МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Методичні рекомендації
до виконання лабораторних робіт
з навчальної дисципліни

"ОПЕРАЦІЙНІ СИСТЕМИ"

для студентів спеціальності 7.05150102
"Технології електронних мультимедійних видань"
усіх форм навчання

Харків. ХНЕУ ім. С. Кузнеця, 2015

Затверджено на засіданні кафедри комп'ютерних систем і технологій.
Протокол № 7 від 03.02.2015 р.

Самостійне електронне текстове мережне видання

Укладач В. П. Гаврилов

М 54 Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Операційні системи" для студентів спеціальності 7.05150102 "Технології електронних мультимедійних видань" усіх форм навчання : [Електронне видання] / уклад. В. П. Гаврилов. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 129 с. (Укр. мов.)

Подано методичні рекомендації до виконання лабораторних робіт. Наведено матеріали, що охоплюють діапазон від даних, оброблюваних операційною системою, управління програмами і пристроями до створення безпечного середовища функціонування. Подано велику кількість прикладів, що допоможуть студентам зрозуміти сутність процесів, які відбуваються всередині операційної системи.

Рекомендовано для студентів спеціальності 7.05150102 "Технології електронних мультимедійних видань" усіх форм навчання.

Вступ

Побудова складних програмних систем на сучасному етапі розвитку обчислювальної техніки неможлива без знання структури й особливостей роботи сучасних операційних систем (ОС).

Основні прості функції ОС:

управління апаратними засобами;

управління оперативною пам'яттю;

забезпечення файлового введення – виведення;

завантаження додатків в оперативну пам'ять і їх виконання;

забезпечення призначеного для користувача інтерфейсу від простого командного рядка (деякі мережеві ОС) до багатофункціональних графічних (Windows, MAC OS, KDE для UNIX подібних ОС);

забезпечення мережевої взаємодії (підтримка стека мережевих протоколів);

забезпечення багатозадачності;

розподіл ресурсів комп'ютера між завданнями (процесами) і організація взаємодії завдань (процесів) один з одним;

захист системних ресурсів, даних і програм користувача, процесів, що виконуються, і самої себе від помилкових і шкідливих дій користувачів і їх програм;

розмежування прав доступу і розрахований на багато користувачів режим роботи (аутентифікація, авторизація);

організація міжмашинної взаємодії і розподілення ресурсів.

Багатозадачність і розподіл повноважень вимагають певної ієрархії привілеїв компонентів самої ОС. У складі ОС розрізняють три групи компонентів: ядро, системні бібліотеки і призначений для користувача інтерфейс.

Ядро забезпечує:

управління процесами;

обробку переривань;

операції введення – виведення;

управління пам'яттю.

Системні бібліотеки надають додаткам інтерфейс до функцій ядра на абстрактнішому рівні і забезпечують додаткову функціональність.

Призначений для користувача інтерфейс – це інтерфейс командного рядка і графічний інтерфейс, призначений для користувача (рис. 1.1).

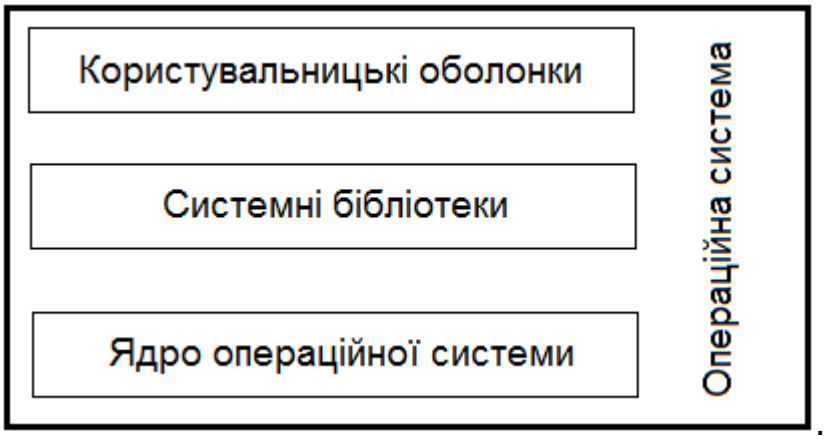


Рис. 1.1. **Структура ОС**

Більшість програм, як системних (що входять до ОС), так і прикладних, виконуються в непривілейованому режимі роботи процесора і дістають доступ до устаткування (і, у разі необхідності, до інших ядерних ресурсів, а також ресурсів інших програм) лише за допомогою системних викликів. Ядро виконується в привілейованому режимі. Аби забезпечити велику гнучкість своїх програм програміст повинен грамотно використовувати всі можливості сучасних ОС.

Даний лабораторний практикум поглибить знання студентів зі структури сучасних ОС і дозволить розробляти ефективніші програмні застосування.

Змістовий модуль 1
Функції і принципи побудови ОС

Лабораторна робота 1
Арифметичні і логічні основи ОС

Мета: вивчення способів подання і обробки інформації програмними модулями ОС.

1.1. Система числення

Система числення – це сукупність прийомів і правил, за якими числа записуються і читаються.

Існують позиційні і непозиційні системи числення.

У позиційних системах числення вага кожної цифри змінюється залежно від її положення (позиції) у послідовності цифр, що змальовують число. Наприклад, число 757.7 можна подати у вигляді:

$$700 + 50 + 7 + 0.7 = 7 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0 + 7 \cdot 10^{-1} = 757.7.$$

Будь-яка позиційна система числення характеризується своєю підставою. Підстава позиційної системи числення – це кількість різних цифр, що використовуються для зображення чисел у даній системі числення.

За підставу системи можна прийняти будь-яке натуральне число – 2, 8, 10, 16 і т. д. Запис чисел у кожній із систем числення з підставою q означає скорочений запис виразу:

$$a_{n-1}\, q^{n-1} + a_{n-2}\, q^{n-2} + \ldots + a_1\, q^1 + a_0\, q^0 + a_{-1}\, q^{-1} + \ldots + a_{-m}\, q^{-m},$$

де a_i – цифри системи числення;

n і m – число цілих і дробових розрядів, відповідно.

Окрім десяткової широко використовуються системи з підставою, що є цілою мірою числа 2, а саме:

двійкова (використовуються цифри 0, 1);

вісімкова (використовуються цифри 0, 1, .., 7);

шістнадцяткова (для перших цілих чисел від нуля до дев'яти використовуються цифри 0, 1, .., 9, а для наступних чисел – від десяти до п'ятнадцяти – як цифри використовуються символи A, B, C, D, E, F).

Переведення чисел із десяткової системи в двійкову і навпаки виконує машина. Проте, аби професійно використовувати комп'ютер, слід навчитися розуміти слово машини. Для цього і розроблені вісімкова і шістнадцяткова системи.

Переведення вісімкових і шістнадцяткових чисел у двійкову систему дуже просте: досить кожну цифру замінити еквівалентною їй двійковою тріадою (трійкою цифр) або тетрадою (четвіркою цифр). Наприклад:

$$\begin{array}{rcccccc} 537.1_8 & = & 101 & 011 & 111 & .001_2 \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \\ & & 5 & 3 & 7 & .1 & . \\ 1A3.F_{16} & = & 1 & 1010 & 0011 & 1111_2 & \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \\ & & 1 & A & 3 & F & \end{array}$$

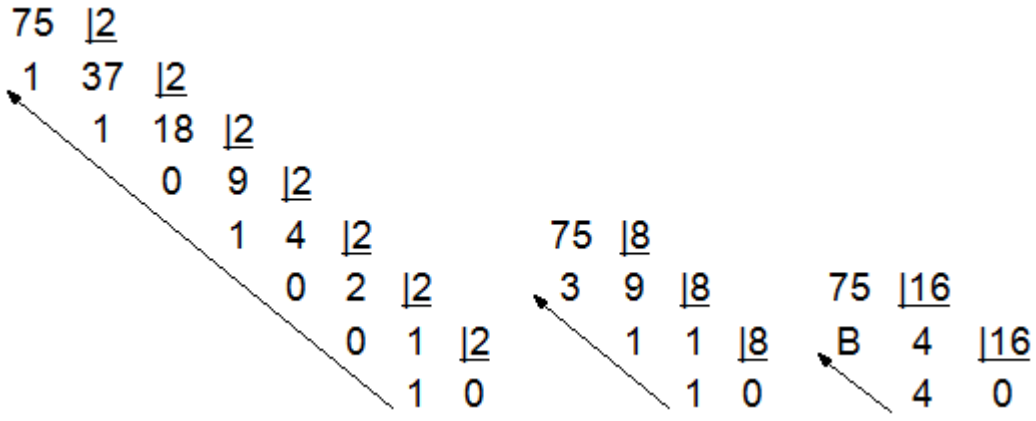
Аби перевести число з двійкової системи у вісімкову або шістнадцяткову, його потрібно розподілити вліво і вправо від коми

(крапки) на тріади (для вісімкової) або тетради (для шістнадцяткової) і кожну таку групу замінити відповідною вісімковою (шістнадцятковою) цифрою. Наприклад:

$$10101001.10111_2 = 010\ 101\ 001\ .101\ 110 = 251.56_8$$
$$\downarrow\quad\downarrow\quad\downarrow\quad\downarrow\quad\downarrow$$
$$2\quad 5\quad 1\quad 5\quad 6$$
$$10101001.10111_2 = 1010\ 1001\ .1011\ 1000 = A9.B8_{16}$$
$$\downarrow\quad\downarrow\quad\downarrow\quad\downarrow$$
$$A\quad 9\quad B\quad 8$$

Для переведення цілого десяткового числа N у систему числення з основою q необхідно N розділити з залишком на q, записане в тій же десятковій системі. Потім неповну частку, отриману від такого поділу, потрібно знову розділити з залишком на q, і так далі, поки остання отримана неповна частка не стане рівною нулю. Поданням числа N у новій системі числення буде послідовність залишків поділу, зображених однією q-ою цифрою і записаних у порядку, зворотному порядку їх отримання.

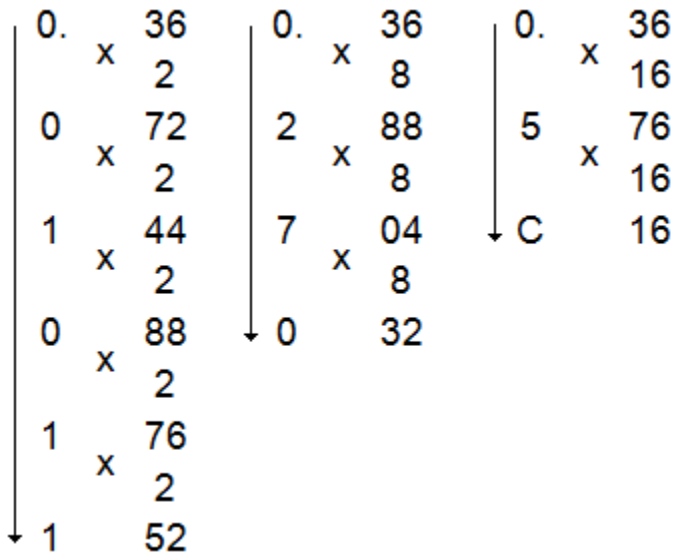
Приклад переведення числа 75 із десяткової системи в двійкову, вісімкову і шістнадцяткову:



Результат: $75_{10} = 1001011_2 = 1138 = 4B_{16}$.

