; в деяких випадках навіть помилки в коментарях були ідентичні.

Філ Кац більше не міг продавати PKARC після 1988 року через угоди про перехресне ліцензування, тому в 1989 році він створив змінену версію PKARC, яка тепер називається форматом ZIP. В результаті використання LZW він був визнаний обтяженим патентом, і пізніше Кац вирішив переключитися на новий алгоритм IMPLODE. Формат був знову оновлений в 1993 році, коли Кац випустив PKZIP 2.0, в якому реалізований алгоритм DEFLATE, а також інші функції, такі як поділ томів. Ця версія формату ZIP зустрічається повсюдно сьогодні, так як майже всі файли zip відповідають формату PKZIP 2.0, незважаючи на свій великий вік.

GIF, або Graphics Interchange Format , був розроблений CompuServe в 1987 році, щоб дозволити спільне використання растрових зображень без втрати даних (хоча формат обмежений 256 кольорами на кадр), при цьому істотно зменшуючи розмір файлу, щоб дозволити передачу по модемів комутованого доступу. Однак, як і формат ZIP, GIF також заснований на алгоритмі LZW. Незважаючи на те, що патент був обтяжений, Unisys не зміг забезпечити дотримання своїх патентів в

достатній мірі, щоб зупинити поширення формату. Навіть зараз, через 20 років, GIF продовжує використовуватися, особливо завдяки його здатності до анімації.

Хоча GIF не можна було зупинити, CompuServe шукав формат, не обтяжений патентами, і в 1994 році представив формат Portable Network Graphics (PNG). Як і ZIP, стандарт PNG використовує алгоритм DEFLATE для виконання стиснення. Хоча DEFLATE був запатентований Кацем, патент так і не був введений в дію, і, таким чином, PNG та інші формати на основі DEFLATE уникають посягань на патенти. Хоча L злетіла в перші дні стиснення, завдяки сутяжницькі природі Unisys в його більш-менш відмирали на користь більш швидкий і ефективного алгоритм DEFLATE. DEFLATE в даний час є найбільш використовуваним алгоритмом стиснення даних, оскільки він трохи схожий на швейцарський армійський ніж стиснення.

Крім використання в форматах PNG і ZIP, DEFLATE також дуже часто використовується в обчислювальній техніці. Наприклад, формат файлу gzip (. Gz) використовує DEFLATE, оскільки по суті це версія ZIP з відкритим вихідним кодом. Інші застосування DEFLATE включають HTTP, SSL і інші технології, розроблені для ефективного стиснення даних по мережі.

На жаль, Філ Кац не прожив досить довго, щоб побачити, як його алгоритм DEFLATE захоплює комп'ютерний світ. Він страждав від алкоголізму протягом декількох років, і його життя почало розвалюватися в кінці 1990-х, будучи кілька разів заарештованим за водіння в нетверезому вигляді і інші порушення. Кац був знайдений мертвим у готельному номері 14 квітня 2000 року, у віці 37 років. Було встановлено, що причиною смерті була гостра панкреатичне кровотеча через його алкоголізму, викликане безліччю порожніх пляшок з лікером, виявлених біля його тіла.

3.1.2. Поточне архівне програмне забезпечення

Формат ZIP і інші формати, засновані на DEFLATE, використовувалися до середини 1990-х років, коли почали з'являтися нові та вдосконалені формати. У 1993 році Євген Рошаль випустив свій архіватор, відомий як WinRAR , який використовує власний формат RAR. В останній версії RAR використовується комбінація алгоритмів PPM і LZSS, але про більш ранніх реалізаціях відомо небагато. RAR став стандартним форматом для обміну файлами через Інтернет, особливо при поширенні піратських носіїв. Реалізація з відкритим вихідним кодом перетворення Берроуза- Уилера , названа bzip2, була представлена в 1996 році і швидко виросла в популярності на платформі UNIX в порівнянні з форматом gzip на основі DEFLATE. Ще одна програма стиснення з відкритим вихідним кодом була випущена в 1999 році в форматі 7-Zip або .7z. 7-Zip може стати першим форматом, який кине виклик домінуванню ZIP і RAR через його зазвичай високого ступеня стиснення, а також модульности і відкритості формату. Цей формат не обмежується використанням одного алгоритму стиснення, але замість цього може вибирати між алгоритмами bzip2, LZMA, LZMA2 і PPMd . Нарешті, на передньому краї архівного програмного забезпечення знаходяться формати PAQ *. Перший формат PAQ був випущений Меттом Махоуни в 2002 році під назвою PAQ1. PAQ істотно покращує алгоритм PPM, використовуючи метод, відомий як змішування контексту, який об'єднує дві або більше статистичних моделей для генерації кращого прогнозу наступного символу, ніж будь-яка з цих моделей самостійно.

3.1.3. Майбутні перспективні розробки

Майбутнє ніколи не визначено, але виходячи з поточних тенденцій, можна зробити деякі прогнози щодо того, що може статися в майбутньому стиснення даних. Алгоритми змішування контексту, такі як PAQ і його варіанти, почали залучати популярність, і вони мають тенденцію досягати

найвищих коефіцієнтів стиснення, але зазвичай вони повільні. З експонентним збільшенням апаратної швидкості відповідно до закону Мура, алгоритми змішування контексту, ймовірно, будуть процвітати, так як штрафи за швидкість долаються за допомогою швидшого обладнання через їх високий ступінь стиснення. Алгоритм, який PAQ прагнув поліпшити, називається Прогнозування шляхом часткового зіставлення (PPM), також можуть з'явитися нові варіанти. Нарешті, ланцюгової алгоритм Лемпеля-Зива Маркова (LZMA) незмінно демонструє чудовий компроміс між швидкістю і високим ступенем стиснення і, ймовірно, породить більше варіантів в майбутньому. LZMA може навіть стати «переможцем», оскільки він отримав подальший розвиток, оскільки вже був прийнятий у багатьох конкуруючих форматах стиснення, так як він був представлений у форматі 7-Zip. Іншим потенційним розвитком є використання стиснення за допомогою перерахування подстрок (CSE), яке представляє собою перспективну технологію стиснення, в якій ще не було реалізовано багато програмних реалізацій. У своїй наївній формі він працює аналогічно bzip2 і PPM, і дослідники працюють над підвищенням його ефективності.

3.1.4. Загальні поняття теорії інформації. Поняття ентропії

Шеннон запозичив визначення *ентропії* з статистичної фізики, де ентропія представляє собою випадковість чи безлад системи. Зокрема, передбачається, що система має набір можливих станів, в яких вона може перебувати, і в даний момент часу існує розподіл ймовірностей по цим станам. Ентропія визначається, як :

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)} \tag{3.1}$$

де S являє собою безліч можливих станів, і $p(s)$ є ймовірність стану $s \in S$. Це визначення вказує на те, що чим

більше парних ймовірностей, тим вище ентропія (безлад) і чим більше зміщені ймовірності, тим нижче ентропія - наприклад, якщо ми точно знаємо, в якому стані знаходиться система, то $H(S)=0$. Можна згадати, що другий закон термодинаміки в основному говорить про те, що ентропія замкнутої системи може тільки зростати.

У контексті теорії інформації Шеннон просто замінив термін «стан» на «повідомлення», тому S являє собою набір можливих повідомлень, а $p(s)$ є ймовірність повідомлення $s \in S$. Шеннон також визначив поняття *власної інформації* повідомлення як

$$i(s) = \log_2 \frac{1}{p(s)}. \tag{3.2}$$

Ця власна інформація являє собою кількість бітів інформації, що міститься в ній, і, грубо кажучи, кількість бітів, які потрібно використовувати для кодування цього повідомлення. Визначення власної інформації вказує, що повідомлення з більш високою ймовірністю будуть містити менше інформації (*наприклад,* повідомлення про те, що завтра в Одесі буде сонячно, менш інформативно, ніж повідомлення про те, що у Києві піде сніг).

Ентропія - це просто середньозважена ймовірність інформації про себе для кожного повідомлення. Отже, це середнє число бітів інформації, що міститься в повідомленні, обраному випадковим чином з розподілу ймовірностей. Великі ентропії представляють велику середню інформацію, і, можливо, нелогічно, ніж більш випадковий набір повідомлень (чим більше навіть ймовірності), тим більше інформації вони містять в середньому.

Розглянемо кілька прикладів ентропій для різних розподілів ймовірностей по п'яти повідомленнями.

$$
\begin{aligned}
p(S) &= \{0.25, 0.25, 0.25, 0.125, 0.125\} \\
H &= 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8 \\
&= 1.5 + 0.75 \\
&= 2.25
\end{aligned}
$$

$$
\begin{aligned}
p(s) &= \{0.5, 0.125, 0.125, 0.125, 0.125\} \\
H &= 0.5 \times \log_2 2 + 4 \times 0.125 \times \log_2 8 \\
&= 0.5 + 1.5 \\
&= 2
\end{aligned}
$$

$$
\begin{aligned}
p(s) &= \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\} \\
H &= 0.75 \times \log_2(\frac{4}{3}) + 4 \times 0.0625 \times \log_2 16 \\
&= 0.3 + 1 \\
&= 1.3
\end{aligned}
$$

Чим нерівномірній розподіл, тим нижче ентропія.

Чому логарифм зворотного від ймовірності правильна міра для самостійної інформації про наявність повідомлення? По-перше, для набору з $n=2^i$ повідомлень з рівною ймовірністю ймовірність кожного становить $1/n$. Ми також знаємо, що якщо все вони мають однакову довжину, то для кодування кожного повідомлення потрібно $\log_2 n$ бітів. Ну, це саме інформація про себе, так як $i(S_i) = \log_2 \frac{1}{p_i} = \log_2 n$.

Інша властивість інформації, яку нам хотілося б розуміти, полягає в тому, що інформація, яка надається двома незалежними повідомленнями, повинна бути сумою інформації, наданої кожним з них. Зокрема, якщо повідомлення A і B незалежні, ймовірність відправки одного за іншим дорівнює $p(A)p(B)$, а інформація, що міститься в них, дорівнює

$$
i(AB) = \lg \frac{1}{p(A)p(B)} = \lg \frac{1}{p(A)} + \lg \frac{1}{p(A)} = i(A) + i(B). \qquad (3.3)
$$

Логарифм - це сама «проста» функція, що відповідає цій властивості.

3.1.5. Умовна ентропія і ланцюги Маркова

Часто ймовірності подій (повідомлень) залежать від контексту, в якому вони відбуваються, і з допомогою контексту часто можна поліпшити наші ймовірності і зменшити ентропію. Контекстом можуть бути попередні символи в тексті або сусідні пікселі в зображенні.

Умовна ймовірність того чи іншого події e на основі контексту c записується в вигляді $p(e|c)$. Загальна ймовірність події e співвідноситься з співвідноситься з $p(e) = \sum_{c \in C} p(c)p(e|c)$ де C сукупність із всіх можливих контекстів. На основі умовних ймовірностей ми можемо визначити умовної самооцінки як $i(e|c) = \log_2 \frac{1}{p(e|c)}$ події e в контексті c . Це не повинно бути таким же, як безумовна самоінформація. Наприклад, повідомлення про те, що в Лос-Анджелесі піде дощ без будь-якої іншої інформації, говорить нам більше, ніж повідомлення про те, що піде дощ в контексті того, що в даний час січень.

Як і в безумовному випадку, ми можемо визначити усереднену умовну самоінформацію, і ми називаємо це умовною ентропією джерела повідомлень. Ми повинні отримати це середнє значення шляхом усереднення як по контекстами, так і по повідомленнями. Для множини повідомлень S і множини контекстів C, *умовна ентропія* є

$$H(S|C) = \sum_{c \in C} p(c) \sum_{s \in S} p(s|c) \log_2 \frac{1}{p(s|c)}. \qquad (3.4)$$

Якщо розподіл ймовірностей S не залежить від контексту C, то $H(S|C)=H(S)$, а в іншому випадку $H(S|C)<H(S)$. Іншими словами, знання контексту може тільки зменшити ентропію.

Шеннон фактично визначив ентропію з точки зору джерел інформації. *Джерело інформації* генерує нескінченну послідовність повідомлень X_k, $k \in \{-\infty, ..., \infty\}$ з фіксованого набору повідомлень S. Якщо ймовірність кожного повідомлення не залежить від попередніх повідомлень, то

система називається *незалежним і однаково розподіленим* (*independent and identically distributed - iid*) джерелом. Ентропія такого джерела називається *безумовною* ентропією або ентропією *першого порядку*. По замовчуванням використовують термін ентропія для позначення ентропії першого порядку.

Іншим джерелом повідомлень є Марківский процес, точніше, *ланцюг Маркова з дискретним часом*. Послідовність слід марківської моделі порядку k, якщо доступність кожного повідомлення (або подія) залежить тільки від k попередніх повідомлень, в зокрема $p(x_n|x_{n-1},...,x_{n-k}) = p(x_n|x_{n-1},...,x_{n-k},...)$, де x_i - це i- е повідомлення, згенероване джерелом. Значення, які можуть бути прийняті $\{ x_{n-1}, ..., x_{n-k} \}$, називаються станами системи. Ентропія марківского процесу визначається умовної ентропією, яка заснована на умовних ймовірності $p (x_n | x_{n-1}, ..., x_{n-k})$.

На рис. 3.2 показано приклад моделі Маркова з першим порядком. Ця Марківська модель являє ймовірності того, що джерело генерує чорний (b) або білий (w) піксель. Кожна дуга представляє умовну ймовірність генерації певного пікселя.

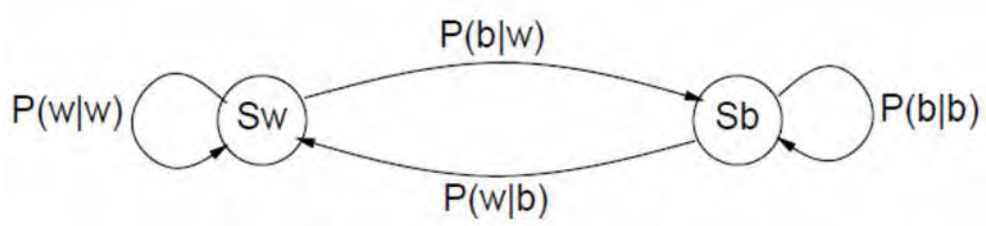


Рис. 3.2. Марківська модель першого порядку з двома станами

Для прикладу, $p(w|b)$ умовна ймовірність генерування білого пікселя, враховуючи, що попередній був чорним. Кожен вузол представляє один з станів, який у марківській моделі першого порядку згенерований попереднім повідомленням. Давайте розглянемо конкретні ймовірності $p(b|w)$=0.01, $p(w|w)$=0.99, $p(b|b)$=0.7, $p(w|b)$=0.3. Не важко з'ясувати, що для $p(b) = 1/31$ та $p(w) = 30/31$. Ці ймовірності дають умовну ентропію:

$$30/31(.01\log(1/.01) + .99\log(1/.99)) + 1/31(.7\log(1/.7) + .3\log(1/.3)) \approx .107 \quad (3.5)$$

Це дає очікуване кількість біт інформації, що міститься в кожному пікселі, створеному джерелом. Ентропія першого порядку джерела складає

$$30/31\log(31/30) + 1/31\log(1/30) \approx .206 \quad (3.6)$$

що майже в два рази більше.

Шеннон також визначив загальне поняття ентропії джерела для довільного джерела. Нехай A^n позначає безліч всіх рядків довжини n з алфавіту A, тоді *нормована ентропія n-порядку* визначається як

$$H_n = \frac{1}{n}\sum_{X\in A^n} p(X)\log\frac{1}{p(X)}, \quad (3.7)$$

Це нормалізовано, оскільки ми ділимо суму на n - це визначає ємність інформації на символ. *Ентропія джерела* тоді визначається як

$$H = \lim_{n\to\infty} H_n. \quad (3.8)$$

В загальному, надзвичайно важко визначити початкову ентропію довільного вихідного процесу, просто глянувши на результати процесу. Це пов'язано з тим, що для обчислення точних ймовірностей навіть для відносно простого процесу може знадобитися пошук надзвичайно довгих послідовностей.

3.1.6. Ймовірнісне кодування

Як згадувалося у передмові до розділу, кодування - це робота по отриманню ймовірностей для повідомлень і генерації бітових рядків на основі цих ймовірностей. Те, як генеруються ймовірність, є частиною моделі, що становить алгоритм.

На практиці ми зазвичай використовуємо ймовірності для частин більшого повідомлення, а не для повного повідомлення, *наприклад,* для кожного символу або слова в тексті. Щоб відповідати термінології в попередньому сенсі, розглянемо кожен з цих компонентів як окреме повідомлення, і ми будемо використовувати термін *послідовність повідомлень* для більш великого повідомлення, що складається з цих компонентів. В загальному, кожне маленьке повідомлення може мати різний тип і виходити з свого ймовірного розподілу. Наприклад, при відправці зображення ми можемо відправити повідомлення з зазначенням кольору, за яким слідують повідомлення з зазначенням частотної складової цього кольору. Навіть повідомлення, що визначають колір, можуть надходити з різних розподілів ймовірності, оскільки ймовірність конкретних квітів може залежати від контексту.

Ми розрізняємо алгоритми, які присвоюють унікальний код (бітову рядок) для кожного повідомлення, і алгоритми, які «змішують» коди разом більш ніж в одному повідомленні поспіль. У першому класі ми розглянемо коди Хаффмана, які є типом префіксного коду. У більш пізньої категорії ми розглянемо арифметичні коди. Арифметичні коди можуть забезпечити краще стиснення, але можуть вимагати, щоб кодувальник затримав відправку повідомлень, так як повідомлення повинні бути об'єднані перед їх відправкою.

3.1.7. Префіксне кодування

Код C для множини повідомлень S є відображення кожного повідомлення в бітовий рядок. Кожен бітовий рядок називається *кодовим словом,* і ми будемо позначати коди, використовуючи синтаксис $C=\{(s_1,w_1),(s_2,w_2),\cdots ,(s_m,w_m)\}$. Зазвичай в комп'ютерних науках ми маємо справу з кодами фіксованою довжини, такими як код ASCII, який відображає кожен друкований символ і деякі керуючі символи в 7 біт. Однак, для стиснення нам хотілось би, щоб кодові слова могли відрізнятися по довжині в залежності від ймовірності повідомлення. У таких кодів *змінної довжини* є потенційна

проблема, яка полягає в тому, що якщо ми посилаємо одне кодове слово за іншим, може бути важко або неможливо визначити, де закінчується одне кодове слово і починається наступне. Наприклад, з огляду на код {(*a*,`1`), (*b*,`01`), (*c*,`101`), (*d*,`011`)}, бітова послідовність `1011` може бути декодована як **aba**, **ca** або **ad**. Щоб уникнути цієї неоднозначності, ми могли б додати спеціальний символ зупинки в кінець кожного кодового слова, або відправити серію перед кожним символом. Ці рішення, однак, вимагають відправки додаткових даних. Більш ефективним рішенням є розробка кодів, в яких завжди можна однозначно розшифрувати бітову послідовність в її кодових словах. Такі коди прийнято називати *однозначно декодуємі* коди.

Префіксний код - це особливий вид декодуємого унікального коду, в якому жоден з бітових рядків не є префіксом іншого, наприклад {(*a*,`1`),(*b*,`01`),(*c*,`000`),(*d*,`001`)}. Усі префіксні коди однозначно декодуються, оскільки після того, як ми отримаємо збіг, більше не буде коду, який також може збігатися.

Вправа Придумайте приклад унікально декодуемого коду, який не є префіксним кодом.

Префіксні коди на самому ділі мають перевагу перед іншими однозначно декодуємими кодами в тому, що можна розшифровувати кожне повідомлення без необхідності бачити початок наступного повідомлення. Це важливо при відправці повідомлень різних типів (наприклад, з різних розподілів ймовірностей). В деяких додатках одне повідомлення може вказати тип наступного повідомлення, так що може бути необхідно повністю декодувати поточне повідомлення до того, як може бути інтерпретоване наступне. Код префіксу можна розглядати як двійкове дерево наступним чином:

- Кожне повідомлення являє собою лист в дереві
- Код для кожного повідомлення задається шляхом проходження шляху від кореня до листу і додавання 0 кожен раз, коли береться ліва гілка, і 1 кожен раз, коли береться права гілка.

Будемо називати це дерево *префіксним кодом* . Таке дерево також може бути корисно при декодуванні префіксних кодів. Коли біти надходять, декодер може слідувати по шляху вниз до дерева, доки не досягне листа, після чого він виводить повідомлення і повертається до кореню для наступного біта (або, можливо, до кореню іншого дерева для іншої тип повідомлення).

У загальному випадку префіксні коди не повинні обмежуватися двійковими алфавітами. Може бути префіксний код, в якому біти мають 3 можливих значення, і в цьому випадку відповідне дерево буде потрійним.

З огляду на розподіл ймовірностей для набору повідомлень і відповідного коду C змінної довжини, ми визначаємо *середню довжину* коду .

$$l_a(C) = \sum_{(s,w)\in C} p(s)l(w) \tag{3.9}$$

де $l(w)$ - довжина кодового слова w .

Кажуть, що префіксний код C є *оптимальним* префіксним кодом, якщо $l(w)$ зводиться до мінімуму (якщо, немає ніякого іншого префіксного коду для заданого розподілу ймовірностей, який має більш низьку середню довжину).

3.1.8. Співставлення до ентропії

Виявляється, що можна пов'язати середню довжину префіксних кодів з ентропією набору повідомлень. Для цього будемо використовувати нерівняння Крафт-Макміллана (Kraft-McMillan).

Лемма 3.1.1. Нерівність Крафт-Макміллана. Для будь-якого унікально декодуємого коду C,

$$\sum_{(s,w)\in C} 2^{-l(w)} \leq 1 \tag{3.10}$$

де l(w) - довжина кодового слова w .

Також для будь-якого набору довжин L такого, що

$$\sum_{l \in L} 2^{-l} \leq 1 \tag{3.11}$$

існує префіксний код C того ж розміру, що

$$l(w_i)=l_i (i=1,...,|L|). \tag{3.12}$$

Використовуючи це, можна доказати наступне.

Лемма 3.1.2. Для будь-якого набору повідомлень S з розподілом ймовірностей $\{p_1, p_2, \ldots, p_n\}$ і асоційованим однозначно декодіруемой кодом C,

$$H(S) \leq l_a(C) \tag{3.13}$$

Доведення. У наступних рівняннях для повідомлення $s \in S$, $l(s)$ відноситься до довжині асоційованого коду в C.

$$\begin{aligned} H(S) - l_a(C) &= \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)} - \sum_{s \in S} p(s) l(s) \\ &= \sum_{s \in S} p(s) \left(\log_2 \frac{1}{p(s)} - l(s) \right) \\ &= \sum_{s \in S} p(s) \left(\log_2 \frac{1}{p(s)} - \log_2 2^{l(s)} \right) \\ &= \sum_{s \in S} p(s) \log_2 \frac{2^{-l(s)}}{p(s)} \\ &\leq \log_2 \left(\sum_{s \in S} 2^{-l(s)} \right) \\ &\leq 0 \end{aligned} \tag{3.14}$$

З другого до останнього рядки засновані на нерівності Дженсена, яка стверджує, що якщо функція $f(x)$ є увігнутою, $\sum_i p_i f(x_i) \leq f(\sum_i p_i x_i)$. В останньому рядку використовується нерівність Крафт-Макміллана.

Ця теорема каже, що ентропія є нижньою межею середньої довжини коду. Тепер визначимо верхню межу, засновану на ентропії для оптимальних префіксних кодів.

Лемма 3.1.3. Для будь-якої множини повідомлень S з розподілом ймовірностей і пов'язаним з оптимальним кодом префікса:

$$l_a(C) \leq H(S) + 1. \tag{3.15}$$

Доведення. Кожне повідомлення $s \in S$ і яке має довжину $l(s) = \lceil \log \frac{1}{p(s)} \rceil$. Отримаємо:

$$\begin{aligned} \sum_{s \in S} 2^{-l(s)} &= \sum_{s \in S} 2^{-\lceil \log \frac{1}{p(s)} \rceil} \\ &\leq \sum_{s \in S} 2^{-\log \frac{1}{p(s)}} \\ &= \sum_{s \in S} p(s) \\ &= 1 \end{aligned} \tag{3.16}$$

Отже, відповідно нерівності Крафт-Макміллана існує префіксний код C' з кодовими словами довжини $l(s)$. В даний час:

$$\begin{aligned} l_a(C') &= \sum_{(s,w) \in C'} p(s) l(w) \\ &= \sum_{(s,w) \in C'} p(s) \lceil \log \frac{1}{p(s)} \rceil \\ &\leq \sum_{(s,w) \in C'} p(s) (1 + \log \frac{1}{p(s)}) \\ &= 1 + \sum_{(s,w) \in C'} p(s) \log \frac{1}{p(s)} \\ &= 1 + H(S) \end{aligned} \tag{3.17}$$

За визначенням оптимальних префіксних кодів $l_a(C) \leq l_a(C')$. Інша властивість оптимальних префіксних кодів полягає в тому, що великі ймовірності ніколи не можуть привести до більш довгих кодів.

Теорема 3.1.1. Якщо C є оптимальним префіксним кодом для ймовірностей $\{p_1, p_2, \ldots, p_n\}$, то з $p_i > p_j$ слідує, що $l(c_i) \leq l(c_j)$.

Доведення. Припустимо, що $l(c_i) > l(c_j)$. Тепер розглянемо код, отриманий перемиканням c_i і c_j. Якщо l_a - середня довжина нашого вихідного коду, цей новий код буде мати довжину

$$\begin{aligned} l'_a &= l_a + p_j(l(c_i) - l(c_j)) + p_i(l(c_j) - l(c_i)) \\ &= l_a + (p_j - p_i)(l(c_i) - l(c_j)) \end{aligned} \tag{3.18}$$

З огляду на наші припущення, $(p_j - p_i)(l(c_i) - l(c_j))$ є негативним, що суперечить припущенню, що l_a є оптимальним префіксним кодом.

3.1.9. Коди Хаффмана

Коди Хаффмана є оптимальними префіксними кодами, що генеруються з набору ймовірностей алгоритмом кодування Хаффмана. Девід Хаффман розробив цей алгоритм ще в 1950 році, будучи студентом з курсу по теорії інформації в Массачусетському технологічному інституті. В даний час цей алгоритм, ймовірно, є найбільш широко використовуваним компонентом алгоритмів стиснення, які використовуються в якості серверної частини GZIP, JPEG і багатьох інших утиліт.

Алгоритм Хаффмана дуже простий і його легше всього описати з точки зору того, як він генерує дерево префіксного коду.