Для переведення правильного десяткового дробу F у систему числення з основою q необхідно F помножити на q, записане в тій же десятковій системі, потім дробову частину отриманого добутку знову помножити на q, і так далі, до тих пір, поки дробова частина чергового добутку не стане рівною нулю, або не буде досягнута необхідна точність подання числа F у q-й системі.

Слід перевести число 0.36 з десяткової системи в двійкову, вісімкову і шістнадцяткову:



Результат: $0.36_{10} = 0.01011_2 = 0.270_8 = 0.5C_{16}$.

Для чисел, що мають як цілу, так і дробову частину, переведення з десяткової системи числення в іншу здійснюється окремо для цілої і дробової частин за правилами, які були вказані.

Переведення в десяткову систему числа x, записаного в q-й системі числення (q = 2, 8 або 16) у вигляді $x_q = (a_n a_{n-1} .. a_0, a_{-1} a_{-2} .. a\text{-}m)q$ зводиться до обчислення значення многочлена:

$$x_{10} = a_n q^n + a_{n-1} q^{n-1} + ... + a_0 q^0 + a_{-1} q^{-1} + a_{-2} q^{-2} + ... + a_{-m} q^{-m}$$

засобами десяткової арифметики. Приклади:

$$1011.1_2 = 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} = 11.5_{10};$$

$$276.5_8 = 2\cdot 8^2 + 7\cdot 8^1 + 6\cdot 8^0 + 5\cdot 8^{-1} = 190.625_{10};$$

$$1F3_{16} = 1\cdot 16^2 + F\cdot 16^1 + 3\cdot 16^0 = 499_{10}.$$

Слід розглянути основні арифметичні операції: додавання, віднімання, множення і ділення. Правила виконання цих операцій у десятковій системі добре відомі – це додавання, віднімання, множення стовпчиком і ділення кутом. Ці правила застосовуються і до всіх інших позиційних систем числення.

Приклад додавання чисел 15 і 6 у різних системах числення.

$$\begin{array}{r} 15_{10} \\ 6_{10} \\ \hline 21_{10} \end{array} \quad + \quad \begin{array}{r} 1\ 1\ 1\ 1 \\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 0\ 1 \end{array} .$$

Цілі числа можуть подаватися в комп'ютері зі знаком або без знаку. Цілі числа без знаку займають у пам'яті комп'ютера один або два байти

(табл. 1.1). У однобайтовому форматі набувають значень від 00000000_2 до 11111111_2. У двобайтовому форматі – від 00000000 00000000_2 до 11111111 11111111_2.

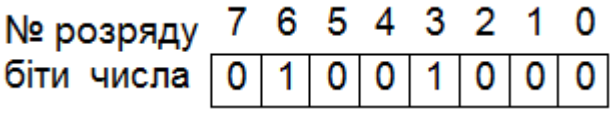
Таблиця 1.1

Діапазони значень цілих чисел без знаку

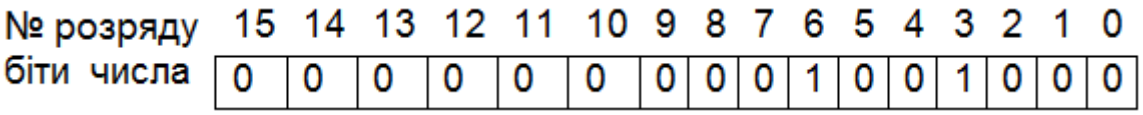
Формат числа у байтах	Діапазон	
	Запис із порядком	Звичайний запис
1	$0 .. 2^8–1$	0 .. 255
2	$0 .. 2^{16}–1$	0 .. 65535

Приклади:

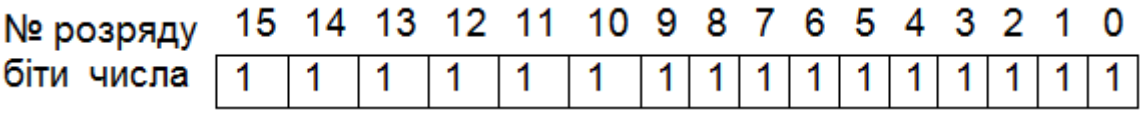
а) число 7210 = 10010002 в однобайтовому форматі:



б) це ж число в двобайтовому форматі:



в) число 6553510 у двобайтовому форматі:



Цілі числа зі знаком займають у пам'яті комп'ютера один, два або чотири байти, при цьому найлівіший (старший) розряд містить інформацію про знак числа (табл. 1.2).

Таблиця 1.2

Діапазони значень цілих чисел із знаком

Формат числа у байтах	Діапазон	
	Запис із порядком	Звичайний запис
1	$-2^7 .. 2^7–1$	-128 .. 127
2	$-2^{15} .. 2^{15}–1$	-32768 .. 32767
4	$-2^{31} .. 2^{31}–1$	-2147483648 .. 2147483647

У комп'ютерній техніці застосовуються три форми запису (кодування) цілих чисел зі знаком: прямий код, зворотний код, додатковий код.

Останні дві форми застосовуються особливо широко, оскільки дозволяють спростити конструкцію арифметико-логічного пристрою комп'ютера шляхом заміни всіляких арифметичних операцій операцією складання.

Позитивні числа в прямому, зворотному і додатковому кодах зображаються однаково – двійковими кодами з цифрою 0 у знаковому розряді. Наприклад:

$+1_{10} \rightarrow 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1_2$ $+127_{10} \rightarrow 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2$

↑ знак числа “+” ↑ знак числа “+”

Від'ємні числа в прямому, зворотному і додатковому кодах мають різне зображення.

1. Прямий код. У знаковий розряд поміщається цифра 1, а в розряди цифрової частини числа – двійковий код його абсолютної величини. Наприклад:

$-1_{10} \rightarrow 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1_2$ $-127_{10} \rightarrow 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2$

↑ знак числа “-” ↑ знак числа “-”

2. Зворотний код. Отримується інвертуванням усіх цифр двійкової коди абсолютної величини числа, включаючи розряд знака: нулі замінюються одиницями, а одиниці – нулями. Наприклад:

$+1_{10} \rightarrow 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0_2$ $+127_{10} \rightarrow 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0_2$

3. Додатковий код. Отримується утворенням зворотного коду з подальшим збільшенням одиниці до його меншого розряду. Наприклад:

$+1_{10} \rightarrow 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2$ $+127_{10} \rightarrow 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1_2$

Зазвичай від'ємні десяткові числа під час уведення в машину автоматично перетворюються в зворотний або додатковий двійковий код

і у такому вигляді зберігаються, переміщаються і беруть участь в операціях. У ході виведення таких чисел із машини буде отримано зворотне перетворення у від'ємні десяткові числа.

Завдання 1.1

1. Записати числа 19.99; 10.102; 64.58; 39.B16 у розгорнутій формі.

2. У скільки разів збільшаться числа 10.1; 10.1_2; 64.5_8; $39.B_{16}$ у процесі перенесення крапки на один знак управо?

3. Під час перенесення крапки на два знаки вправо число 11.11x збільшилося в 4 рази. Чому дорівнює x?

4. Яку мінімальну основу може мати система числення, якщо в ній записані числа 23 і 67?

5. Записати число 1999 у римській системі числення.

6. Перевести в десяткову систему такі числа: 101_2, 110_2, 111_2, 7_8, 11_8, 22_8, $1A_{16}$, BB_{16}, $9C_{16}$.

7. Перевести цілі десяткові числа 9, 17 і 243 у двійкову, вісімкову і шістнадцяткову системи числення.

8. Перевести десяткові дроби 0.2 і 0.35 у двійкову, вісімкову і шістнадцяткову системи числення з точністю до трьох знаків після коми.

9. Перевести десяткові числа 3.5 і 47.85 у двійкову, вісімкову і шістнадцяткову системи числення з точністю до трьох знаків після коми.

10. Скласти таблицю відповідності двійкових тріад і вісімкових цифр, двійкових тетрад і шістнадцяткових цифр.

11. Перевести у вісімкову і шістнадцяткову системи числення такі цілі числа: 1111_2; 1010101_2.

12. Перевести у вісімкову і шістнадцяткову системи числення такі дробові числа: $0{,}01111_2$; $0{,}10101011_2$.

13. Перевести у вісімкову і шістнадцяткову системи числення такі числа: 11.01_2; 110.101_2.

14. Перевести в двійкову систему числення такі числа: 46.27_8; $EF.12_{16}$.

15. Порівняти числа, виражені в різних системах числення: 1101_2 і D_{16}, 0.11111_2 і 0.22_8, 35.63_8 і $16.C_{16}$.

16. Провести додавання, віднімання, множення і ділення двійкових чисел 1010_2 і 10_2 і перевірити правильність виконання арифметичних дій за допомогою електронного калькулятора.

17. Додати вісімкові числа: 5_8 і 4_8, 17_8 і 41_8.

18. Провести віднімання шістнадцяткових чисел: F_{16} і $A1_6$, 41_{16} і 17_{16}.

19. Додати такі числа: 17_8 і 17_{16}, 41_8 і 41_{16}.

20. Записати від'ємні десяткові числа в прямому, зворотному і додатковому кодах у 16-розрядному поданні (старший розряд є знаковим). індивідуальні завдання наведені у табл. 1.3.

Таблиця 1.3

Індивідуальні завдання

1. Які цілі числа слідують за числами		2. Які цілі числа передують числам	
1	2	3	4
№ п/п	Завдання	№ п/п	Завдання
1	177_8	1	110_8
2	37_8	2	10_2
3	$1Б_{16}$	3	1010_2
4	101_2	4	$A10_{16}$
5	F_{16}	5	1000_2
6	77_8	6	10000_2
7	1111_2	7	20_8
8	7777_8	8	10100_2
9	101011_2	9	10_8
10	178	10	20_{16}
11	111_2	11	1100_8
12	1_2	12	11000_{16}
3. Переведіть числа в десяткову систему, перевірте результати, виконавши зворотне переведення		4. Переведіть числа з двійкової системи у вісімкову і шістнадцяткову, перевірте результати, виконавши зворотне переведення	
№ п/п	Завдання	№ п/п	Завдання
1	BE.C816	1	10011111101.0111_2
2	1011011_2	2	1110101011.1001_2
3	123.41_8	3	10111001.1011_2
4	10110111_2	4	1011110011100.11_2
5	0.10001_2	5	10111.1111101_2
6	1234_8	6	1100010101.11001_2
7	$7.\ 110100,11_2$	7	1110101011.1011_2
8	517_8	8	11001.10111_2
9	10108	9	1011101.112
10	0.348	10	1111101111.111012
11	1A16	11	111.110012
12	ABC16	12	1011101.101012

Закінчення табл. 1.3

1	2	3	4
5. Виконаєте операцію віднімання		6. Помножте числа, перевірте результати в десятковій системі	
№ п/п	Завдання	№ п/п	Завдання
1	111_2 з 10100_2	1	101101_2 і 1011_2
2	1011_2 з 1001_2	2	111101_2 і 1101_2
3	1111_2 з 10010_2	3	101111_2 і 1011_2
4	10001_2 з 111011_2	4	101_2 і 1111001_2
5	15_8 з 20_8	5	37_8 і 4_8
6	47_8 з 102_8	6	16_8 і 7_8
7	567_8 з 101_8	7	75_8 і 16_8
8	1654_8 з 3001_8	8	625_8 і 712_8
9	$1A16$ з 31_{16}	9	37_8 і 75_8
10	$F9B16$ з $2A30_{16}$	10	75_8 і 146_8
11	$B116$ з $B92_{16}$	11	75_8 і 16_8
12	$ABC16$ з 5678_{16}	12	16_8 і 712_8

1.2. Алгебра логіки

Алгебра логіки – це математичний апарат, за допомогою якого записують, обчислюють, спрощують і перетворюють логічні вирази. Творцем алгебри логіки є англійський математик Джордж Буль, який жив в XIX столітті, на честь якого ця алгебра названа булевою алгеброю виразів. Логічний вираз – це будь-яка оповідна пропозиція, відносно якої можна однозначно сказати, істинна вона або хибна. Існують три основні логічні операції: заперечення (НЕ), диз'юнкція (АБО) і кон'юнкція (І) Існують і інші логічні операції.