1. Почніть з лісу дерев, по одному на кожне повідомлення. Кожне дерево містить одну вершину з вагою $w_i=p_i$
2. Повторюйте, поки не залишиться тільки одне дерево

a. Виберіть два дерева з найменшим вагою коренів (w_1 і w_2).
b. Об'єднайте їх в одне дерево, додавши новий корінь з вагою w_1+w_2 і зробивши два дерева його дочірніми. Це не має значення, яка є лівим або правим нащадком, але конвенція буде складати нижній корінь ваги на ліву, якщо $w_1 \neq w_2$

Для коду розміром n цей алгоритм зажадає n - 1 кроків, так як кожне повне бінарне дерево з n листям має n - 1 внутрішніх вузлів, і кожен крок створює один внутрішній вузол. Якщо ми використовуємо чергу пріоритетів з $O(n \log n)$ вставками часу і хвилинами пошуку (*наприклад,* купою), алгоритм буде працювати за O ($n \log n$) часу.

Ключовою властивістю кодів Хаффмана є те, що вони генерують оптимальні префіксние коди. Ми покажемо це в наступній теоремі, спочатку наведеної Хаффманом.

Лемма 3.2.1. Алгоритм Хаффмана генерує оптимальний префіксний код.

Доведення. Доказом буде індукція кількості повідомлень в коді. У статті ми покажемо, що якщо код Хаффмана генерує оптимальний префіксний код для всіх розподілів ймовірності n повідомлень, то він генерує оптимальний префіксний код для всіх розподілів n + 1 повідомлень. Базовий випадок тривіальний, оскільки префіксний код для 1 повідомлення є унікальним (*то є* порожнім повідомленням) і тому оптимальний.

Спочатку ми стверджуємо, що для будь-якого набору повідомлень S існує оптимальний код, для якого два повідомлення з мінімальною ймовірністю є однорівневими (мають одного і того ж батька в своєму дереві префіксів). За лемі 3.1.1 ми знаємо, що дві мінімальні ймовірності знаходяться на самому низькому рівні дерева (будь-яке повне бінарне дерево має по крайней мере два листа на своєму нижньому рівні). Крім того, ми можемо перемикати будь-які листи на самому низькому рівні, НЕ впливаючи на середню довжину коду, оскільки всі ці коди мають однакову довжину.

Тому ми можемо просто перемкнути дві найменші ймовірності, щоб вони були братами і сестрами.

Тепер для індукції розглянемо набір ймовірностей повідомлень S розміру $n + 1$ і відповідне дерево T, побудоване за алгоритмом Хаффмана. Назвіть два вузла з найменшою вірогідністю в дереві x і y, які повинні бути братами і сестрами в T через структури алгоритму. Розглянемо дерево T_0 отримане шляхом заміни x і y їх батьком, назвемо його z з ймовірністю $p_z = p_x + p_y$ (це фактично те, що робить алгоритм Хаффмана). Припустимо, глибина z дорівнює d, тоді

$$\begin{aligned} l_a(T) &= l_a(T') + p_x(d+1) + p_y(d+1) - p_z d \\ &= l_a(T') + p_x + p_y . \end{aligned} \tag{3.19}$$

Щоб побачити, що T є оптимальним, звернемо увагу, що існує оптимальне дерево, в якому x і y є спорідненими, і що куди б ми їх не помістили, вони додадуть константу $p_x + p_y$ до середньої довжині будь-якого дерева префіксів на S з парою x і y, заміненої їх батьком z. За припущенням індукції $l_a(T^0)$ мінімізується, оскільки T^0 має розмір n і побудований по алгоритму Хаффмана, і, отже, $l_a(T)$ мінімізується і T є оптимальним.

Оскільки кодування Хаффмана є оптимальним, ми знаємо, що для будь-якого розподілу ймовірності S і пов'язаного коду Хаффмана C - $H(S) \le l_a(C) \le H(S) + 1$.

3.1.10. Об'єднання повідомлень

Навіть якщо коди Хаффмана є оптимальними по відношенню до інших префіксним кодами, префіксние коди можуть бути вельми неефективними по порівнянні з ентропією. В Зокрема, $H(S)$ може бути набагато менше 1, і тому додатковий 1 в $H(S) + 1$ може бути дуже значним.

Один з способів зменшити накладні витрати на повідомлення - групувати повідомлення. Це особливо легко, якщо все послідовності повідомлень мають однакове розподіл ймовірностей. Розглянемо розподіл шести можливих

повідомлень. Ми могли б згенерувати ймовірності для всіх 36 пар шляхом множення ймовірностей кожного повідомлення (буде не більш 21 унікальною ймовірності). Код Хаффмана тепер може бути згенерований для цього нового розподілу ймовірностей і використаний для кодування двох повідомлень одночасно. Зверніть увагу, що цей метод НЕ використовує переваги умовних ймовірностей, оскільки він безпосередньо примножує ймовірності. В цілому, шляхом угруповання k повідомлень накладні витрати на кодування Хаффмана можуть бути зменшені з 1 біта на повідомлення до $1 \ / \ k$ біт на повідомлення. Проблема з цим способом полягає в тому, що на практиці повідомлення часто НЕ з одного і того ж розподілу, і об'єднання повідомлень з різних розподілів може бути дорогим через всіх можливих комбінацій ймовірностей, які, можливо, повинні бути згенеровані.

3.1.11. Коди Хаффмана з мінімальною дисперсією

Алгоритм кодування Хаффмана має деяку гнучкість, коли знайдені дві рівні частоти. Вибір, зроблений в таких ситуаціях, змінить остаточний код, включаючи, можливо, довжину коду кожного повідомлення. Однак, оскільки всі коди Хаффмана є оптимальними, він НЕ може змінити середню довжину. Для прикладу, розглянемо в наступних повідомлень ймовірності, і коди (табл. 3.1).

Таблиця 3.1.

Співвідношення кодів та ймовірностей для кода Хафмана

умовне позначення	ймовірність	код 1	код 2
a	0.2	01	10
б	0,4	1	00
з	0.2	000	11
d	0,1	0010	010
e	0,1	0011	011

Обидва кодування виробляють в середньому 2,2 біта на символ, хоча довжини в обох кодах досить різні. З огляду на цей вибір, є чи причина вибирати один код поверх іншого?

Для деяких додатків може бути корисно зменшити дисперсію довжини коду. Дисперсія визначається як:

$$\sum_{c \in C} p(c)(l(c) - l_a(C))^2 \tag{3.20}$$

При більш низькій дисперсії може бути простіше підтримувати постійну швидкість передачі символів або зменшити розмір буферів. У наведеному вище прикладі код 1 явно має набагато більш високу дисперсію, ніж код 2. Виявляється, що проста модифікація алгоритму Хаффмана може використовуватися для генерації коду з мінімальною дисперсією. В Зокрема, при виборі двох вузлів для злиття і вибору, заснованого на вазі, завжди вибирайте вузол, який був створений самим раннім в алгоритмі. Передбачається, що кінцеві вузли створюються перед усіма внутрішніми вузлами. У наведеному вище прикладі після об'єднання **d** і **e** пара буде мати таку ж ймовірність, що **c** і **a** (.2), але згодом вона була створена, тому ми об'єднуємо **c** і **a**. Точно так же ми вибираємо `b` замість `ac` для з'єднання з `de`, так як він був створений раніше. Це дасть код 2 вище і відповідне дерево Хаффмана на рис. 3.3.

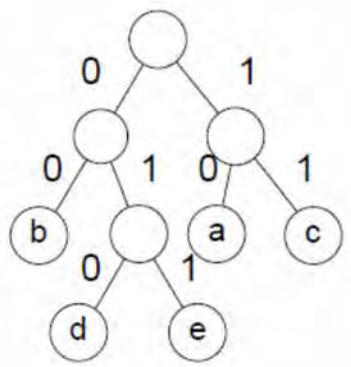


Рис. 3.3. Двійкове дерево для коду Хаффмана 2

3.1.12. Арифметичний кодування

Арифметичне кодування - це метод кодування, який дозволяє об'єднувати інформацію з повідомлень в послідовності повідомлень, щоб спільно використовувати одні і ті ж біти. Методика дозволяє загальному кількості відправлених бітів асимптотично наближатися до сумі власної інформації окремих повідомлень (нагадаємо, що власна інформація повідомлення визначається як $\log_2 \frac{1}{p_i}$.

Щоб побачити значення цього, розгляньте можливість відправки тисячі повідомлень, кожне з яких має ймовірність . 999 . Використовуючи код Хаффмана, кожне повідомлення має займати НЕ менш 1 біта, що вимагає відправки 1000 біт. З іншого боку інформацією самої кожним повідомлення є $\log_2 \frac{1}{p_i} = .00144$ біт, тому сума цієї само-інформації, більш 1000 повідомлень всього 1,4 біт. Виявляється, що арифметичне кодування буде відправляти все повідомлення, використовуючи тільки 3 біта, в сотні разів менше, ніж кодер Хаффмана. Звичайно, це крайній випадок, і коли все ймовірності малі, посилення буде менш значним. Тому арифметичні кодери найбільш корисні, коли в розподілі ймовірностей є великі ймовірності .

Основна ідея арифметичного кодування полягає в тому, щоб представляти кожну можливу послідовність з *n* повідомлень окремим інтервалом в числовий рядку між 0 і 1, *наприклад,* інтервалом від 0,2 до 0,5. Для послідовності повідомлень з вірогідністю $p_1,...,p_n$ алгоритм призначить послідовність з інтервалом розміру $\prod_{i=1}^{n} p_i$, починаючи з інтервалу розміру 1 (від 0 до 1) і звужуючи інтервал з коефіцієнтом з p_I на кожному повідомленні *i*. Ми можемо обмежити кількість бітів, необхідних для однозначної ідентифікації інтервалу розміру *s,* і використовувати це, щоб зв'язати довжину представлення з власної інформацією повідомлень.

У наступному обговоренні ми припускаємо, що декодер знає, коли послідовність повідомлень завершена, або знаючи довжину послідовності повідомлень, або включивши спеціальне повідомлення кінця файлу. Це також неявно передбачалося при відправці послідовності повідомлень з кодами Хаффмана, оскільки декодер все ще повинен знати, коли закінчена послідовність повідомлень.

Ми позначимо розподілу ймовірностей повідомлення, заданого як $\{p(1), \ldots, p(m)\}$, і визначимо *накопичену ймовірність* для розподілу ймовірностей як

$$f(j) = \sum_{i=1}^{j-1} p(i) \quad (j = 1, \ldots, m) \qquad (3.21)$$

Інтервал, заданий послідовністю повідомлень *babc* рис.3.4, дорівнює $[0.255, 0.27)$.

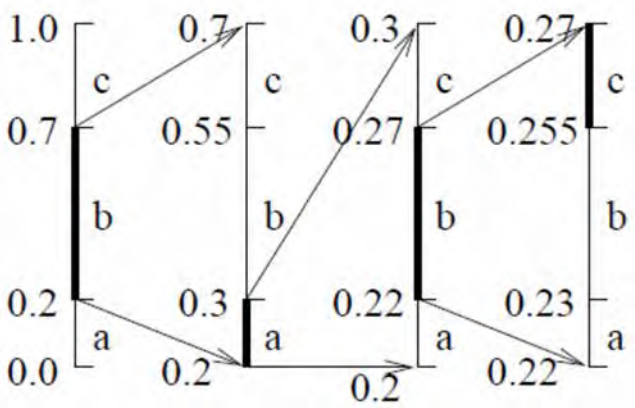


Рис. 3.4. Приклад генерації арифметичного коду, який передбачає, що всі повідомлення мають однаковий розподіл ймовірностей *а* = . 2, *б* = . 5 і *з* = . 3 .

Так, наприклад, ймовірності {0.2,0.5,0.3} з відповідністю до накопиченим ймовірностям {0,0.2,0.7} . Оскільки ми будемо часто говорити про послідовностях повідомлень, кожне з яких, можливо, має різний розподіл ймовірностей, ми будемо позначати розподіл ймовірностей *i*- ₂₀ повідомлення як $\{p_i(1), \ldots,$

$p_i(m_i)\}$, і накопичені ймовірності як $\{ f_i(1), \ldots, f_i(m_i)\}$. Для конкретної послідовності значень повідомлення ми позначаємо індекс i значення повідомлення як v_i. Ми будемо використовувати скорочення p_i для $p_i(v_i)$ і f_i для $f_i(v_i)$.

Арифметичне кодування призначає інтервал послідовності повідомлень, використовуючи наступні повторення

$$l_i = \begin{cases} f_i & i = 1 \\ l_{i-1} + f_i * s_{i-1} & 1 < i \leq n \end{cases}$$
$$s_i = \begin{cases} p_i & i = 1 \\ s_{i-1} * p_i & 1 < i \leq n \end{cases} \quad (3.22)$$

де l_n - нижня межа інтервалу, а s_n - розмір інтервалу, *тобто* інтервал задається як $[l_n, l_n + s_n)$.

Ми припускаємо, що інтервал включає нижню межу, але виключає верхню межу. Повторення звужує інтервал на кожному кроці до деякої частини попереднього інтервалу. Оскільки інтервал починається в діапазоні [0,1), він завжди залишається в межах цього діапазону. Приклад генерації інтервалу для послідовностей коротких повідомлень показаний на рис.3.4. Важливим властивістю інтервалів, що генеруються рівнянням 7, є те, що всі унікальні послідовності повідомлень довжиною n будуть мати непересічні інтервали. Фактично, будь-який число в межах інтервалу однозначно визначає послідовність повідомлень. Робота по декодуванню в основному аналогічна кодування, але замість того, щоб використовувати значення повідомлення для звуження інтервалу, ми використовуємо інтервал для вибору значення повідомлення, а потім звужує його. Тому ми можемо «відправити» послідовність повідомлень, вказавши число в відповідному інтервалі.

Залишається питання, як ефективно відправити послідовність бітів, що представляють інтервал, або число в інтервалі. Дійсні числа від 0 до 1 можуть бути представлені в двійковому вигляді.