Операція, що виражається в'язками "якщо, то", "з слідує" "вабить", називається імплікацією (від лат. implico – тісно пов'язані) і позначається знаком →.

Вислів А → В помилковий тоді і лише тоді, коли А істинне, а В помилкове.

Операція, що виражається в'язками "тоді і лише тоді", "необхідно і досить", "рівносильно", називається еквіваленцією або подвійною імплікацією і позначається знаком або ↔. Висловлення А істинно тоді і тільки тоді, коли значення А і В збігаються.

Імплікацію можна виразити через диз'юнкцію і заперечення:

$$A \rightarrow B = \overline{A} \vee B.$$

Еквіваленцію можна виразити через заперечення, диз'юнкцію і кон'юнкцію.

$$A \leftrightarrow B = (\overline{A} \vee B) \vee (\overline{B} \vee A).$$

Порядок виконання логічних операцій задається круглими дужками. Але для зменшення кількості дужок слід вважати за замовчуванням, що спочатку виконується операція заперечення, потім кон'юнкція, після кон'юнкції – диз'юнкція і в останню чергу – імплікація.

За допомогою логічних змінних і символів логічних операцій будь-який вислів можна формалізувати, тобто замінити логічною формулою.

Математичний апарат алгебри логіки дуже зручний для опису того, як функціонують апаратні засоби комп'ютера, оскільки основною системою числення в комп'ютері є двійкова система, в якій використовуються цифри 1 і 0, а значень логічних змінних теж два: "1" і "0".

Існують різні фізичні способи кодування двійкової інформації, але найчастіше одиниця кодується вищим рівнем напруги, ніж нуль.

Існують такі набори логічних функцій, за допомогою яких можна виразити будь-які інші логічні функції. Вони називаються функціонально повними або базисами.

Найбільш відомий базис – це набір функцій І, АБО, НЕ Функція "Штрих Шеффера" є базисним, також як і функція "Стрільця Пірса". Тому за допомогою логічних елементів АБО-НЕ або І-НЕ можна зібрати будь-яку логічну схему. На таких елементах зібраний мікропроцесор комп'ютера і інші логічні пристрої. Логічні схеми складаються з логічних елементів, що здійснюють логічні операції.

1.2.1. Логічні елементи і логічні функції

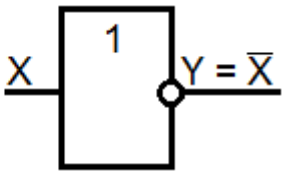
Логічна функція – це функція логічних змінних, яка може набувати лише два значення: 0 або 1. У свою чергу, власне логічна змінна (аргумент логічної функції) теж може набувати лише два значення: 0 або 1.

Логічний елемент – це пристрій, що реалізовує ту або іншу логічну функцію. $Y = f(X_1, X_2, X_3, X_n)$ – логічна функція, вона може бути задана таблицею, яка називається таблицею істинності.

Функція $Y = f(X) = НЕ(X)$ – заперечення НЕ або інверсія. Технічна реалізація цієї функції – інвертор на будь-якому транзисторі або логічному елементі, або транзисторний ключ.

Таблиця істинності функції заперечення і його позначення на схемах мають вигляд:

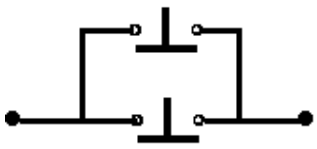
X	Y = f(X)
0	1
1	0



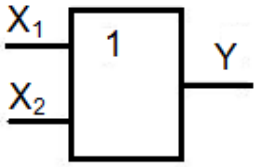
Логічне АБО (логічне складання, диз'юнкція):

$$Y = X_1 + X_2 = X_1 \vee X_2.$$

Технічна реалізація цієї функції, таблиця істинності і позначення на схемах мають вигляд:



X_1	X_2	$Y = X_1 + X_2$
0	0	0
0	1	1
1	0	1
1	1	1

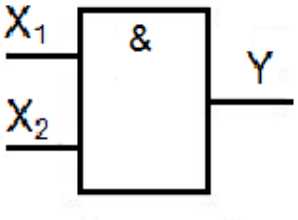


Логічне І (логічне множення, кон'юнкція, схема збігів):

$$Y = X_1 \cdot X_2 = X_1 \wedge X_2 = X_1 \,\&\, X_2.$$

Технічна реалізація цієї функції, таблиця істинності і позначення на схемах мають вигляд:

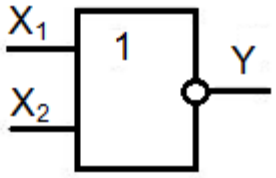
X_1	X_2	$Y = X_1 * X_2$
0	0	0
0	1	0
1	0	0
1	1	1



Функція "Стрільця Пірса" (АБО-НЕ): $Y = NOT(X_1+X_2)$.

Таблиця істинності функції АБО-НЕ і позначення на схемах мають вигляд:

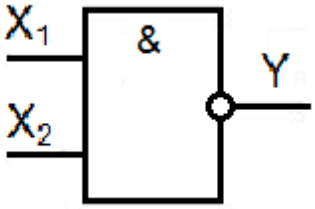
X_1	X_2	$Y = \overline{X_1 + X_2}$
0	0	1
0	1	0
1	0	0
1	1	0



Функція "Штрих Шеффера" (І-НЕ): $Y = X_1|X_2 = NOT(X_1 /\ X_2)$.

Таблиця істинності функції І-НЕ і позначення на схемах мають вигляд:

X_1	X_2	$Y = \overline{X_1 * X_2}$
0	0	1
0	1	1
1	0	1
1	1	0



Є й інші логічні функції двох змінних, що мають спеціальні назви – імплікація, еквівалентність, нерівнозначність, елемент пам'яті – тригер.

1.2.2. Перетворення логічних виразів

Рівнозначні перетворення логічних формул мають те ж призначення, що і перетворення формул у звичайній алгебрі. Вони слугують для спрощення формул або приведення їх до певного вигляду шляхом використання основних законів алгебри логіки.

Під спрощенням формули, що не містить операцій імплікації і еквіваленції, розуміють рівнозначне перетворення, що посилається на формулу, яка або містить порівняно з початковою меншу кількість операцій кон'юнкції і диз'юнкції і не містить заперечень неелементарних формул, або містить меншу кількість входжень змінних.

Деякі перетворення логічних формул схожі на перетворення формул у звичайній алгебрі (винесення загального множника за дужки, використання переміщувального і сполучного законів і тому подібне), тоді як інші перетворення ґрунтуються на властивостях, якими не

володіють операції звичайної алгебри (використання розподільного закону для кон'юнкції, законів поглинання, склеювання, де Моргана й ін.).

Слід показати на прикладах деякі прийоми і способи, що вживаються під час спрощення логічних формул:

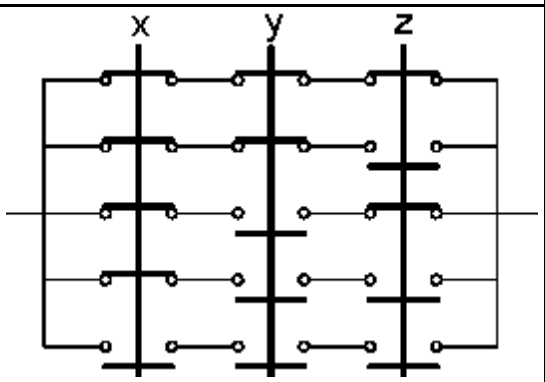
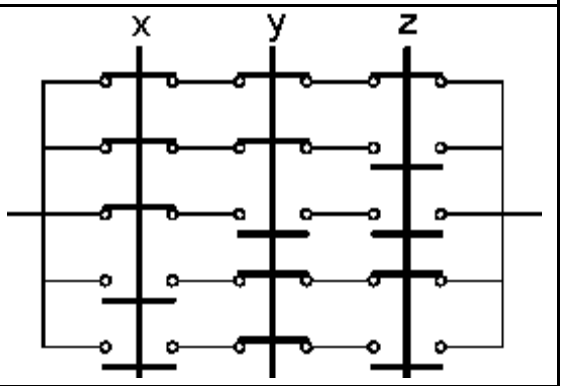
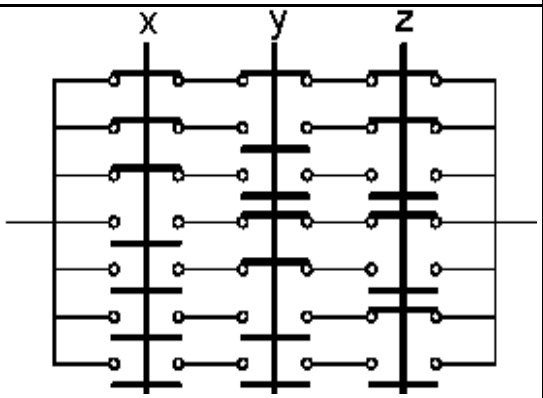
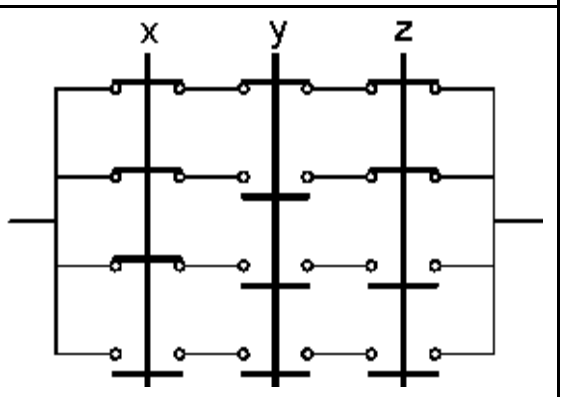
$$\overline{x + y}\cdot(x\cdot\bar{y}) = \bar{x}\cdot\bar{y}\cdot(x\cdot\bar{y}) = \bar{x}\cdot x\cdot\bar{y}\cdot\bar{y} = 0\cdot\bar{y}\cdot\bar{y} = 0\cdot\bar{y} = 0.$$

Закони алгебри логіки застосовуються у такій послідовності: правило де Моргана, сполучний закон, правило операцій змінної з її інверсією і правило операцій з константами.