Позначення, як $.b_1\ b_2\ b_3\ \ldots$. Для прикладу $0.75=0.11$, $9/16=0.1001$ і $1/3=0.01(01)$, де (w) означає, що послідовність w повторюється нескінченно. Тому ми можемо подумати, що досить уявити кожен інтервал, вибравши число в інтервалі, який має найменшу кількість бітів в двійковій дробової нотації, і використовувати його в якості коду. Наприклад, якщо б ми мали інтервали $[0,0.33)$, $[0.33, 67)$ і $[0.67, 1)$ ми представили б їх як $0.01\ (1/4)$, $0.1\ (1/2)$, і $0.11\ (3/4)$. Неважко показати, що для інтервалу розміру s нам потрібно саме більше – $[\log_2 s]$ біт, щоб представити таке число. Проблема в тому, що ці коди не є набором префіксних кодів. Якщо прислати 1 в наведеному вище прикладі, не можна було б узнати, чекати чи ще 1 або інтерпретувати це негайно як інтервал $[0.33, 67)$.

Щоб уникнути цієї проблеми, ми інтерпретуємо кожне бінарне дробове кодове слово як сам інтервал. Зокрема, як інтервал всіх можливих завершень. Наприклад, кодове слово $.010$ буде представляти інтервал $[1/4, 3/8)$, так як найменше можливе входження $.010(0) = 1/4$ і саме велике $.010\overline{1} = 3/8 - \epsilon$

Можливе завершення є ., Оскільки тепер у нас є кілька видів інтервалів, ми будемо використовувати наступні терміни для їх розрізнення. Ми будемо називати поточний інтервал послідовності повідомлень (тобто $[l_i, l_i + s_i)$) *інтервалу послідовності,* інтервал, відповідний ймовірності i-го повідомлення ($[f_i, f_i + p_i)$) *інтервал послідовності,* а інтервал кодового слова *інтервал коду*.

Важливим властивістю кодових інтервалів є те, що існує пряма відповідність між тим, чи перекриваються інтервали і чи утворюють вони префіксні коди.

Тема 3.2. Алгоритми стиснення інформації без втрат. Алгоритм RLE.

Мета: отримати розуміння базових принципів стиснення інформації у комп'ютерних системах та навчитися визначати

ефективність алгоритмів стиснення інформації за базовим і покращеним алгоритмами RLE.

3.2.1. Теоретичні відомості

Відповідно до того, як процеси стиснення та декомпресії впливають на кінцевий результат (після декомпресії), виділяють два класи алгоритмів – без втрат даних та з втратами даних.

У першому варіанті після виконання декомпресії отримують такий же набір даних, який був до стиснення.

За другим варіантом, отриманий після декомпресії набір містить тільки деяку інформацію з початкового набору даних та інші дані, які отримують різними шляхами – апроксимацією, наближенням чи усередненням.

Відповідно до цього, другий клас алгоритмів стиснення не можна використовувати до будь-якого типу даних, а перший клас у цьому сенсі є універсальним. Але, як за правило, отримані коефіцієнти стиснення будуть різними.

Алгоритми стиснення без втрат інформації мають досить незначні значення цього коефіцієнту – від одиниць до кількох десятків. Алгоритми стиснення з втратами – до кількох сотень одиниць. Але такі значення, або збільшення ємності інформації, спостерігаються відносно різних типів даних.

Так, наприклад, застосування алгоритму JPEG до даних із фотографічним зображенням дає коефіцієнт стиснення більше 100. А застосування до того ж зображення алгоритму стиснення RLE може призвести до його збільшення, хоча цей алгоритм застосовують для зберігання зображень у форматі *.bmp. У зв'язку із цим, говорять про класи алгоритмів, придатних для використання на певних класах даних.

Надалі, розглянемо найпростіший тип з універсальних алгоритмів стиснення даних RLE.

Особливості реалізації алгоритмів типу RLE.

Назва алгоритму RLE – від скороченого Run-length encoding – кодування довгими серіями. Ідея алгоритму полягає в тім, що ланцюг з однакових байтів (наступний байт такий же

як поточний) має бути замінений на конструкцію, яка указує скільки раз і що треба повторити.

Стиснення за базовим алгоритмом може здійснюватись за двома типами із використанням структур різного виду.

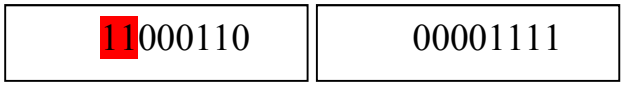
Базовий алгоритм стиснення RLE

При використанні алгоритму використовують дані, згруповані у структури двох видів.

Перша структура (рис. 3.5) складається з двох елементів – «Лічильник повторюваних даних» – «Дані для повторення». Ця конструкція формується у вихідний потік даних замість ланцюга, який складається з деякої кількості однакових елементів, , а «Лічильник для повторення» ідентифікується для відмінності від інших даних встановленням двох старших біт в структурі в «1»:

Початкові дані: 0F 0F 0F 0F 0F 0F => після стиснення - C6 0F

($128_{10} + 64_{10} + 6_{10}=178_{10} = 110000110_2 = C6_{16}$)



Байт-лічильник Байт даних для повтору

Рис. 3.5. Структура повторюваних даних

Враховуючи, що під лічильник відводиться 6 молодших біт, максимальна довжина ланцюга, який ми можемо стиснутий до 2-х байт складає $2^7 - 1 = 63$ байти. Відповідно, максимальний коефіцієнт стиснення визначається як - $K_{ст.макс} = 63/2 \approx 32$.

Друга структура складається з даних, які не є однаковими у суміжних позиціях та розподіляються по молодших 6 бітах байтів вихідного потоку, встановлюючи старші 2 біти відмінними від 11, наприклад:

у 16 розрядному поданні - AB CA 0C => 2A 3C 28 0C.
Вхідні дані - 10101011 11001010 00001100 =>

вихідні дані - 00101010 00111100 00101000 00001100

У цьому варіанті, при найгіршому наборі даних у якому усі сусідні елементи відрізняються один від одного, отримаємо $K_{ст}$ = 3/4 ≈ 0.75, що відповідає збільшенню інформації за рахунок службових бітів на третину.

Покращений алгоритм стиснення RLE

Інший варіант алгоритму стиснення, також, використовує структури двох видів але дещо інші за змістом (рис. 3.6) - «Лічильник даних для пропуску» - «Дані для пропуску» та «Лічильник повторюваних даних» - «Дані для повторення».

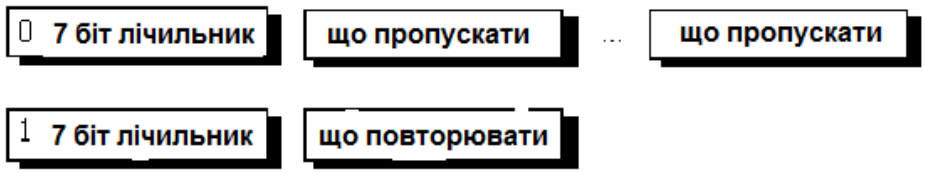


Рис. 3.6. Структури даних при стисненні RLE

Використання таких структур дозволяє вирішити кілька задач:

- спростити роботу алгоритму при обробці даних які не повторюються;
- зменшити ємність додаткової службової інформації у загальному обсягу;
- підвищити ступень стиснення.

Так, враховуючи що під лічильник даних, які повторюються, відводиться на один розряд більше ніж у попередньому варіанті, максимальне значення коефіцієнту стиснення майже вдвічі більше, ніж у попередньому варіанті і досягає значення $K_{ст.макс}$ = 127/2 ≈ 64.

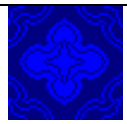
У найгіршому варіанті, $K_{ст}$=127/128≈0.992, тобто буде додано тільки 0.8% від ємності початкових даних.

3.2.2. Завдання до самостійного виконання

Виконайте стиснення та декомпресію для вказаної послідовності (табл. 3.2) яскравості частки зображення (50 байт) у відповідності до двох варіантів алгоритму RLE. Порівняйте отримані результати, визначте характеристики та особливості застосування.

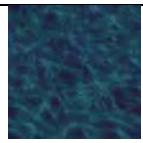
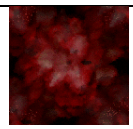
Таблиця 3.2.

Похідні дані для завдання з RLE-кодуванням

Варіант	Зображення[1]
1	Голубі кружева 16.bmp  15 15 15 15 29 29 29 15 15 15 15 15 15 15 15 29 15 15 29 15 15 15 15 15 15 15 15 15 29 15 15 29 15 15 15 15 15 15 15 15 15 29 29 29 15 15 15 15 15 15 15
2	Зелений камінь.bmp  112 105 113 124 138 145 138 137 135 140 139 138 143 138 145 142 137 129 144 137 129 125 135 143 140 144 142 138 138 138 121 80 98 98 73 42 74 31 14 76 141 154 153 137 143 135 137 125 137 142
3	Кофійня.bmp  115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 99 89 96 75 59 49 89 78 75 75 89 89 89 107 96 104 88 99 107 104 108 97 99 93 95 89 107 102 86 97 91 86

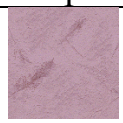
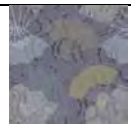
[1] З директорії, в яку встановлено Windows.

Продовження табл. 3.2.

Варіант	Зображення[2]
4	На рибалку.bmp  114 117 122 107 107 107 107 111 104 90 83 102 139 141 136 149 147 145 147 142 134 134 136 141 134 134 134 134 134 134 134 134 134 134 134 134 134 141 141 149 142 149 142 141 134 134 134 134 136 141
5	Нічний ковиль.bmp  38 40 33 36 31 31 31 31 31 31 26 31 24 31 26 31 31 31 31 36 33 41 40 54 45 53 45 53 50 59 56 59 57 53 50 53 50 52 50 52 50 53 50 53 55 59 74 73 77 65
6	Паркет.bmp  87 87 87 75 75 75 70 70 75 70 70 75 70 70 75 70 70 75 69 75 70 75 75 70 75 89 89 89 89 94 95 95 94 89 95 89 95 95 72 69 72 69 69 68 69 68 68 72 69 68
7	Бульбашки.bmp  56 51 50 43 43 41 42 35 34 35 41 43 41 43 41 43 35 41 34 34 27 27 26 26 26 26 26 26 27 27 34 41 43 35 34 32 34 27 27 26 27 26 26 26 26 26 26 26 26
8	Рододендрон.bmp  33 30 32 33 30 43 40 46 40 40 39 32 25 15 15 17 15 25 24 23 23 20 17 17 61 19 17 12 12 7 13 15 15 15 12 7 7 7 7 7 7 10 7 7 7 5 7 2 2 2

[2] З директорії, в яку встановлено Windows.

Продовження табл. 3.2.

Варіант	Зображення[3]
9	Бузковий пух.bmp  170 168 160 168 155 162 167 171 162 163 160 170 162 170 169 162 168 170 168 170 170 171 170 168 146 160 167 163 170 171 171 168 162 162 170 162 155 162 154 162 162 160 162 160 154 162 162 146 163 163
10	Штукатурка.bmp  145 143 145 136 151 150 151 143 145 140 133 136 128 128 129 145 150 151 151 143 138 145 145 151 151 151 151 150 136 135 136 128 136 140 145 136 136 128 120 128 138 136 138 145 151 151 152 151 152 143
11	Японський мотив.bmp  119 150 150 150 119 119 119 119 119 119 119 119 108 119 134 150 150 134 119 134 134 150 150 150 150 150 134 134 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 134 119 119 119

3.2.3. Запитання для самоперевірки

1. Поясніть принцип стиснення інформації за алгоритмом RLE.
2. Поясніть принцип стиснення інформації за покращеним алгоритмом RLE.
3. Чому дорівнює максимальний коефіцієнт стиснення за алгоритмом RLE? Поясніть відповідь.

[3] З директорії, в яку встановлено Windows.

4. Чому дорівнюють максимальний і мінімальний коефіцієнти стиснення за покращеним алгоритмом RLE? Поясніть відповідь.
5. Чому дорівнює максимальна довжина ланцюга, який ми можемо стиснутий до 2-х байт за базовим і покращеним за алгоритмами RLE?

Тема 3.3. Алгоритми стиснення інформації без втрат. Алгоритм LZW.

Мета: отримати розуміння базових принципів стиснення інформації у комп'ютерних системах та навчитися визначати ефективність алгоритмів стиснення інформації за алгоритмом LZW.

3.3.1. Теоретичні відомості

Розглянутий нами нижче варіант алгоритму буде використовувати дерево для подання та зберігання ланцюжків. Очевидно, що це досить сильне обмеження на вид ланцюжків, і далеко не всі однакові ланцюжки в нашому зображенні будуть використані при стисненні. Однак в пропонованому алгоритмі вигідно стискати навіть ланцюжки, що складаються з 2 байт.

Процес стиснення виглядає досить просто. Ми зчитуємо послідовно символи вхідного потоку і перевіряємо, чи є у створеній нами таблиці рядків такий рядок. Якщо рядок є, то ми зчитуємо наступний символ, а якщо рядки немає, то ми заносимо в потік код для попередньої знайденої рядки, заносимо рядок в таблицю і починаємо пошук знову.

Функція InitTable () очищає таблицю і поміщає в неї все рядки одиничної довжини.