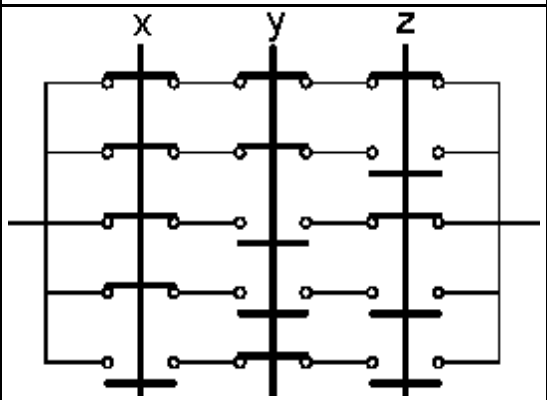
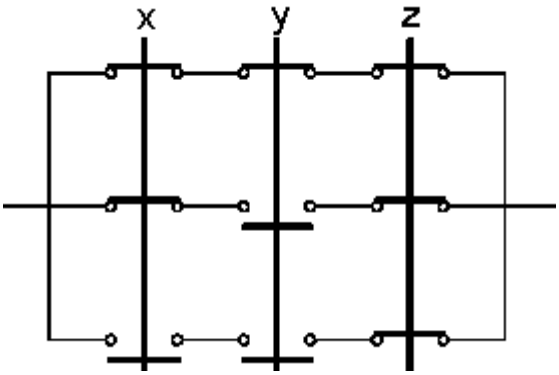
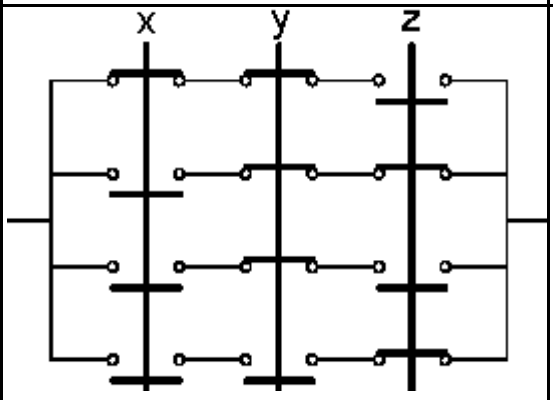
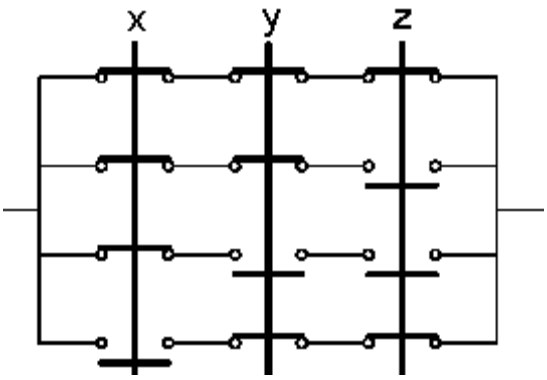
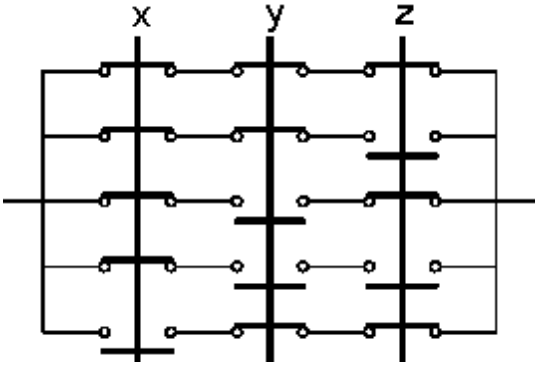
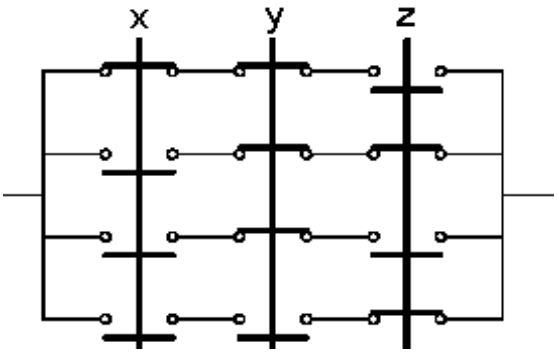
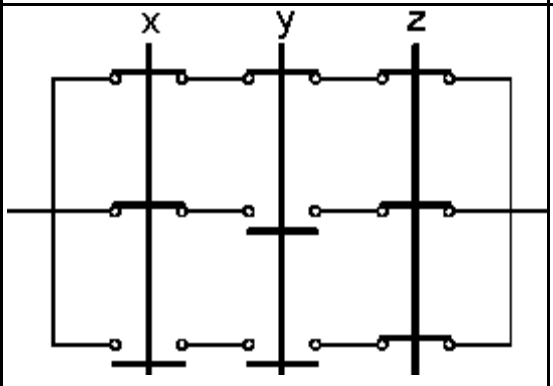
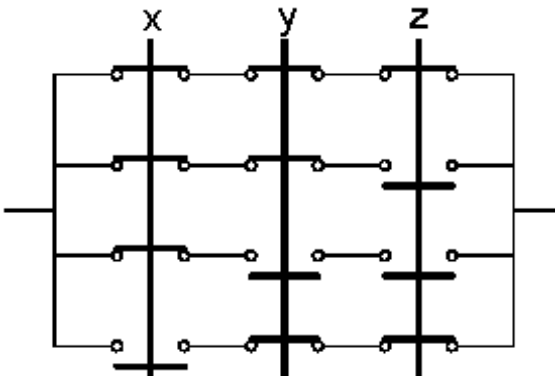
Завдання 1.2. Слід спростити схему перемикача (табл. 1.4 і 1.5).

Таблиця 1.4

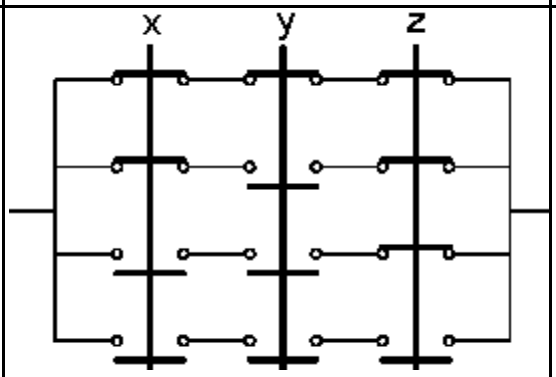
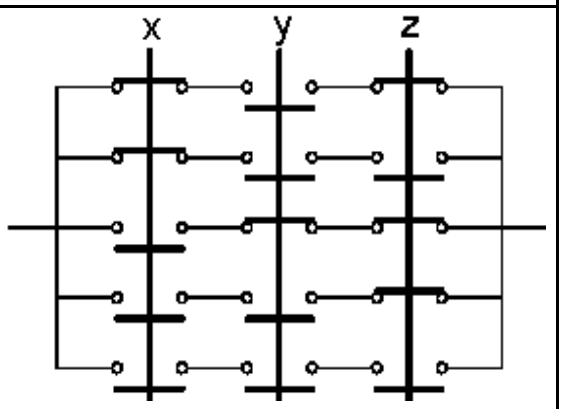
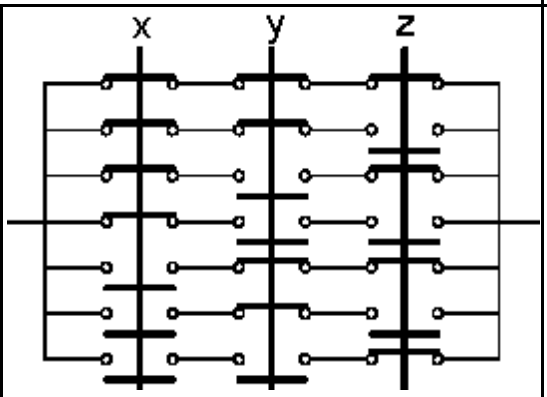
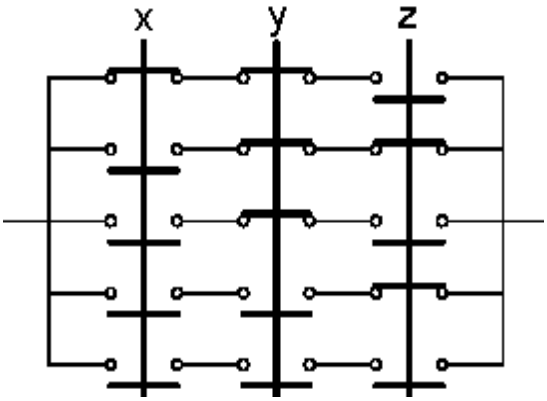
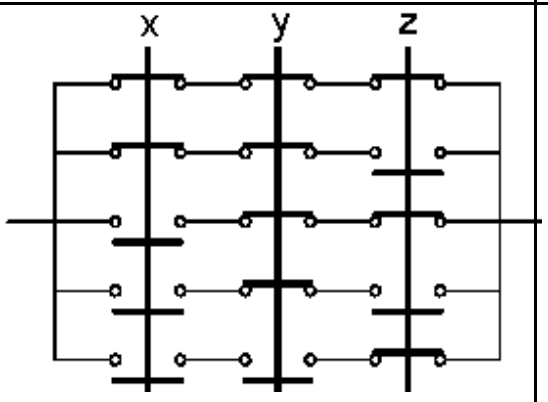
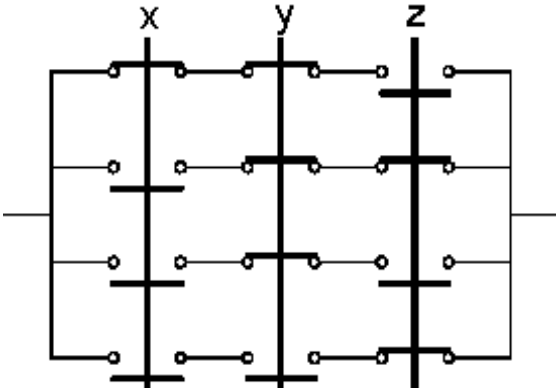
Спростити схему перемикача

№ п/п	Схема перемикача	№ п/п	Схема перемикача
1	2	3	4
1		10	
2		11	

Продовження табл. 1.4

1	2	3	4
3		12	
4		13	
5		14	
6		15	

Закінчення табл. 1.4

1	2	3	4
7		16	
8		17	
9		18	

Умовні позначення: x, y, z – увімкнено, $\bar{x}$, $\bar{v}$, $\bar{z}$ – вимкнено.

Таблиця 1.5

Спрощення схем перемикачів (результати)

№ схеми	Результат	№ схеми	Результат	№ схеми	Результат
1	$\bar{z}\bar{y}$ v x	7	$\bar{x}\bar{y}$ v xz	13	$x\bar{z}$ v yz
2	$\bar{x}$ v $\bar{y}$ v z	8	x v y v z	14	$\bar{x}z$ v $y\bar{z}$
3	$\bar{z}$y v x	9	$\bar{x}z$ v y	15	$x\bar{z}$ v yz
4	$\bar{x}z$ v $y\bar{z}$	10	$x\bar{z}$ v y	16	$\bar{x}z$ v $\bar{y}$
5	zy v x	11	xz v $\bar{y}\bar{z}$	17	$\bar{z}$y v $\bar{x}$
6	xz v $\bar{y}z$	12	xz v $\bar{y}z$	18	$\bar{x}z$ v $y\bar{z}$

Питання для самоперевірки

1. Що таке система числення?
2. Що є позиційною системою числення?
3. Чим відрізняються позиційні системи числення від непозиційних?
4. Чи може як цифра використовуватися символ букви?
5. Яка кількість цифр використовується в n-й системі числення?
6. Як виконуються арифметичні операції у вісімковій системі числення?
7. Як виконуються арифметичні операції в шістнадцятковій системі числення?
8. Як виконуються арифметичні операції в двійковій системі числення?
9. Який порядок переведення цілих чисел із системи числення з підставою n у десяткову систему числення?
10. Який порядок переведення правильних дробів із системи числення з підставою в десяткову систему числення?
11. Який порядок переведення цілих чисел із десяткової системи числення в систему числення з підставою n?
12. Який порядок переведення правильних дробів із десяткової системи числення в систему числення з підставою n?
13. Який порядок переведення неправильних дробів із однієї системи числення в іншу?
14. Що є прямим, зворотним і додатковим кодами, для чого вони використовуються?
15. Що таке вираз? Чи може бути вираз у формі питальної пропозиції?
16. Що таке диз'юнкція? Коли диз'юнкція істинна?
17. Що таке кон'юнкція? Коли кон'юнкція істинна?
18. Коли заперечення істинне? Коли заперечення хибне?
19. Чому відповідає заперечення заперечення?
20. Чому відповідає заперечення кон'юнкції?
21. Чому відповідає заперечення диз'юнкції?
22. Що таке імплікація? Коли імплікація істинна?
23. Що таке логічний вираз?
24. Як визначається порядок виконання логічних операцій у складних виразах?

25. Що містять таблиці істинності логічних виразів і який порядок їх побудови?

26. Які базові логічні елементи реалізують три основні логічні операції?

Лабораторна робота 2
Управління процесами і потоками в середовищі ОС Windows

Мета: дослідження і практична реалізація механізмів створення багато потокових програм.

2.1. Теоретичні посилання

Багатопоточність дозволяє додаткам розподіляти завдання і працювати над кожним незалежно, аби максимально ефективно задіювати процесор і призначений для користувача час.

2.1.1. Потоки і багатозадачність

Потік є одиницею обробки даних, а багатозадачність – це одночасне виконання декількох потоків. Існує два види багатозадачності – спільна (cooperative) і витіснювальна (preemptive). Ранні версії Microsoft Windows підтримували спільну багатозадачність. Це означало, що кожен потік відповідав за повернення управління процесору, аби той зміг обробити інші потоки. Той, хто раніше працював з іншими ОС (наприклад, OS/2), може розповісти історію, як одного разу він "підвісив" комп'ютер, забувши зробити виклик PeekMessage, аби забезпечити обслуговування процесором інших потоків у системі.

Проте Microsoft Windows NT і (пізніше) Windows 95, Windows 98 і Windows 2000 стали підтримувати витіснювальну багатозадачність, яку підтримувала OS/2. У ході цього процесор відповідає за видачу кожному потоку певної кількості часу, протягом якого потік може виконуватися, – кванта часу (timeslice). Далі процесор перемикається між різними потоками, видаючи кожному потоку його квант часу, а програміст може не піклуватися про те, як і коли повертати управління, унаслідок чого можуть працювати й інші потоки.

2.1.2. Перемикання контексту

Процесор за допомогою апаратного таймера визначає момент закінчення кванта, виділеного для даного потоку. Коли апаратний таймер генерує переривання, процесор зберігає у стеку вміст усіх регістрів для

даного потоку. Потім процесор переміщує вміст цих же регістрів у структуру даних CONTEXT. У разі необхідності перемикання назад на потік, що виконувався раніше, процесор виконує зворотну процедуру і відновлює вміст регістрів зі структури CONTEXT, що асоціюється з потоком. Увесь цей процес називається перемиканням контексту.

2.1.3. Багатопотокове застосування на C#

Перш ніж вивчати способи використання потоків на C#, слід розглянути, як легко створити вторинний потік на C#. У даному прикладі показано, як створити другий потік у методі Main. Метод, що асоціюється з іншим потоком, виводить рядок, що сигналізує про виклик цього потоку.

```
using System;
using System.Threading;
class SimpleThreadApp {
Public static void WorkerThreadMethod() {
Console.WriteLine("Worker thread started");
}
public static void Main() {
ThreadStart worker = new ThreadStart(WorkerThreadMethod);
Console.WriteLine("Main - Creating worker thread");
Thread t = new Thread(worker); t.Start();
Console.WriteLine ("Main - Have requested the start of worker thread");
} }
```

Скомпілювавши і запустивши це застосування, можна побачити (рис. 2.1), що повідомлення методу Main виводиться перед повідомленням робочого потоку. Це доводить, що робочий потік дійсно працює асинхронно.

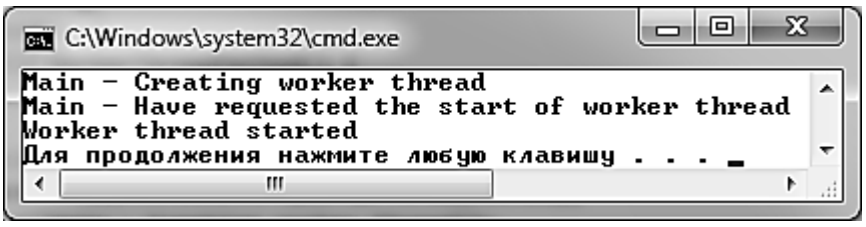


Рис. 2.1. **Результат роботи програми**

Слід проаналізувати події, що тут відбуваються.

Перший новий елемент, на який слід звернути увагу, – це оператор using у просторі імен System. Threading. Простір імен містить класи, необхідні для організації потоків у середовищі .NET. Варто розглянути перший рядок методу Main:

```
ThreadStart WorkerThreadMethod = new
ThreadStart(WorkerThreadMethod);
```

ThreadStart – це делегат, який задіяний під час створення нового потоку. Він використовується, аби задати метод, який повинен викликатися як метод потоку. Тут створюється екземпляр об'єкта Thread, при цьому конструктор приймає як аргумент лише делегат ThreadStart:

```
Thread t = new Thread(worker);
```

Після цього викликається метод Start об'єкта Thread і далі метод WorkerThreadMethod.

2.1.4. Робота з потоками

Практично вся робота з потоками відбувається з використанням класу Thread.

Можна створювати екземпляри об'єкта Thread двома способами. Один уже розглянуто, а саме створення нового потоку і здобуття об'єкта Thread, що дозволяє маніпулювати новим потоком у цьому процесі. Інший спосіб отримати об'єкт Thread для потоку, що виконується в даний момент, – це виклик статичного методу Thread. CurrentThread.

Аби управляти активністю або часом життя потоку, треба вирішити багато завдань. Із цим дозволяють впоратися методи Thread. Наприклад, досить часто потік потрібно припинити. Це можна зробити, викликавши метод Thread.Sleep, що приймає єдиний аргумент, що є часом (у мілісекундах), на який потрібно припинити потік. Слід зазначити, що метод Thread.Sleep є статичним і не може бути викликаний з екземпляром об'єкта Thread. На те є вагома причина. Не допускається викликати Thread.Sleep для будь-якого іншого потоку, окрім того, що виконується у даний момент. Статичний метод Thread.Sleep викликає статичний метод CurrentThread, який потім припиняє цей потік на вказаний час. Приклад:

```
using System;
using System.Threading;
class ThreadSleepApp {
public static void WorkerThreadMethod() {
Console.WriteLine("Worker thread started");
int sleepTime = 5000;
Console.WriteLine("\tsleeping for {0} seconds", sleepTime / 1000);
Thread.Sleep(sleepTime); // Призупинення на п'ять секунд.
Console.WriteLine("\twaking up");
}
public static void Main() {
ThreadStart worker = new ThreadStart(WorkerThreadMethod);
Console.WriteLine("Main - Creating worker thread");
Thread t = new Thread(worker);
t.Start();
Console.WriteLine ("Main - Have requested the start of worker thread");
Console.ReadLine();
} }
```

Є ще два способи виклику методу Thread.Sleep. Перший – це виклик Thread.Sleep зі значенням 0. У ході цього поточний потік звільнить невикористаний залишок свого кванта. У процесі передачі значення Timeout.Infinite потік буде припинений на невизначено довгий термін, поки це стан потоку не буде перерваний іншим потоком, що викликав метод припиненого потоку Thread.Interrupt.

Другий спосіб припинити виконання потоку – викликати методу Thread.Suspend.

Між цими методами існує декілька важливих відмінностей. По-перше, можна викликати метод Thread. Suspend для потоку, що виконується у даний момент, або для іншого потоку. По-друге, якщо таким чином припинити виконання потоку, будь-який інший потік здатний відновити його виконання за допомогою методу Thread.Resume.

Слід звернути увагу, що коли один потік припиняє виконання іншого, то перший потік не блокується. Повернення управління після виклику відбувається негайно. Крім того, єдиний виклик Thread.Resume відновить виконання даного потоку незалежно від кількості викликів методу Thread.Suspend, виконаних раніше.

Знищити потік можна викликом методу Thread.Abort. Виконуюче середовище примусово завершує виконання потоку, генеруючи вимкнення ThreadAbortException. Навіть якщо метод спробує уловити ThreadAbort-Exception, то виконуюче середовище цього не допустить. Проте виконає код із блоку finally потоку, виконання якого перерване, якщо цей блок присутній. Це проілюстровано наступним кодом. Виконання методу Main припиняється на 2 секунди, аби дати середовищу, що виконується, час для запуску робочого потоку. Після запуску робочий потік рахує до десяти, зупиняючись після кожного відліку на секунду. Коли виконання методу Main поновлюється після двосекундної паузи, він перериває виконання робочого потоку. Після цього виконується блок finally.

```
using System;
using System.Threading;
class ThreadAbortApp {
public static void WorkerThreadMethod() {
try {
Console.WriteLine("Worker thread started");
Console.WriteLine ("Worker thread - counting slowly to 10");
for (int i = 0; i < 10; i++) {
Thread. Sleep(1000);
Console.Write("{0}...", i);
}
Console.WriteLine("Worker thread finished");
}
catch(ThreadAbortException e){
}
finally {
Console.WriteLine ("Worker thread - I can't catch the exception, but I can cleanup");
} }
public static void Main() {
ThreadStart worker = new ThreadStart(WorkerThreadMethod);
Console.WriteLine("Main - Creating worker thread");
Thread t = new Thread(worker);
t.Start();
```

```
// Даємо час для запуску робочого потоку.
Console.WriteLine("Main - Sleeping for 2 seconds");
Thread.Sleep(2000);
Console.WriteLine("\nMain - Aborting worker thread");
t.Abort();
Console.ReadLine();
} }
```

Скомпілювавши і запустивши це застосування, буде отримано такий результат (рис. 2.2).