InitTable ();CompressedFile.WriteCode (ClearCode); CurStr = порожній рядок; while (НЕ ImageFile.EOF ()) {// Ще не кінець файлу C = ImageFile.ReadNextByte (); if (CurStr + C є в таблиці) CurStr = CurStr + C; // Приклеїти символ до рядка else {code = CodeForString (CurStr); // code-ні байт! CompressedFile.WriteCode (code); AddStringToTable (CurStr +

C); CurStr = C; // Рядок з одного символу}} code = CodeForString (CurStr); CompressedFile.WriteCode (code); CompressedFile.WriteCode (CodeEndOfInformation);

Як вказувалося вище, функція InitTable () ініціалізує таблицю рядків так, щоб вона містила всі можливі рядки, що складаються з одного символу. Наприклад, якщо ми стискаємо байтові дані, то таких рядків в таблиці буде 256 (⌠0 ■, 1 ■, ..., 255 ■). Для коду очищення (ClearCode) і коду кінця інформації (CodeEndOfInformation) зарезервовані значення 256 і 257. В даному варіанті алгоритму використовується 12-бітний код, і, відповідно, під коди для рядків нам залишаються значення від 258 до 4095. Додаються рядки записуються в таблицю послідовно, при цьому індекс рядка в таблиці стає її кодом.

Функція ReadNextByte () читає символ з файлу. Функція WriteCode () записує код (не дорівнює за розміром байту) в вихідний файл. Функція AddStringToTable () додає новий рядок в таблицю, приписуючи їй код. Крім того, в даній функції відбувається обробка ситуації переповнення таблиці. В цьому випадку в потік записується код попередньої знайденої рядки і код очищення, після чого таблиця очищається функцією InitTable (). Функція CodeForString () знаходить рядок в таблиці і видає код цього рядка.

Приклад:

Нехай ми стискаємо послідовність 45, 55, 55, 151, 55, 55, 55. Тоді, відповідно до викладеного вище алгоритму, ми помістимо в вихідний потік спочатку код очищення <256>, потім додамо до спочатку порожній рядку 45 і перевіримо, чи є рядок 45 в таблиці. Оскільки ми при ініціалізації занесли в таблицю всі рядки з одного символу, то рядок 45 є в таблиці. Далі ми читаємо наступний символ 55 з вхідного потоку і перевіряємо, чи є рядок 45, 55 в таблиці. Такий рядки в таблиці поки немає. Ми заносимо в таблицю рядок 45, 55 (з першим вільним кодом 258) і записуємо в потік код <45>. Можна коротко представити архівацію так:

- ⌠45 ≈ є в таблиці;
- ⌠45, 55 ≈ немає. Додаємо в таблицю <258> ⌠45, 55 ■. У потік: <45>;
- ⌠55, 55 ≈ немає. У таблицю: <259> ⌠55, 55 ■. У потік: <55>;
- ⌠55, 151 ≈ немає. У таблицю: <260> ⌠55, 151 ■. У потік: <55>;
- ⌠151, 55 ≈ немає. У таблицю: <261> ⌠151, 55 ■. У потік: <151>;
- ⌠55, 55 ≈ є в таблиці;
- ⌠55, 55, 55 ≈ немає. У таблицю: 55, 55, 55 <262>. У потік: <259>;

Послідовність кодів для даного прикладу, що потрапляють у вихідний потік: <256>, <45>, <55>, <55>, <151>, <259>.

Особливість LZW полягає в тому, що для декомпресії нам не треба зберігати таблицю рядків в файл для розпакування. Алгоритм побудований таким чином, що ми в змозі відновити таблицю рядків, користуючись тільки потоком кодів.

Алгоритм декомпресії, що здійснює цю операцію, виглядає наступним чином:

```
code = File.ReadCode ();
while (code! = CodeEndOfInformation)
{if (code = ClearCode)
{InitTable (); code = File.ReadCode ();
if (code = CodeEndOfInformation) {закінчити роботу};
ImageFile.WriteString (StrFromTable (code));
old_code = code;}
Else
{if (InTable (code))
{ImageFile.WriteString (FromTable (code));
AddStringToTable (StrFromTable (old_code) + FirstChar (StrFromTable (code)));
old_code = code;}
Else
```

```
{OutString = StrFromTable (old_code) + FirstChar
(StrFromTable (old_code));
    ImageFile.WriteString (OutString);
    AddStringToTable (OutString);
    old_code = code;}}}
```

Тут функція ReadCode () читає черговий код з декомпрессіруемого файлу. Функція InitTable () виконує ті ж дії, що і при компресії, тобто очищає таблицю і заносить в неї все рядки з одного символу. Функція FirstChar () видає нам перший символ рядка. Функція StrFromTable () видає рядок з таблиці за кодом. Функція AddStringToTable () додає новий рядок в таблицю (привласнюючи їй перший вільний код). Функція WriteString () записує рядок у файл.

Зауваження 1. Як ви могли помітити, що записуються в потік коди поступово зростають. До тих пір, поки в таблиці не з'явиться, наприклад, в перший раз код 512, всі коди будуть менше 512. Крім того, при компресії і при декомпресії коди в таблиці додаються при обробці одного і того ж символу, тобто це відбувається синхронно ■. Ми можемо скористатися цим властивістю алгоритму для того, щоб підвищити ступінь компресії. Поки в таблицю не додано 512 символ, ми будемо писати в вихідний потік бітів коди з 9 біт, а відразу при додаванні 512 ≈ коди з 10 біт. Відповідно декомпресор також повинен буде сприймати все коди вхідного потоку 9-бітними до моменту додавання в таблицю коду 512, після чого буде сприймати все вхідні коди як 10-бітові. Аналогічно ми будемо поступати при додаванні в таблицю кодів 1024 і 2048.

Зауваження 2. При стисненні зображення нам важливо забезпечити швидкість пошуку рядків в таблиці. Ми можемо скористатися тим, що кожна наступна подстрока на один символ довше попередньою, крім того, попередній рядок вже була нами знайдена в таблиці. Отже, досить створити список посилань на рядки, що починаються з даної підрядка, як весь процес пошуку в таблиці зведеться до пошуку в рядках, що

містяться в списку для попереднього рядка. Зрозуміло, що така операція може бути проведена дуже швидко.

Зауважимо також, що реально нам досить зберігати в таблиці тільки пару <Код попередньої підрядка, доданий символ>. Цієї інформації цілком достатньо для роботи алгоритму. Таким чином, масив від 0 до 4095 з елементами <код попередньої підрядка; доданий символ; список посилань на рядки, що починаються з цього рядка> вирішує поставлене завдання пошуку, хоча і дуже повільно.

На практиці для зберігання таблиці використовується таке ж швидке, як у випадку списків, але більш компактне по пам'яті рішення ≈ хеш-таблиця. Таблиця складається з 8192 (213) елементів. Кожен елемент містить <код попередньої підрядка; доданий символ; код цього рядка>. Ключ для пошуку довжиною в 20 біт формується з використанням двох перших елементів, збережених в таблиці як одне число (key). Молодші 12 біт цього числа віддані під код, а наступні 8 біт під значення символу.

Як хеш-функції при цьому використовується:

Index (key) = ((key >> 12) ^ key) & 8191;

Де >> ≈ побітовий зрушення вправо (key >> 12 ≈ ми отримуємо значення символу), ^ ≈ логічна операція побітового виключає АБО, & логічне побітовое І.

Таким чином, за лічений кількість порівнянь ми отримуємо шуканий код або повідомлення, що такого коду в таблиці немає.

Підрахуємо кращий і гірший коефіцієнти компресії для даного алгоритму. Кращий коефіцієнт, очевидно, буде отримано для ланцюжка однакових байт великої довжини (тобто для 8-бітного зображення, всі крапки якого мають, для визначеності, колір 0). При цьому в 258 рядок таблиці ми запишемо рядок 0, 0 ■, в 259 ≈ 0, 0, 0 ■, ... в 4095 ≈ рядок з 3839 (= 4095-256) нулів. При цьому в потік потрапить (перевірте за алгоритмом!) 3840 кодів, включаючи код очищення. Отже, порахувавши суму арифметичної прогресії від 2 до 3839 (тобто довжину стислій ланцюжка) і поділивши її на 3840 * 12/8 (в

потік записуються 12-бітові коди), ми отримаємо кращий коефіцієнт компресії.

Найгірший коефіцієнт буде отримано, якщо ми жодного разу не зустрінемо подстроку, яка вже є в таблиці (в ній не повинно зустрітися жодної однакової пари символів).

У разі, якщо ми постійно будемо зустрічати новий підрядок, ми запишемо в вихідний потік 3840 кодів, яким буде відповідати рядок з 3838 символів. Без урахування зауваження 1 це складе збільшення файлу майже в 1.5 рази.

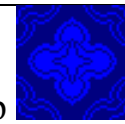
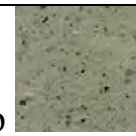
LZW реалізований в форматах GIF і TIFF.

3.3.2. Завдання до самостійного виконання

Виконайте стиснення та декомпресію для вказаної послідовності (табл. 3.3) яскравості частки зображення (50 байт) у відповідності до двох варіантів алгоритму RLE. Порівняйте отримані результати, визначте характеристики та особливості застосування.

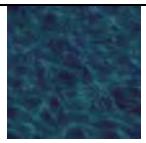
Таблиця 3.3.

Похідні дані для завдання з LZW-кодуванням

Варіант	Зображення[4]
1	Голубі кружева 16.bmp  15 15 15 15 29 29 29 15 15 15 15 15 15 15 15 29 15 15 29 15 15 15 15 15 15 15 15 15 29 15 15 29 15 15 15 15 15 15 15 15 15 29 29 29 15 15 15 15 15 15 15
2	Зелений камінь.bmp  112 105 113 124 138 145 138 137 135 140 139 138 143 138 145 142 137 129 144 137 129 125 135 143 140 144 142 138 138 138 121 80 98 98 73 42 74 31 14 76 141 154 153 137 143 135 137 125 137 142

[4] З директорії, в яку встановлено Windows.

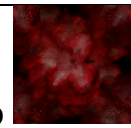
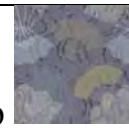
Продовження табл. 3.3.

Варіант	Зображення[5]
3	 Кофійня.bmp 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 115 99 89 96 75 59 49 89 78 75 75 89 89 89 107 96 104 88 99 107 104 108 97 99 93 95 89 107 102 86 97 91 86
4	 На рибалку.bmp 114 117 122 107 107 107 107 111 104 90 83 102 139 141 136 149 147 145 147 142 134 134 136 141 134 134 134 134 134 134 134 134 134 134 134 134 134 134 141 141 149 142 149 142 141 134 134 134 134 136 141
5	 Нічний ковиль.bmp 38 40 33 36 31 31 31 31 31 31 26 31 24 31 26 31 31 31 31 36 33 41 40 54 45 53 45 53 50 59 56 59 57 53 50 53 50 52 50 52 50 53 50 53 55 59 74 73 77 65
6	 Паркет.bmp 87 87 87 75 75 75 70 70 75 70 70 75 70 70 75 70 70 75 69 75 70 75 75 70 75 89 89 89 89 94 95 95 94 89 95 89 95 95 72 69 72 69 69 68 69 68 68 72 69 68
7	 Бульбашки.bmp 56 51 50 43 43 41 42 35 34 35 41 43 41 43 41 43 35 41 34 34 27 27 26 26 26 26 26 26 27 27 34 41 43 35 34 32 34

[5] З директорії, в яку встановлено Windows.

Варіант	Зображення[5]
	27 27 26 27 26 26 26 26 26 26 26 26 26

Продовження табл. 3.3.

Варіант	Зображення[6]
8	Рододендрон.bmp  33 30 32 33 30 43 40 46 40 40 39 32 25 15 15 17 15 25 24 23 23 20 17 17 61 19 17 12 12 7 13 15 15 15 12 7 7 7 7 7 7 10 7 7 7 5 7 2 2 2
9	Бузковий пух.bmp  170 168 160 168 155 162 167 171 162 163 160 170 162 170 169 162 168 170 168 170 170 171 170 168 146 160 167 163 170 171 171 168 162 162 170 162 155 162 154 162 162 160 162 160 154 162 162 146 163 163
10	Штукатурка.bmp  145 143 145 136 151 150 151 143 145 140 133 136 128 128 129 145 150 151 151 143 138 145 145 151 151 151 151 150 136 135 136 128 136 140 145 136 136 128 120 128 138 136 138 145 151 151 152 151 152 143
11	Японський мотив.bmp  119 150 150 150 119 119 119 119 119 119 119 119 108 119 134 150 150 134 119 134 134 150 150 150 150 150 134 134 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 134 119 119 119

[6] З директорії, в яку встановлено Windows.