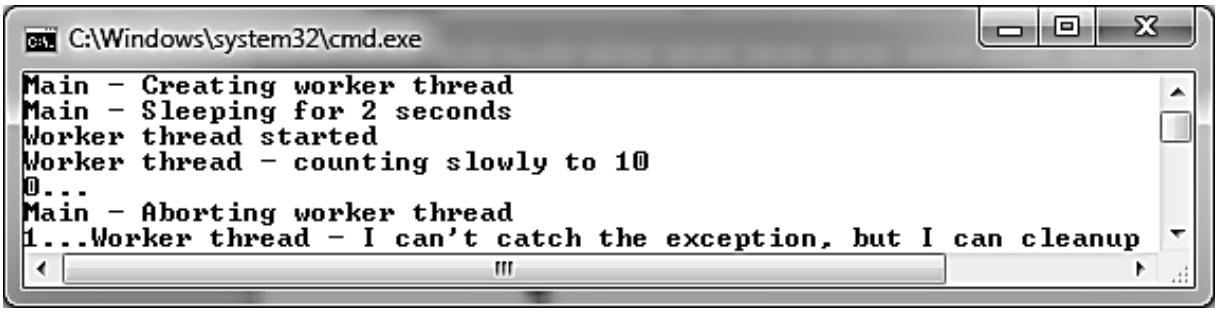


Рис. 2.2. **Результат роботи програми**

Варто усвідомлювати те, що під час виклику методу Thread.Abort виконання потоку не може зупинитися відразу. Середовище, що виконується, чекає, поки потік не досягне тієї крапки, яка в документації названа "безпечною крапкою" ("safe point"). Тому якщо програма залежить від деяких дій, які відбуваються після переривання потоку, і треба впевнитися в тому, що потік зупинений, то можна використовувати метод Thread.Join.

Це синхронний виклик, тобто він не поверне управління, поки потік не буде зупинений.

2.1.5. Планування потоків

У ході перемикання процесора після закінчення виокремленого потоку кванта часу, процес вибору наступного потоку, призначеного для виконання, зовсім не довільний. У кожного потоку є пріоритет, вказуючий процесору, як повинне плануватися виконання цього потоку щодо інших потоків системи. Для потоків, що створюються в період виконання, рівень пріоритету за замовчуванням рівний Normal. Потоки, створені не в період виконання, зберігають свій вихідний пріоритет. Для перегляду й установки

цього значення слугує властивість Thread.Priority. Установник властивості Thread.Priority приймає аргумент типу Thread.ThreadPriority, який є enum, що визначає значення Highest, AboveNormal, Normal, BelowNormal, Lowest.

Аби проілюструвати, як пріоритети можуть впливати навіть на простий код, слід навести приклад, де один робочий потік рахує від 1 до 10, а інший – від 11 до 20. Варто звернути увагу на вкладений цикл у кожному методі WorkerThread. Цикли використовуються тут для подання дій, які виконував би цей додаток. Оскільки насправді ці методи нічого не роблять, то без циклів кожен потік завершив би свою роботу протягом першого ж свого кванта часу.

```
using System;
using System.Threading;
class ThreadSchedulelApp {
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread started");
Console.WriteLine("Worker thread - counting slowly from 1 to 10");
for (int i=1; i < 11; i++) { for (int j = 0; j < 100; j++) {
Console.Write(".");
int a; a = 15; }
Console.Write("{0}", i); }
Console.WriteLine("Worker thread finished"); }
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread started");
Console.WriteLine("Worker thread - counting slowly from 11 to 20");
for (int i = 11; i < 20; i++) { for (int j = 0; j < 100; j++) {
Console.Write(".");
int a; a = 15; }
Console.Write("{0}", i); }
Console.WriteLine("Worker thread finished"); }
public static void Main() {
ThreadStart worker1 = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
Console.WriteLine("Main - Creating worker threads");
```

```
Thread t1 = new Thread(worker1);
Thread t2 = new Thread(worker2);
t1.Start(); t2.Start(); Console.ReadLine(); } }
```

Запустивши це застосування, буде отримано таке (рис. 2.3) (для кращого розуміння більша частина крапок прибрана).

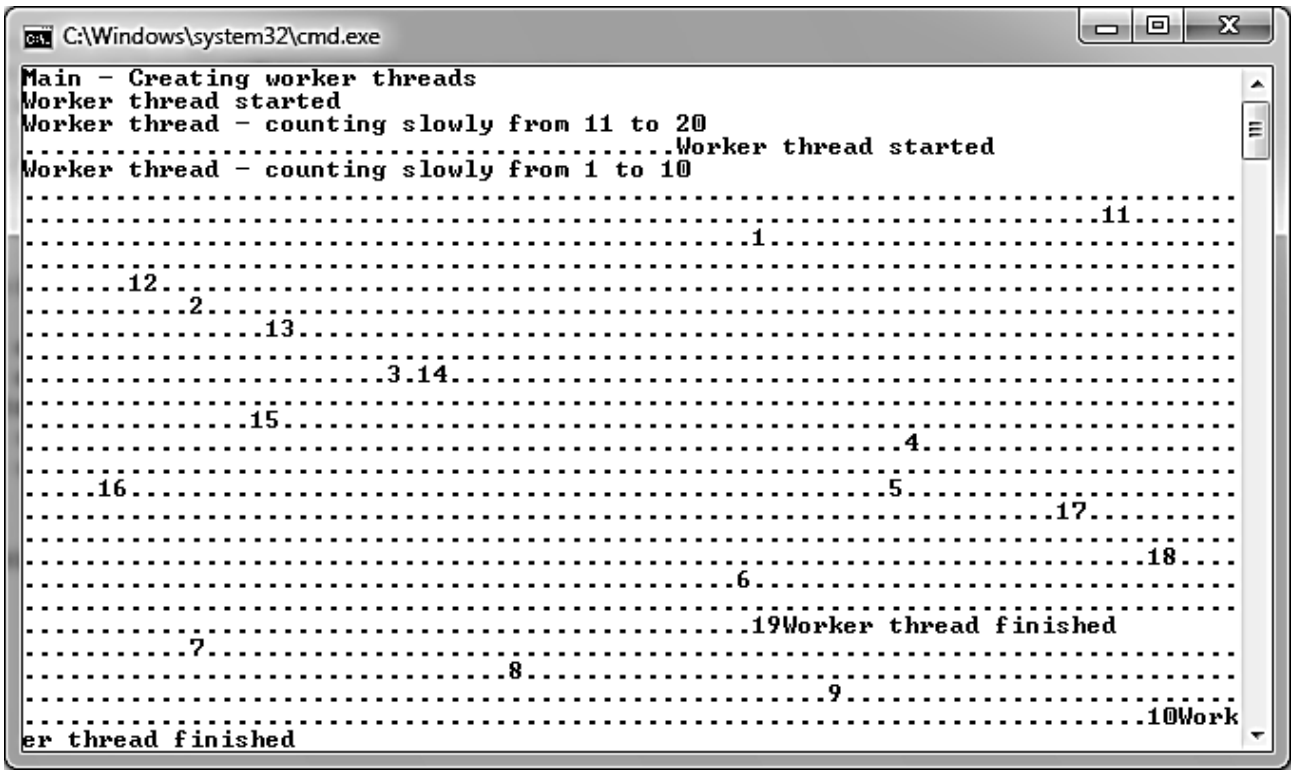


Рис. 2.3. **Робота потоків з однаковими пріоритетами**

Як можна бачити, обидва потоки отримують однаковий час для "гри" з процесором. А тепер необхідно змінити властивість Priority кожного потоку, як це зроблено в наступному коді (найбільший пріоритет із можливих першому потоку і найменший – другому). Результати будуть відрізнятися від попередніх.

```
using System;
using System.Threading;
class ThreadSchedule2App {
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread started");
```

```
Console.WriteLine ("Worker thread - counting slowly from 1 to 10");
for (int 1=1; i < 11; i++) {
for (int j = 0; j < 100; j++) {
Console.Write(".");
// Код, який імітує роботу, що підлягає виконанню.
int a; a = 15; }
Console.Write("{0}", i); }
Console.WriteLine("Worker thread finished"); }
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread started");
Console.WriteLine
("Worker thread - counting slowly from 11 to 20");
for (int i = 11; i < 20; i++) {
for (int j = 0; J < 100; J++) {
Console.Write(".");
// Код, який імітує роботу, що підлягає виконанню, int a; a = 15;
}
Console.Write("{0}", i); }
Console.WriteLine("Worker thread finished"); }
public static void Main() {
ThreadStart workerl = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
Console.WriteLine("Main - Creating worker threads");
Thread t1 = new Thread(workerl);
Thread t2 = new Thread(worker2);
t1.Priority = ThreadPriority.Highest;
t2.Priority = ThreadPriority.Lowest;
t1.Start(); t2.Start(); } }
```

Результат показано далі (рис. 2.4). Зазначте, що в другого потоку був встановлений такий низький пріоритет, що він не зміг виконати жодного циклу, поки перший потік не закінчив свою роботу.

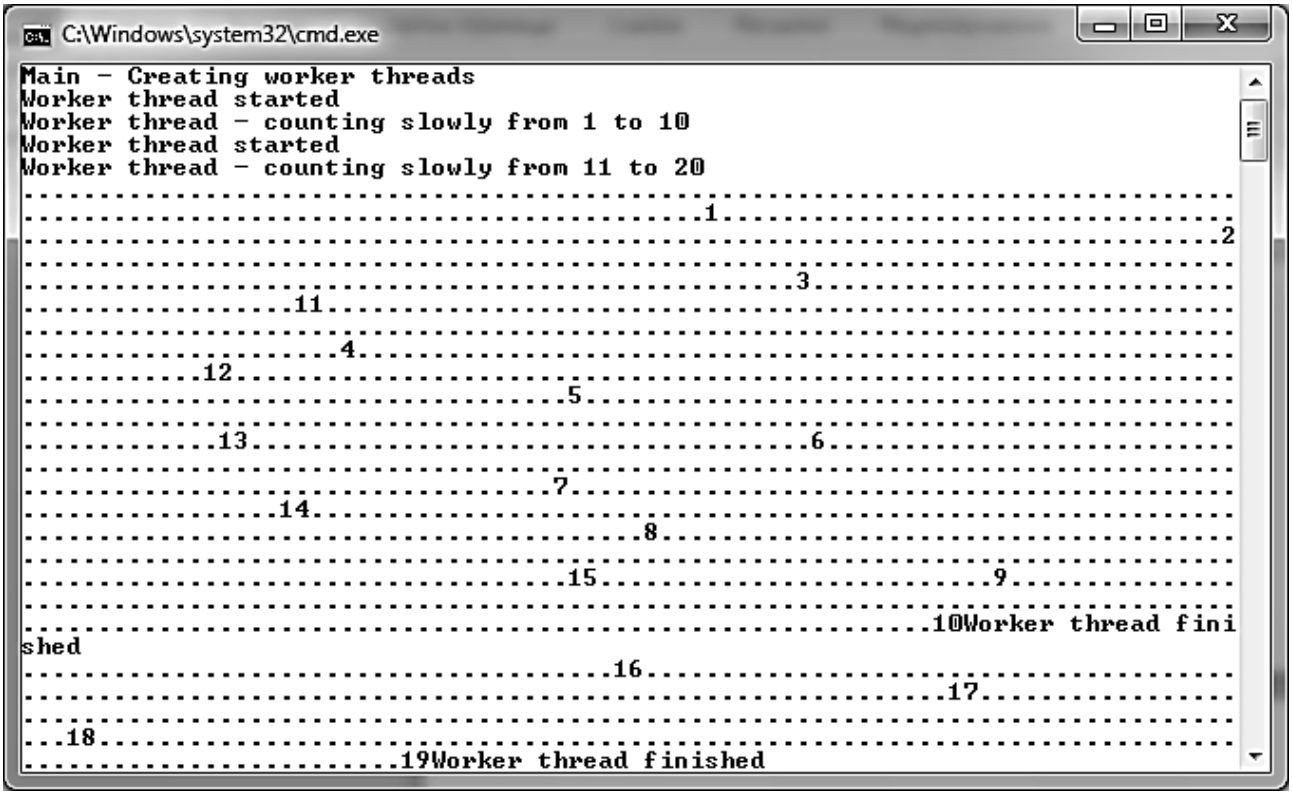


Рис. 2.4. **Робота потоків із різними пріоритетами**

Слід пам'ятати, що коли вказується пріоритет процесора, який необхідно встановити для даного потоку, то його значення використовується ОС як частина алгоритму планування розподілу процесорного часу. У .NET цей алгоритм заснований на рівнях пріоритету, які тільки що було використано (за допомогою властивості Thread.Priority), а також на класі пріоритету (priority class) процесу і значенні динамічного підвищення (dynamic boost) пріоритету. За допомогою всіх цих значень створюється числове значення (для процесорів Intel – 0 – 31), що становить пріоритет потоку. Потоки з найбільшим значенням є найпріоритетнішими.