3.3.3. Запитання для самоперевірки

1. Поясніть принцип стиснення інформації за алгоритмом LZW.
2. Чому дорівнює максимальний коефіцієнт стиснення за алгоритмом LZW? Поясніть відповідь.
3. Чому максимальний коефіцієнт стиснення за алгоритмом LZW має тільки теоретичне значення? Поясніть відповідь.
4. Чому дорівнюють максимальний і мінімальний коефіцієнти стиснення за алгоритмом LZW?
5. До якого класу відноситься алгоритм стиснення LZW?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Okada, Kanzo, Naoto Kojima, and Keitaro Yamashita. "A novel drive architecture of HDD:" multimode hard disc drive"." 2000 Digest of Technical Papers. International Conference on Consumer Electronics. Nineteenth in the Series (Cat. No. 00CH37102). IEEE, 2000.
2. Shiroishi, Y., et al. "Future options for HDD storage." IEEE Transactions on Magnetics 45.10 (2009): 3816-3822.
3. Park, Jeong Joo. "Hard disk equipped with a memory for storing file allocation table (FAT) information." U.S. Patent No. 6,195,217. 27 Feb. 2001.
4. Guo, Yinghua, and Jill Slay. "Data recovery function testing for digital forensic tools." IFIP International Conference on Digital Forensics. Springer, Berlin, Heidelberg, 2010.
5. Lee, Sun-Ho, Kang-Bin Yim, and Im-Yeong Lee. "A secure solution for USB flash drives using FAT file system structure." 2010 13th International Conference on Network-Based Information Systems. IEEE, 2010.
6. Yao, Qingshan, and Chunying Gu. "Research and implementation of data recovery technology based on Windows FAT." 2010 International Conference on Machine Vision and Human-machine Interface. IEEE, 2010.
7. PATEL, Sachin; GOPALAN, Yadhu N. Transaction-safe FAT file system improvements. U.S. Patent No 7,363,540, 2008.
8. Kryder, Mark H., and Chang Soo Kim. "After hard drives—What comes next?." IEEE Transactions on Magnetics 45.10 (2009): 3406-3413.
9. Eady, Fred. "Construct an ATA Hard Drive Controller." CIRCUIT CELLAR (2003): 58-67.
10. Baba, Tatsuru. "Storage device array system, information processing apparatus, storage device array control method, and program." U.S. Patent No. 8,819,522. 26 Aug. 2014.
11. Geier, Florian. "The differences between SSD and HDD technology regarding forensic investigations." (2015).

12. Tomlin, Andrew, and Sergey Anatolievich Gorobets. "Systems for managing file allocation table information." U.S. Patent No. 7,681,008. 16 Mar. 2010.
13. Wilson, Christopher, and Viresh Rustagi. "Access control using file allocation table (FAT) file systems." U.S. Patent Application No. 11/339,154.
14. Nichols, Tony, and Paul Para. "Method and system for rapid data-fragmentation analysis of a file-allocation-table (FAT) file system." U.S. Patent Application No. 11/386,375.
15. Cooke, Thomas. "File allocation table management." U.S. Patent Application No. 11/859,449.
16. Bhat, WASIM AHMAD, and S. M. K. Quadri. "Review of FAT Data Structure of FAT32 file system." Oriental Journal of Computer Science & Technology 3.1 (2010).
17. Spilo, Michael L., and Jonathan A. Daub. "Recoverable computer file system with a signature area containing file integrity information located in the storage blocks." U.S. Patent No. 6,208,999. 27 Mar. 2001.
18. Pudipeddi, Ravisankar V., and Vishal V. Ghotge. "Extensible file system." U.S. Patent No. 9,454,542. 27 Sep. 2016.
19. Tang, Chih-Chuan, and Hung-Lin Chou. "Journaling FAT file system and accessing method thereof." U.S. Patent No. 7,970,804. 28 Jun. 2011.
20. Duncan, Roy. "Design goals and implementation of the new High Performance File System." MICROSOFT SYST. J. 4.5 (1989): 1-14.
21. SAWANT, Anand, and Vinay Srinivas. "File system for digital processing systems with limited resources." U.S. Patent Application No. 10/710,998.
22. Lee, Gui-Jung, and Jung-Gi Kim. "Device and method for data recovery in a file system." U.S. Patent No. 5,974,426. 26 Oct. 1999.
23. Weinstein, Mark R. "DOS and Macintosh preformatted computer storage media." U.S. Patent No. 5,608,905. 4 Mar. 1997.

24. Rusbarsky, Kelsey Laine, and K. City. "A forensic comparison of NTFS and FAT32 file systems." http://www. marshall. edu/forensics/files/RusbarskyKelsey_Research-Paper-Summer-2012. pdf. Fetched: July 6 (2012): 2017.
25. Wong, Thomas K., and Peter W. Madany. "File allocation tables with holes." U.S. Patent No. 5,819,298. 6 Oct. 1998.
26. Wong, Thomas K., and Peter W. Madany. "Disk fragmentation reduction using file allocation tables." U.S. Patent No. 5,890,169. 30 Mar. 1999.
27. Graham, Christoph, Tri Nguyen, and Timothy Terry. "Systems and methods for building a disk image." U.S. Patent Application No. 10/872,259.
28. Nishikado, Takashi, Megumu Kondo, and Fumiya Murata. "File system management method and file management system." U.S. Patent No. 4,987,531. 22 Jan. 1991.
29. Chen, Hsuan-Tung, Kuang-Shih Lin, and Chun Liu. "Method for backup and recovery of the long filename in computer system." U.S. Patent No. 6,374,265. 16 Apr. 2002.
30. Munegowda, Keshava, et al. "FAT file in reserved cluster with ready entry state." U.S. Patent No. 8,452,734. 28 May 2013.
31. Huebner, Ewa, Derek Bem, and Cheong Kai Wee. "Data hiding in the NTFS file system." digital investigation 3.4 (2006): 211-226.
32. Russon, Richard, and Yuval Fledel. "NTFS documentation." 2013-03-05]. ht-tp://inform, pucp. edu. pe/~ in 232 (2004).
33. Nabity, Phil, and Brett JL Landry. "Recovering deleted and wiped files: A digital forensic comparison of FAT32 and NTFS file systems using evidence eliminator." (2013).
34. Wee, Cheong Kai. "Analysis of hidden data in NTFS file system." Edith Cowan University (2006).
35. Alazab, Mamoun, Sitalakshmi Venkatraman, and Paul Watters. "Digital forensic techniques for static analysis of NTFS images." Proceedings of ICIT2009, Fourth International Conference on Information Technology, IEEE Xplore. 2009.

36. Nichols, Tony, and Paul Para. "Method and system for rapid data-fragmentation analysis of a file-allocation-table (FAT) file system." U.S. Patent Application No. 11/386,375.
37. Chen, Huamin, and Bradley Childs. "Method of managing system utilities access control." U.S. Patent No. 10,182,076. 15 Jan. 2019.
38. Hass, Jon Robert, et al. "Systems and methods for accessing system utilities." U.S. Patent No. 8,819,563. 26 Aug. 2014.
39. Solomon, David A., and Helen Custer. Inside Windows NT. Vol. 2. Redmond: Microsoft Press, 1998.
40. Butler, James, Jeffrey L. Undercoffer, and John Pinkston. "Hidden processes: the implication for intrusion detection." IEEE Systems, Man and Cybernetics SocietyInformation Assurance Workshop, 2003.. IEEE, 2003.
41. Dubal, Scott P. "Method for running diagnostic utilities in a multi-threaded operating system environment." U.S. Patent No. 6,976,193. 13 Dec. 2005.
42. Mayer, Yaron. "System and method for improving the efficiency, comfort, and/or reliability in Operating Systems, such as for example Windows." U.S. Patent Application No. 11/382,698.
43. Hart, Johnson M. Windows system programming. Pearson Education, 2010.
44. Cant, Chris. Writing Windows WDM device drivers. CRC Press, 1999.
45. Beck, Douglas Reed, et al. "Method and system for detecting infection of an operating system." U.S. Patent No. 7,627,898. 1 Dec. 2009.
46. Williams, Ross N. "An extremely fast Ziv-Lempel data compression algorithm." [1991] Proceedings. Data Compression Conference. IEEE, 1991.
47. Craft, David J. "A fast hardware data compression algorithm and some algorithmic extensions." IBM Journal of Research and Development 42.6 (1998): 733-746.

48. Burtscher, Martin. "VPC3: A fast and effective trace-compression algorithm." ACM SIGMETRICS Performance Evaluation Review. Vol. 32. No. 1. ACM, 2004.
49. Whitcher, Robert H., David N. Miller, and Eric Sean Parham. "System and method for selecting a compression algorithm according to an available bandwidth." U.S. Patent No. 6,754,221. 22 Jun. 2004.
50. Fowler, James E., and Roni Yagel. "Lossless compression of volume data." (1994).
51. Carrazza, Stefano, et al. "A compression algorithm for the combination of PDF sets." The European Physical Journal C 75.10 (2015): 474.
52. Suarjaya, I. M. A. D. "A new algorithm for data compression optimization." arXiv preprint arXiv:1209.1045 (2012).

Додаток 1

Таблиця Д1.1

Фізичні та логічні складові жорсткого диску

Елемент диска	Призначення
Сторона	Жорсткі диски мають кілька поверхонь для запису. Першій стороні першої пластини присвоєний номер 0, другій стороні - 1, першій стороні другої пластини - 2 і так далі. Кожна сторона пластини розділена на концентричні кола, названі доріжками.
Доріжка	Зона для запису даних. Доріжки нумеруються послідовно від нульової до найближчої до центру. Найдальша від центру доріжка на нульовий стороні верхньої пластини диска - доріжка з номером 0 на стороні 0. Число доріжок залежить від типу диска.
Сектор	Це найменша фізична одиниця зберігання даних на жорсткому диску. Кожна доріжка ділиться на сектори. Сектор на диску служить для розмітки простору з метою швидкого знаходження інформації. У диска на кожній доріжці однакову кількість секторів. Нумерація починається з 1 сектора.
Кластер	Мінімальна одиниця адресації при зверненні до даних, яка складається з одного або декількох суміжних секторів доріжки. Якщо для розміщення файлу суміжних кластерів не вистачає, то для нього виділяються інші несуміжні кластери. В результаті виникає фрагментованість файлу, що призводить до зниження швидкості зчитування даних. Розмір кластера кратний цілому числу секторів. Файл завжди займає ціле число кластерів.
Циліндр	Сукупність доріжок, що належить всім поверхням і знаходиться на рівній відстані від

	осі обертання, з однаковими порядковими номерами, розташовані на різних дисках вінчестера.

Структура завантажувального сектора накопичувача з файловою системою FAT

Загальна структура завантажувального сектора накопичувача

Зсув	Розмір байт	Інформація
0	446	Код програми-завантажувальника
1BE	16	Розділ 1
1CE	16	Розділ 2
1DE	16	Розділ 3
1EE	16	Розділ 4
1FE	2	Сигнатура кінця сектора (55 AA)

Структура таблиці розділів (Partition Table) (для виду розмітки MBR)

Зсув	Розмір байт	Інформація
0	1	Прапор активності розділу
1	1	Початок розділу – голівка
2	1	Початок розділу – сектор (біти – 0-5, доріжка – біти 6,7)
3	1	Початок розділу – доріжка (старші біти 8,9 зберігаються в попередньому байті)
4	1	Код типу розділу
5	1	Кінець розділу – голівка
6	1	Кінець розділу – сектор (біти – 0-5, доріжка – біти 6,7)
7	1	Кінець розділу – доріжка (старші біти 8,9 зберігаються в попередньому байті)
8	4	Зсув першого сектора розділу
0C	4	Кількість секторів розділу

Додаток 2

Таблиця Д2.1

Структура завантажувального сектора накопичувача з файловою системою NTFS

Зсув	Розмір байт	Інформація
0	3	Команда JMP на початковий завантажувач
3	8	Ідентифікатор NTFS (ASCII-символи)
0B	2	Розмір сектора (в байтах)
0D	1	Розмір кластера (в секторах)
0E	2	Нульове
10	3	Нульове
13	2	Нульове
15	1	Описувач середовища (F8 або F0)
16	2	Нульове
18	2	Кількість секторів на доріжці
1A	2	Кількість головок
1C	4	Кількість скритих секторів (секторів до початку логічного диску)
20	4	Нульове
24	4	80 00 80 00 (не використовується)
28	8	Розмір логічного диска в секторах
30	8	Номер початкового кластера MFT
38	8	Номер початкового кластера копії MFT
40	1	Розмір запису MFT (в кластерах або байтах)
41	3	Нульове
44	1	Розмір індексного блока (в кластерах)
45	3	Нульове
48	8	Серійний номер логічного диска
50	4	Нульове

54	426	Код початкового завантажувача
1FE	2	Сигнатура кінця завантажувального сектора (55 AA)

Структура завантажувального сектору розділу (PBR) для FAT 16

Зсув	Розмір байт	Інформація
0	3	Команда JUMP на програму продовження завантаження ОС
3	8	Назва фірми і ОС в якій було відформатовано диск. (необов'язково)
11	2	Кількість байт в секторі
13	1	Кількість секторів в кластері
14	2	Число резервних секторів до складу якого входить BOOT-сектор
16	1	Кількість копій FAT
17	2	Кількість елементів ROOT.
19	2	Загальна кількість секторів логічного диска, за умови, що він менше 32 Mbyte. Якщо більше, то там «0»
21	1	Байт дескриптора середовища
22	2	Кількість секторів, які займає одна копія FAT
24	2	Кількість секторів на доріжці
26	2	Кількість голівок або поверхонь
28	4	Кількість прихованих секторів
32	4	Загальна кількість секторів, якщо розділ більше 32 Mbyte
36	1	Тип пристрою (80h)
37	1	Резервний байт (заповнений «0»)
38	1	ASCII код «)»
39	4	Серійний номер диска
43	11	Мітка тому
54	8	«FAT 12 або FAT 16» (рядок)
62	2	Сигнатура «55AA»

Додаток 3

Таблиця Д3.1

Структура завантажувального сектору розділу (PBR) для FAT 32.

Зсув	**Розмір в байтах**	**Інформація**
17[7]	4	Резервна область, використовувана системою (заповнена «0»)
21	1	Байт дескриптора середовища
22	2	Резерв (рівний «0» для FAT 32)
24	2	Кількість секторів на доріжці
26	2	Кількість голівок або поверхонь
28	4	Кількість прихованих секторів
32	4	Резерв
36	2	Кількість секторів під одну копію FAT
38	6	Резерв
44	4	Початковий кластер кореневого каталогу
48	2	Записана «1»
50	2	Номер сектору з копією BOOT сектору (як правило «6»)
52	12	Резерв
64	1	Фізичний номер пристрою
65	1	Резерв
66	1	Розширена сигнатура завантажувача (код «29h»)
67	4	Серійний номер пристрою
71	11	Мітка тому
82	8	рядок «FAT32 » (без коду кінця рядка «\0»)
90	2	Сигнатура «55AA»

[7] (перші 17 (0-16) байт збігаються з завантажувальним сектором FAT 16)

Таблиця Д3.2

Структура FSInfo (зсув відносно початку сектора)

Зсув	**Розмір байт**	**Інформація**
0	4	Розширена сигнатура «41615252» підтверджує те, що сектор використовується для структури FSInfo
4	480	Резерв
484	4	Сигнатура «61417272» – структурна сигнатура FSInfo
488	4	Загальна кількість вільних секторів
492	4	Номер останнього кластера
496	2	Сигнатура «55AA»

Додаток 4.

Структура записів про об’єкт в файлових системах FAT 16 і FAT 32

Таблиця Д4.1

Структура записів про об’єкт в кореневому каталозі в файловій системі FAT 16

Зсув	**Розмір байт**	**Інформація**
0	8	Ім'я об'єкту файлової системи. Це ім'я завжди записане у верхньому регістрі і вирівняне по лівому краю, справа додані символи пропуску
8	3	Розширення імені об'єкту
11	1	Байт атрибутів об'єкту
12	10	Резерв
22	2	Час створення або останньої модифікації об'єкту (гг,хв,сс)
24	2	Дата створення або останньої модифікації об'єкту (дд,мм,рік)
26	2	Номер першого кластера розподіленого об'єкту
28	4	Розмір файлу в байтах або «0» якщо це директорія

Таблиця Д4.2

Структура записів про об’єкт в кореневому каталозі в файловій системі FAT 32 (перші 12 байт збігаються із структурою дескриптора FAT 16)

Зсув	Розмір байт	Інформація
12	8	Резерв
20	2	Старша частина номера кластера
22	2	Час створення або останньої модифікації об'єкту (гг,хв,сс)
24	2	Дата створення або останньої модифікації об'єкту (дд,мм,рік)
26	2	Молодша частина номера кластера
28	4	Розмір файлу в байтах або «0» якщо це директорія

Додаток 5

Таблиця Д5.1

Порівняльна характеристика розглянутих файлових систем

Фактор порівняння	**FAT**	**FAT32 (vFAT)**	**NTFS**
Максимальний розмір тому	2 Гбайт	2 Тб	Практично необмежений
Максимальне число файлів на томі	Приблизно 65 тисяч	Практично не обмежено	Практично не обмежено
Ім'я файлу	З підтримкою довгих імен - 255 символів, системний набір символів	З підтримкою довгих імен - 255 символів, системний набір символів	255 символів, будь-які символи будь-яких алфавітів (65 тисяч різних зображень)
Можливі атрибути файлу	Базовий набір	Базовий набір	Всі можливі
Безпека	Немає	Немає	Так (починаючи з NT5.0 вбудована можливість фізично шифрувати дані)
Стискання	Немає	Немає	Так
Стійкість до збоїв	Середня (система дуже проста і тому ламатися особливо нічому :))	Погана (засоби оптимізації за швидкістю привели до появи слабких по надійності місць)	Повна – автоматичне відновлення системи при будь-яких збоях (не рахуючи фізичні помилки запису, коли пишеться одне, а насправді записується інше)
Економічність	Мінімальна (великі розміри кластерів на великих	Покращена за рахунок зменшення розмірів кластерів	Максимальна. Дуже ефективна система зберігання даних.

Фактор порівняння	FAT	FAT32 (vFAT)	NTFS
	дисках)		
Швидкодія	Висока для малого числа файлів, але швидко зменшується з появою великої кількості файлів в каталогах. результат – для слабо заповнених дисків – максимальна, для заповнених – погана	Повністю аналогічно FAT, але на дисках великого розміру (десятки Гбайт) починаються проблеми із спільною організацією даних	Система не дуже ефективна для малих і простих розділів (до 1 Гбайт), але робота з величезними масивами даних і значними каталогами організована як не можна ефективніше і дуже сильно перевершує за швидкістю інші системи

Додаток 6

Формат полів у описі файлу

Таблиця Д6.1

Формат поля часу

Біти формату	Значення
15–11	Години (0–23)
10–5	Хвилини (0–59)
4–0	Секунди/2 (0–29)

Таблиця Д6.2

Формат поля дати

Біти формату	Значення
15–9	Рік (0–119)+1980
8–5	Місяць (1–12)
0–4	День (1–31)

ДОДАТОК 7

Значення елементів таблиці FAT

FAT16		FAT32	
Значення	Тип кластеру	Значення	Тип кластеру
0000	Вільний кластер	00000000	Вільний кластер
FFF0–FFF6	Зарезервований кластер	FFFFFFF0–FFFFFFF6	Зарезервований кластер
FFF7	Поганий кластер	FFFFFFF7	Поганий кластер
FFF8– FFFF	Останній кластер в списку	FFFFFFF8–FFFFFFFF	Останній кластер в списку
0002–FFEF	Номер наступного кластеру в списку	00000002–FFFFFFEF	Номер наступного кластеру в списку

ДОДАТОК 8

Атрибути файлів

Біти атрибута	Значення
0	Лише читання. Файл призначений тільки для читання, в нього не можна нічого записувати або стирати його.
1	Об'єкт прихованого типа. Це файл не буде з’являтися в списку файлів, створюваному командою операційної системи.
2	Системний об'єкт. Цей біт встановлений у файлах, які є складовою частиною операційної системи.
3	Об'єкт визначає мітку тому. Для цього дескриптора поле імені файлу і розширення розглядаються як одне поле довжиною 11 байт.
4	Об'єкт – це директорія. Дескриптор описує об’єкт, який є підкаталогом даного каталогу.
5	Об'єкт – це файл. Дескриптор описує файл, який є підкаталогом даного каталогу.
6	Зарезервований.
7	Зарезервований.

Додаток 8

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Кафедра комп'ютерних систем і мереж

"ЗАТВЕРДЖУЮ"

Декан факультету
інформаційних технологій
_____проф. О.Г. Глазунова
"____" _____ 2019 р.

РОЗГЛЯНУТО І СХВАЛЕНО
На засіданні кафедри комп'ютерних систем і мереж
Протокол № __ від "__" _____ 2019 р.
зав. кафедри_____Лахно В.А.

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Спеціальність 123 "Комп'ютерна інженерія"

Факультет Інформаційних технологій
Розробник: Савицька Я.А., доцент

Київ – 2019 р.

1. Опис навчальної дисципліни «Системне програмне забезпечення»

Галузь знань, напрям підготовки, спеціальність, освітньо-кваліфікаційний рівень		
Галузь знань	12 – Інформаційні технології	
Спеціальність	123 – Комп’ютерна інженерія	
Освітньо-кваліфікаційний рівень	бакалавр	
Характеристика навчальної дисципліни		
Вид	Вибіркова	
Загальна кількість годин	90	90
Кількість кредитів ECTS	3	3
Кількість змістових модулів	2	
Курсовий проект (робота) (якщо є в робочому навчальному плані)	–	
Форма контролю	залік	іспит
Показники навчальної дисципліни для денної та заочної форм навчання		
Рік підготовки	3,4 (2,3 ск)	
Семестр	6(4)	7(5)
Лекційні заняття	15	30
Практичні, семінарські заняття	–	–
Лабораторні заняття	30	15
Самостійна робота	45	45
Індивідуальні завдання	–	–
Всього	90	90

Мета та завдання навчальної дисципліни

Мета: професійно застосовувати системне програмне забезпечення комп’ютерних технологій під час навчання і навчитися створювати власне системне програмне забезпечення

з метою більш ефективного вирішення задач аналізу, оптимізації, прогнозування стану роботи комп'ютерного і мережевого устаткування.

Завдання полягає в отриманні студентами знань та навиків роботи з сучасними інформаційними технологіями, які дозволяють діагностувати і виправляти логічні помилки накопичувачів на низькому рівні.

Освоєння сучасного системного забезпечення передбачає добрі знання в сфері сучасних операційних систем та основних пристроїв персонального комп'ютера, а також мов програмування як низького, так і високого рівнів.

У результаті вивчення навчальної дисципліни студент повинен

знати:

- склад, призначення та основні технічні характеристики елементів персонального комп'ютера;
- призначення, види, структуру та функції системного програмного забезпечення;
- призначення, можливості та функції файлових систем, які можуть бути використані в практичній роботі за спеціальністю;

вміти:

- добре працювати на сучасних комп'ютерах, використовуючи знання про операційні системи;
- робити постановку задачі для вирішенні її за допомогою системного програмного забезпечення;
- правильно вибрати або створити власне системне програмне забезпечення, яке допоможе вирішити задачу;
- використовувати системне програмне забезпечення для вирішення задач по спеціальності.

1. Програма та структура навчальної дисципліни на 4-ий (6-ий) семестр

8.

Назви змістових модулів і тем	Кількість годин					
	денна форма					
	усього	у тому числі				
		л	п	лаб.	інд.	с.р.
1	2	3	4	5	6	7
Змістовий модуль 1						
Тема 1. Поняття СПЗ, характеристики основних складових.	12	1		3		8
Тема 2. Класифікація ОС та тенденції у структурній побудові.	13	3		4		6
Тема 3. Структура файлових систем типу FAT.	11	2		4		5
Тема 4. Доступ до об'єктів файлової системи.	12	2		4		6
Разом за змістовим модулем 1	48	8		15		25
Змістовий модуль 2						
Тема 5. Організація файлової системи NTFS	12	2		4		6
Тема 6. Фізична організація файлової системи.	10	1		4		5
Тема 7. Організація пам'яті реального режиму.	12	3		4		5
Тема 8. Організація управління процесами в однозадачній ОС.	8	1		3		4
Разом за змістовим модулем 2	42	7		15		20
Усього годин	90	15		30		45

2. Теми лабораторних занять

№ з/п	Назва теми	Кількість годин
1	Дослідження конфігурації технічних засобів ЕОМ	4
2	Дослідження організації розподілу жорсткого диску на логічні диски	6
3	Дослідження організації доступу до об'єктів файлової системи FAT16	4
4	Дослідження організації доступу до об'єктів файлової системи FAT32	4
5	Усунення логічних помилок в структурі файлових систем типу FAT	6
6	Розробка програмних модулів дослідження завантаження виконавчих модулів	6

4. Програма та структура навчальної дисципліни на 5-ий (7- ий) семестр

Назви змістових модулів і тем	Кількість годин					
	денна форма					
	усього	у тому числі				
		л	п	лаб.	інд.	с.р.
1	2	3	4	5	6	7
Змістовий модуль 1						
Тема 1. Організація системи переривань у реальному режимі роботи процесора.	14	5		3		6
Тема 2. Дисципліни обслуговування переривання	15	5		2		8
Тема 3. Обробник переривань, як складова системних утиліт та ОС.	14	5		3		6
Разом за змістовим	43	15		8		20

модулем 1						
Змістовий модуль 2						
Тема 4. Логічна архітектура сучасних процесорів.	17	6		3		8
Тема 5. Організація пам’яті та кеш пам’яті у сучасних ПЕОМ.	14	4		2		8
Тема 6. Організація взаємодії між процесором та пам’яттю.	16	5		2		9
Разом за змістовим модулем 2	47	15		7		25
Усього годин	90	30		15		45

5. Теми лабораторних занять

№ з/п	Назва теми	Кількість годин
1	Дослідження реалізації апаратних та програмних обробників переривань	4
2	Розробка апаратних обробників переривань згідно стандарту AMIS	3
3	Дослідження типових характеристик процесора	4
4	Дослідження розширених можливостей процесора.	4

6. Теми семінарських занять

№ з/п	Назва теми	Кількість годин
	Не передбачено робочим навчальним планом	

7. Теми практичних занять

№ з/п	Назва теми	Кількість годин
	Не передбачено робочим навчальним планом	

8. Анотації лекцій

№ з/п	Тема і зміст лекцій модулю 1 (4-ий або 6-ий семестр)
1	Поняття СПЗ, характеристики основних складових. Надається визначення поняття “Системне програмне забезпечення” (СПЗ), як програмного комплексу, що спільно використовується всіма користувачами в рамках обчислювальної системи. Згідно з цього визначення СПЗ можна розділити на: операційну систему (ОС);інтерфейс ОС з користувачем; системні програми; утиліти.
2	Класифікація ОС та тенденції у структурній побудові. Розглянуті питання та сформульовані вимоги до розробки ОС, серед яких виділено дві категорії: фундаментальні та комерційні. Розглядаються критерії класифікації ОС, такі як, кількість оброблюваних завдань у одиницю часу підтримка декількох процесорів; врахування реального часу виконання процесів; підтримка мережених особливостей та т. д. Визначені та охарактеризовані етапи розвитку ОС: монолітні ОС; багаторівневі ОС; трирівневі ОС; ОС по типу “ клієнт-сервер”; мікроядерні ОС
3	Структура файлової системи типу FAT. Розглянуті ключові поняття файлової системи даного типу. Структура логічного диску. Кореневий каталог, підкаталоги, файли. Структура каталогу. Таблиця розміщення файлів. Організація доступу до інформаційних таблиць файлової системи з рівня BIOS.
4	Доступ до об’єктів файлової системи. Типи об’єктів файлової системи. Представлення інформації про об’єкти у службових даних ОС. Дії ОС

	при створенні, відкритті та вилученні об’єктів файлової системи. Виявлення порушень у структурі файлової системи. Організація доступу у межах жорстких дисків різного обсягу.