Якщо встановити дуже високий пріоритет робочих потоків, робота призначеного для користувача інтерфейсу може сповільнитися, оскільки головний потік, у якому виконується додаток із графічним інтерфейсом, отримує менше циклів процесора. Таким чином, якщо немає особливої причини планувати потік із високим пріоритетом, то найкраще залишити біля потоку пріоритет за замовчуванням – Normal.

2.1.6. Безпека і синхронізація потоків

Під час програмування для однопотокового оточення методи часто пишуться так, що на деяких етапах виконання коду об'єктів тимчасово перебувають у недійсному стані. Вочевидь, що якщо у будь-який момент

до об'єкта звертається лише один потік, то є гарантія, що кожен метод завершиться до того, як буде викликаний наступний метод. Це означає, що для своїх клієнтів об'єкт завжди перебуває в дійсному стані. Проте коли декілька потоків працюють одночасно, можна легко отримати ситуації, в яких процесор перемикається на інший потік, тоді як об'єкт знаходиться в недійсному стані. Якщо потім цей потік також спробує використовувати цей же об'єкт, результат буде абсолютно непередбачуваним. Тому термін "безпека потоків" означає постійну підтримку членів об'єкта в дійсному стані під час їх одночасного використання декількома потоками.

Як уникнути подібних непередбачуваних станів? Насправді, як це зазвичай буває в програмуванні, існує декілька способів вирішення цієї добре відомої проблеми. Існує стандартний засіб – синхронізація. Синхронізація дозволяє задавати критичні секції (critical sections) коду, в які в кожен окремий момент може входити лише один потік, гарантуючи, що будь-які тимчасові недійсні стани об'єкта будуть невидимі його клієнтам. Слід розглянути декілька засобів визначення критичних секцій, включаючи класи .NET Monitor і Mutex, а також оператор C# lock.

2.1.7. Захист коду за допомогою класу Monitor

Клас System.Monitor дозволяє упорядкувати звернення до блоків коду за допомогою блокування і звільнення. Наприклад, метод, що оновлює БД, не може виконуватися двома і більше потоками одночасно. Якщо виконувана ним робота вимагає особливо багато часу й існує декілька потоків, будь-який із них може його викликати, ймовірне виникнення серйозної проблеми. Починається реалізація класу Monitor. Варто поглянути на приклад синхронізації. Тут існують два потоки, кожен із яких викликатиме метод Database.SaveData.

```
using System;
using System.Threading;
class Database {
public void SaveData(string text){
Console.WriteLine("Database.SaveData - Started");
Console.WriteLine("Database.SaveData - Working");
for (int i=0; i < 100; i++) {
Console.Write(text); }
Console.WriteLine("\nDatabase.SaveData - Ended"); } }
class ThreadMonitorlApp {
public static Database db = new Database();
```

```
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread *1 - Started");
Console.WriteLine ("Worker thread <M -Calling Database.SaveData");
db.SaveData("x");
Console.WriteLine("Worker thread *1 - Returned from Output"); }
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread #2 - Started");
Console.WriteLine("Worker thread #2 - Calling Database.SaveData");
db.SaveData("o");
Console.WriteLine("Worker thread *2 - Returned from Output"); }
public static void Main() {
ThreadStart workerl = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
Console.WriteLine("Main - Creating worker threads");
Thread t1 = new Thread(workerl);
Thread t2 = new Thread(worker2);
t1.Start(); t2.Start();
Console.ReadLine(); } }
```

Скомпілювавши і запустивши це застосування, можна побачити (рис. 2.5), що отримана в результаті вихідна інформація складатиметься з довільного набору символів "o" і "x". Це свідчить про те, що виконання методу Database.SaveData одночасно запускається обома потоками.

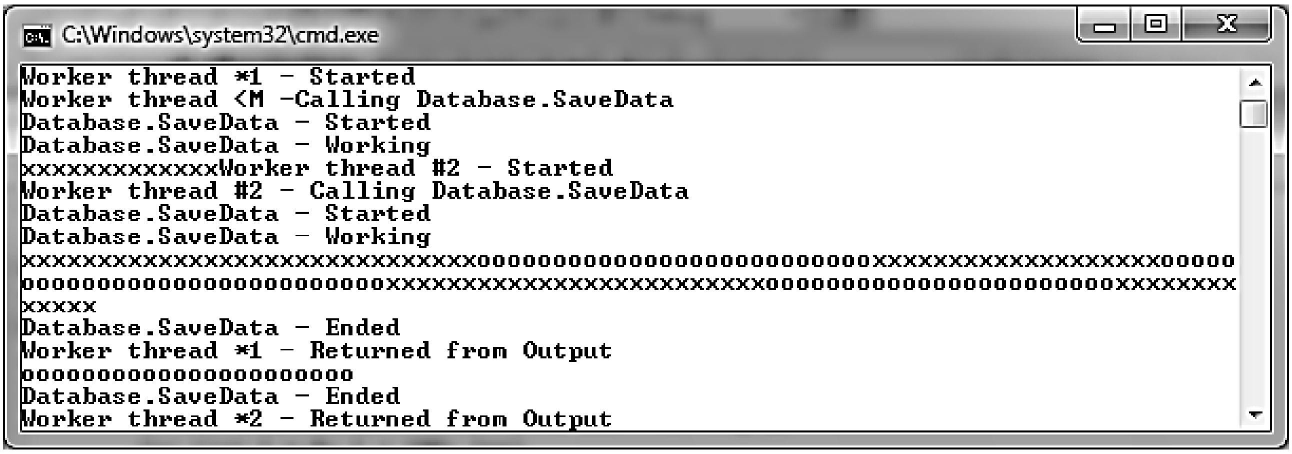


Рис. 2.5. **Результат роботи методу Database.SaveData**

Вочевидь, якщо методу Database.SaveData потрібно завершити оновлення декількох таблиць, перш ніж він буде викликаний іншим потоком, виникне серйозна проблема.

Для включення в цей приклад класу Monitor можна скористатися двома його статичними методами. Перший – Enter – під час виконання намагається отримати блокування монітора для об'єкта. Якщо в іншого потоку вже є це блокування, то метод блокується до того часу, поки блокування не буде звільнено.

Слід зазначити, що не виконується неявна операція пакування, тому для цього методу можна надавати лише посилальні типи. Потім викликається метод Monitor.Exit, аби звільнити блокування. Далі наведено приклад, переписаний для впорядкованого звернення до методу Database.SaveData.

```
using System;
using System.Threading;
class Database {
public void SaveData(string text){
Monitor.Enter(this);
Console.WriteLine("Database.SaveData - Started");
Console.WriteLine("Database.SaveData - Working");
for (int i = 0; i < 100; i++) {
Console.Write(text); }
Console.WriteLine("\nDatabase.SaveData - Ended");
Monitor.Exit(this); } }
class ThreadMonitor2App {
public static Database db = new Database();
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread #1 - Started");
Console.WriteLine ("Worker thread #1 - Calling Database.SaveData");
db.SaveData("x");
Console.WriteLine("Worker thread t1 - Returned from Output"); }
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread #2 - Started");
Console.WriteLine ("Worker thread #2 - Calling Database.SaveData");
db.SaveData("o");
Console.WriteLine("Worker thread #2 - Returned from Output"); }
public static void Main() {
ThreadStart worker1 = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
```

```
Console.WriteLine("Main - Creating worker threads");
Thread t1 = new Thread(worker1); Thread t2 = new Thread(worker2);
t1.Start(); t2.Start(); Console.ReadLine(); } }
```

У наведеній далі вихідній інформації (рис. 2.6) слід звернути увагу на те, що навіть якщо другий потік викликав метод Database.SaveData, метод Monitor.Enter блокував його до того часу, поки перший потік не звільняв утримуване ним блокування.

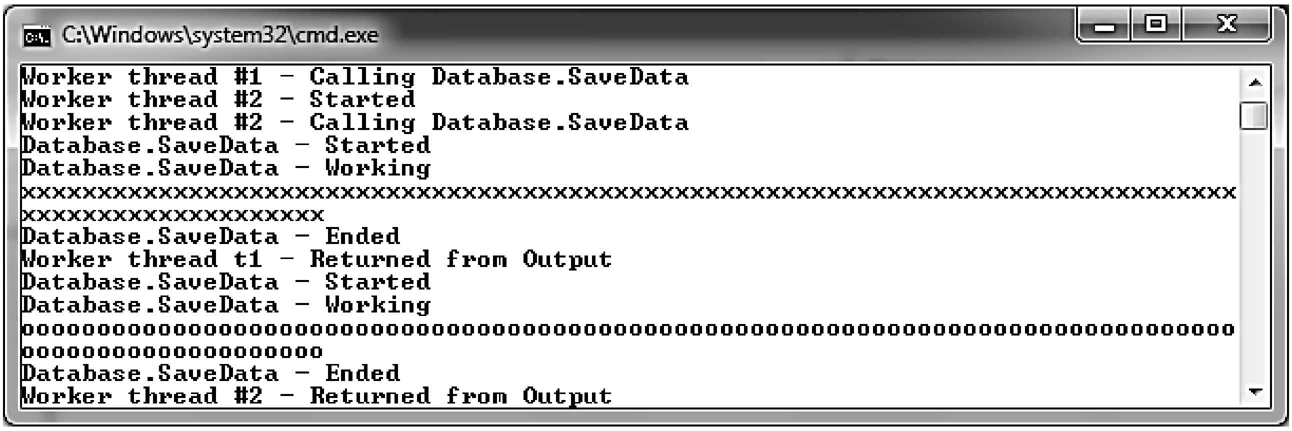


Рис. 2.6. **Результат роботи методу Monitor.Enter**

2.1.8. Застосування блокувань монітора з оператором C# lock

Оператор C# lock не підтримує повний набір функцій класу Monitor, але все ж таки дозволяє отримувати і звільняти блокування монітора. Аби задіювати оператора lock, слід указати його у фігурних дужках разом з упорядковуваним кодом. Початкова і кінцева точки коду, що захищається, вказуються фігурними дужками, тому немає потреби використовувати оператора unlock. Наступний код видасть таку ж синхронізовану інформацію, що і в попередніх прикладах (рис. 2.7).

```
using System;
using System.Threading;
class Database {
public void SaveData(string text){
lock(this){
Console.WriteLine("Database.SaveData - Started");
Console.WriteLine("Database.SaveData - Working");
for (int i = 0; i < 100; i++) {
```

```
Console.Write(text); }
Console.WriteLine("\nDatabase.SaveData - Ended"); } } }
class ThreadLockApp {
public static Database db = new Database();
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread #1 - Started");
Console.WriteLine ("Worker thread #1 - Calling Database.SaveData");
db.SaveData("x");
Console.WriteLine("Worker thread #1 - Returned from Output"); }
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread #2 - Started");
Console.WriteLine ("Worker thread *2 - Calling Database.SaveData");
db.SaveData("o");
Console.WriteLine("Worker thread #2 - Returned from Output"); }
public static void Main() {
ThreadStart worker1 = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
Console.WriteLine("Main - Creating worker threads");
Thread t1 = new Thread(worker1);
Thread t2 = new Thread(worker2);
t1.Start(); t2.Start();
Console.ReadLine(); } }
```