№ п/п	Тема і зміст лекцій модулю 2(4-ий або 6-ий семестр):
1	Організація файлової системи NTFS. Розгляд фундаментальних понять файлової системи NTFS. Підтримка у межах W/2k різних типів файлових систем. Особливості динамічних дисків NTFS. Особливості у організації файлової системи NTFS. Вимоги до NTFS. Можливості NTFS.
2	Фізична організація файлової системи. Потоки, атрибути, структури атрибутів. Аналіз запису MFT. Структури системних таблиць
3	Організація пам’яті реального режиму. Розглядаються особливості організації пам’яті реального режиму. Надається поняття логічного та фізичного адресу реального режиму, структури блоку пам’яті. Організація блоків пам’яті у вигляді однонаправленого списку даних. Класи, типи блоків пам’яті. Ідентифікація блоків пам’яті. Стратегії резервування та звільнення блоків пам’яті.
4	Організація управління процесами в однозадачній ОС. Розглядаються питання структур виконавчих файлів та виконавчих модулів в середовищі однозадачної ОС. Надається поняття заголовку виконавчого файлу та його використання у процесі завантаження та запуску на виконання задачі. Розглядаються етапи завантаження COM та EXE програм. Обчислювання необхідного розміру для формування блоку оточення та програмного блоку при запуску COM та EXE- програм. Настройка регістрів процесору при передачі управління програмам. Поняття оверлейних процесів.

№ з/п	Тема і зміст лекцій модулю 1 (5-ий або 7-ий семестр)
1	Організація системи переривань у реальному режимі роботи процесора. Надається структура системи переривань реальному режимі. Надаються поняття : апаратне переривання , програмне переривання, помилка, ловушка. Розглядаються питання апаратних засобів у системі переривань. Надається логічна структура контролеру переривання. Розглядається схема обробки переривання від пристрою до процесору.
2	Дисципліни обслуговування переривання. Розглядаються найбільш типові сучасні. схеми (дисципліни) обслуговування переривання такі як: з абсолютним пріоритетом, з відносним пріоритетом, та за принципом стеку. Розглядаються питання організації синхронних та асинхронних переривань.
3	Обробник переривань, як складова системних утиліт, та ОС.. Надається поняття типів обробників переривань, питання взаємодії обробника та ОС. Розглядаються різні структури організації обробників. Розглядаються питання проблем використання системних функцій в апаратних обробниках. Надається поняття сучасних стандартів на написання обробників переривань.

№ п/п	Тема і зміст лекцій модулю 2 (5-ий або 7-ий семестр)
1	Логічна архітектура сучасних процесорів. Архітектура р процесорів розглядається на прикладі двох сімейств Intel та AMD. Надається понятті та розглядаються структури сучасних багатоядерних та багатопроцесорних комплексів. Розглядаються питання паралелізму виконання обчислень. Розглядаються можливості здобуття характеристик процесору різними засобами.
2	Організація пам’яті та Кеш пам’яті у сучасних ПЕОМ. Надається поняття підсистеми пам’яті. Пам’ять

	розглядається, з одного боку, як системний розподілений ресурс, з іншого – як фізичний пристрій. Надається поняття латентності пам'яті. Розглядаються питання організації Кеш пам'яті: алгоритми занесення та знищення даних і.
3	Організація взаємодії між процесором та пам'яттю. Розглядається алгоритм доступу до пам'яті в режимі читання (запису) з боку процесору. Розглядають алгоритми взаємодії процесор - кеш пам'ять - пам'ять. Розглядаються можливості підвищення рівня продуктивності процесора за рахунок зменшення часу звернення до пам'яті.

9. Контрольні питання, комплекти тестів для визначення рівня засвоєння знань студентами.

Питання для перевірки знань студентів:

1. Функціональні відмінності між фізичним сектором, логічним сектором і кластером.
2. Логічна структура жорсткого диска.
3. Фізична структура жорсткого диска.
4. Способи розміщення файлів на диску. Характеристика і порівняння.
5. Логічна і фізична адресація на диску.
6. Процедура фізичного або низькорівневого форматування диску.
7. Процедура логічного або високорівневого форматування диску.
8. Типи розділів на жорсткому диску і їх функціональні відмінності.
9. Фрагментація диска. Причини появи, наслідки і способи усунення.
10. Види розмітки на жорсткому диску: характеристика, переваги, недоліки (MBR і GPT).
11. Основні функції файлової системи.

12. Структура і функціональні особливості файлової системи FAT16.

13. Структура і функціональні особливості файлової системи FAT32.

14. Структурні і функціональні відмінності між файловими системами FAT16 і FAT32.

15. Порушення структури файлової системи. Види порушень та їх виправлення.

16. Пошук даних, записаних в файл, в FAT16 (файл складається з декількох кластерів).

17. Пошук даних, записаних в файл, в FAT32 (файл складається з декількох кластерів).

18. Структура файлової системи NTFS.

19. Визначення стандартних атрибутів малих і великих файлів у файловій системі NTFS.

20. Визначення стандартних атрибутів малих і великих директорій у файловій системі NTFS.

21. Дозволи в NTFS (доступ к файлам і директоріям).

22. Помилка «брудний біт». Причини виникнення і способи виправлення в різних файлових системах.

23. Представлення форматів імен файлів в файловій системі на жорсткому диску (довге і коротке ім’я).

24. Контрольна сума і алгоритми її розрахунку. Навести команди перевірки.

25. Реєстр Windows. Характеристика і функціональні особливості розділів.

11. Методи навчання

Бесіда, співбесіда, пояснення, інноваційні методи з використанням мультимедійних презентацій.

12. Форми контролю

- Опитування
- Захист лабораторної роботи, теми.
- Тестування
- Перевірка конспектів

- Реферативні повідомлення
- Модульна контрольна робота.
- Залік
- Іспит

13. Приклади тестових завдань:

1. Поставте у відповідному порядку операції, які виконуються під час зчитування файлу:

A) дізнатись чи є кластер останнім;

B) після зчитування останнього кластера зайві дані кластера відсікаються по довжині файлу;

C) знайти файл в каталозі;

D) зчитати перший кластер файлу.

2. При підготовці жорсткого диска до роботи необхідно виконати певні процедури. Вкажіть порядок операцій при роботі з новим жорстким диском, на якому встановлюється ОС сімейства Windows:

A) низькорівневе форматування;

B) створення логічних дисків;

C) високорівневе форматування;

D) створення розділів.

3. Розташуйте в правильному порядку структурні області файлової системи FAT16:

A) FAT2

B) область даних;

C) FAT1;

D) завантажувальний сектор;

E) кореневий каталог.

4. Останній елемент списку кластерів містить ознаку_____.

5. Для сумісності з старими операційними системами із довгих імен файлів формуються короткі за таким(и) правилом(ами):

A) назву файлу до крапки обрізає до шести символів, у кінець назви додає ~1;

B) із довгої назви видаляє всі символи, що не можуть входити до назви в системі FAT;

C) із імені видаляє всі крапки, розміщені на початку, у кінці та в середині назви, крім останньої.

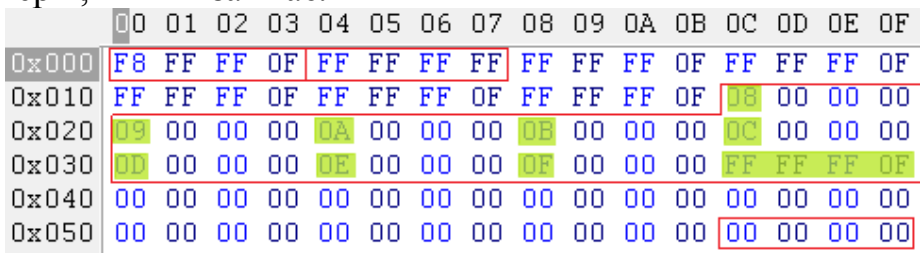
Вкажіть вірне(і) з наведених.

6. Згідно схемі розбиття MBR, інформація про кількість розділів і їх характеристики зберігається в_____

7. Файл на логічному диску FAT16 займає один кластер. Який код (в hex) буде в дескрипторі цього кластера в FAT-таблиці?

8. Файл на логічному диску FAT32 займає один кластер. Який код (в hex) буде в дескрипторі цього кластера в FAT-таблиці?

9. На малюнку наведено фрагмент FAT-таблиці логічного диску FAT32 (перші 8 байт відведені під ідентифікатор файлової системи). Проаналізуйте фрагмент і вкажіть кількість вільних кластерів. На тому ж малюнку визначте найдовший ланцюжок кластерів, що займає один об’єкт. Напишіть номера кластерів, які він займає.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	FF	FF	FF	0F
0x010	FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	08	00	00	00
0x020	09	00	00	00	0A	00	00	00	0B	00	00	00	0C	00	00	00
0x030	0D	00	00	00	0E	00	00	00	0F	00	00	00	FF	FF	FF	0F
0x040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

10. Чому дорівнює розмір запису в таблиці FAT для FAT16 і FAT32 (в байтах)?

11. Як буде записано коротке ім’я для файлу з довгим ім’ям: Internet_File_8.html?

12. Надайте у відповідність поняттям їх визначення.

A	Дзеркальний том	1	об'єкт, що представляє послідовність секторів декількох розділів яким драйвери файлової системи управляють як єдиним цілим.
B	Простий том	2	послідовність безперервних секторів на диску.
C	Складений том	3	об'єкт, що представляє послідовність безперервних секторів в межах одного розділу, яким драйвери файлової системи управляють як єдиним цілим.
D	Розділ	4	точна копія одного або декількох складених томів

13. Приведіть у відповідність значення елементів FAT для FAT 16.

A	0	1	ознака кінця ланцюжка
B	FFF0-FFF6	2	поточний кластер
C	FFF7	3	поганий кластер
D	FFF8-FFFF	4	вільний кластер
E	0002-FFEF	5	зарезервований кластер

14. Розташуйте у відповідності:

A	Ідентифікатор активного розділу	1	80
B	Сигнатура кінця завантажувального сектора	2	00
C	Ідентифікатор неактивного розділу	3	1BE
D	Зміщення, що відповідає початку PT (partition table)	4	55AA

15. Поставте у відповідність типам розміщення файлів на диску їх особливість(-ості):

A	Неперервне розміщення	1	Кожен кластер файлу містить інформацію про те, де перебуває наступний кластер цього файлу (наприклад, його номер). Заголовок файлу містить посилання на його перший кластер.
B	Зв’язні списки	2	Простота реалізації і ефективність (зчитування файлу повністю за одну операцію).
C	Зв’язні списки з таблицею розміщення файлів	3	Базовою ідеєю є перелік адрес всіх кластерів в заголовку файлу. Такий заголовок називають індексним дескриптором або inode.
D	Індексоване розміщення	4	Всі посилання, які формують списки кластерів файлу зберігаються в окремій ділянці файлової системи фіксованого розміру, формуючи FAT.

14. Розподіл балів, які отримують студенти.

Оцінювання студента відбувається згідно положення «Про екзамени та заліки у НУБіП України» від 27.02.2019р. протокол №7.

Оцінка національна	Рейтинг здобувача вищої освіти, бали
Відмінно	90-100
Добре	74-89
Задовільно	60-73
Незадовільно	0-59

15. Методичне забезпечення

1. Кропивницька, В. Б. Системне програмне забезпечення : конспект лекцій Ч1 / В. Б. Кропивницька, Т. В. Гуменюк. Івано-Франківськ : ІФНТУНГ, 2013. 190 с. [Електронний ресурс]. Режим доступу: www. http://chitalnya.nung.edu.ua/sistemne-programne-zabezpechennya.html-0
2. Кропивницька, В. Б. Системне програмне забезпечення : конспект лекцій. Ч. 2 / В. Б. Кропивницька. Івано-Франківськ : ІФНТУНГ, 2015. 112 с. [Електронний ресурс]. Режим доступу: www. http://194.44.112.13/chytalna/4761/index.html
3. Кропивницька, В. Б. Системне програмне забезпечення : лабораторний практикум / В. Б. Кропивницька, Т. В. Гуменюк. Івано-Франківськ : ІФНТУНГ, 2011. 77 с. [Електронний ресурс]. Режим доступу: http://194.44.112.13/chytalna/2578/index.html
4. Жадановская Н.П. Операционные системы. Базовый курс [Текст]: учебное пособие для студентов заочной формы обучения. – СПб.: СПбГТИ(ТУ), 2009. 120 с.

16. Рекомендована література

основна:

1. Дэвид Соломон, Марк Русинович. Внутреннее устройство Microsoft Windows 2000. СПб.: Питер, 2001. 752с.
2. Зубков С.В. Ассемблер для Dos, Windows и Unix-ДМК, 2000. 608с.
3. Вильямс Столлинг Операционные системы, М:”Вильямс”, 2002. 848с.
4. Олифер В.Г., Олифер Н.А. Сетевые операционные системы СПб.:Питер, 2001. 544с.

допоміжна:

1. Нортон П., Джорден Р. Робота з жорстким диском IBM PC: Пер. із англ. М.: Світ, 1992 р. 560 с.

2. Панков Ю. Утворення повного драйверу руcіфікації·дисплею // Комп'ютерпрес, 1993 р №.2. С.7-14.

3. Джордейн Р. Довідник програміста персональних комп'ютерів типу IBM PC, XT та AT: Пер. із англ. М.: Фінанси та статистика, 1992 р 544 с.

4. Керніган Б., Рітчі Д. Мова програмування Сі: Пер. із англ. М.: Фінанси та статистика, 1992 р 272 с.

17. Інформаційні ресурси

Курс «Системне програмне забезпечення»: http://elearn.nubip.edu.ua/course/view.php?id=1929