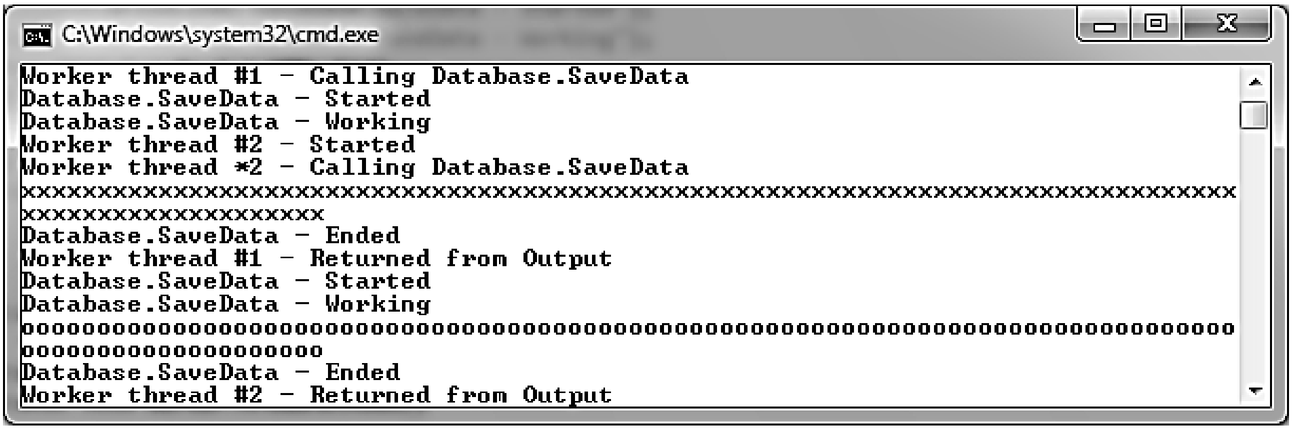


Рис. 2.7. **Робота оператора C # lock**

2.1.9. Синхронізація коду за допомогою класу Mutex

Клас Mutex, визначений в просторі імен System.Threading, – це подання примітиву системи Win32 з тим же ім'ям для періоду виконання.

Можна використовувати Mutex для впорядковування звернень до коду так само, як блокування монітора, але Mutex набагато повільніше через свою універсальність. Термін Mutex походить від фрази mutually exclusive (що взаємно виключає), і оскільки лише один потік може отримати блокування монітора для даного об'єкта у будь-який момент часу, лише один потік у будь-який момент часу може отримати даний Mutex.

Три конструктори дозволяють створювати Mutex на C#:

Mutex();

Mutex(bool спочатку_блокований);

Mutex(bool спочатку_блокований, string ім'я_м'ютекса).

Перший створює безіменний м'ютекс і робить поточний потік його власником, тому м'ютекс блокується поточним потоком. Другий приймає тільки логічний прапор, який визначає, чи збирається потік, що створює м'ютекс, заволодіти ним (заблокувати його). Третій дозволяє вказувати, чи володіє поточний потік м'ютексом, а також задавати ім'я м'ютекса. Слід розглянути м'ютекс, що застосовується для впорядковування звернень до методу Database.SaveData (рис.2.8).

```
using System;
using System.Threading;
class Database {
Mutex mutex = new Mutex(false);
public void SaveData(string text){
mutex.WaitOne();
Console.WriteLine("Database.SaveData - Started");
Console.WriteLine("Database.SaveData - Working");
for (int i = 0; i < 100; i++) {
Console.Write(text); }
Console.WriteLine("\nDatabase.SaveData - Ended");
mutex.Close(); } }
class ThreadMutexApp {
public static Database db = new Database();
public static void WorkerThreadMethod1() {
Console.WriteLine("Worker thread #1 - Started");
Console.WriteLine ("Worker thread #1 - Calling Database.SaveData");
db. SaveData("x");
```

```
Console.WriteLine("Worker thread #1 - Returned from Output"); }
public static void WorkerThreadMethod2() {
Console.WriteLine("Worker thread #2 - Started");
Console.WriteLine ("Worker thread #2 - Calling Database.SaveData");
db.SaveData("o");
Console.WriteLine("Worker thread 92 - Returned from Output"); }
public static void Main() {
ThreadStart worker1 = new ThreadStart(WorkerThreadMethod1);
ThreadStart worker2 = new ThreadStart(WorkerThreadMethod2);
Console.WriteLine("Main - Creating worker threads");
Thread t1 = new Thread(worker1); Thread t2 = new Thread(worker2);
t1.Start(); t2.Start();Console.ReadLine(); } }
```

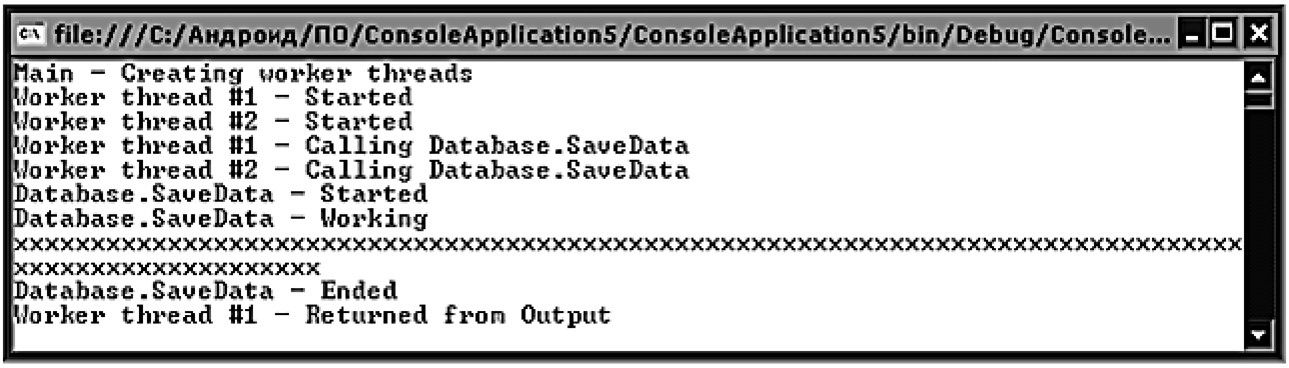


Рис. 2.8. **Застосування м'ютексів для синхронізації потоків**

Тепер у класі Database визначено поле Mutex. Не слід допускати того, аби потік володів м'ютексом просто тому, що у ході цього буде неможливо звернутися до методу SaveData. Перший рядок методу SaveData показує, що треба намагатися отримати м'ютекс за допомогою методу Mutex.WaitOne. У кінці методу викликається метод Close, звільняючий м'ютекс.

Метод WaitOne переобтяжений аби забезпечити велику гнучкість у наданні можливості визначення, наскільки довго потік чекатиме звільнення м'ютекса. От як він перевантажується:

WaitOne();

WaitOne(TimeSpan час, bool покинути_контекст);

WaitOne(int мілісекунди, bool покинути_контекст).

Основна відмінність між цими способами перевантаження в тому, що перша версія (вона використана в прикладі) чекатиме невизначено

довго, а друга і третя чекатиме протягом вказаного проміжку часу, вираженого значенням типу TimeSpan або int.

Багатопоточність дозволяє додаткам розділяти завдання і вирішувати незалежно кожну з них, максимально ефективно використовуючи процесорний час. Проте багатопоточність є правильним вибором не завжди і деколи може уповільнити роботу додатка. Створення і управління потоками на C# здійснюється за допомогою класу System.Threading.Thread. Безпека потоків є важливим поняттям, пов'язаним зі створенням і використанням потоків. Безпека потоків означає, що члени об'єкта завжди підтримуються в дійсному стані під час одночасного використання декількома потоками. Необхідно, аби разом із вивченням синтаксису багатопоточності було зрозуміло, коли її застосовувати, а саме: для підвищення паралелізму, спрощення структури й оптимального використання процесорного часу.

2.2. Консольне застосування

Є просте консольне застосування, яке обчислює вказаний член послідовності Фібоначчі. Обчислення виробляється в окремому потоці, а ще в одному потоці в консоль виводитися напис "Виробляється обчислення...", що повідомляє користувача про те, що ще обчислюється (рис. 2.10).

```
using System;
using System.Threading;
namespace FibonacciNumbers {
public class Program { // Основний клас додатка.
private static int iMemberNum;
private static long lFibonacciNumber;
private static bool bStop;
private static AutoResetEvent LoadingAutoResetEvent;
private static AutoResetEvent CalculatingAutoResetEvent;
public static void Main() { // Точка входу в додаток.
Thread loadingThread = new Thread(LoadingThread);
Thread calculatingThread = new Thread(CalculatingThread);
LoadingAutoResetEvent = new AutoResetEvent(false);
CalculatingAutoResetEvent = new AutoResetEvent(false);
Console.Write("Номер члена послідовності Фібоначчі: ");
```

```
if(!int.TryParse(Console.ReadLine(), out iMemberNum)){
return; }
loadingThread.Start(); calculatingThread.Start();
WaitHandle.WaitAll(new AutoResetEvent[] {
CalculatingAutoResetEvent, LoadingAutoResetEvent });
Console.Clear();
Console.WriteLine("{0}-й член послідовності Фібоначчі
рівний {1}", iMemberNum, lFibonacciNumber);
Console.ReadLine(); }// Метод, який відображує процес обчислення.
public static void LoadingThread() {
int curLoadingStep = 0;
while (!bStop) {
LoadingAutoResetEvent.Reset(); Console.Clear();
switch (curLoadingStep) {
case 0:
Console.WriteLine("Виробляється обчислення");
curLoadingStep = 1; break;
case 1:
Console.WriteLine("Виробляється обчислення.");
curLoadingStep = 2; break;
case 2:
Console.WriteLine("Виробляється обчислення..");
curLoadingStep = 3; break;
case 3:
Console.WriteLine("Виробляється обчислення...");
curLoadingStep = 0; break; }
LoadingAutoResetEvent.Set(); Thread.Sleep(800); } }
// Обчислення члена послідовності Фібоначчі.
public static void CalculatingThread() {
lFibonacciNumber = GetFibonacciNumber(iMemberNum);
bStop = true; CalculatingAutoResetEvent.Set(); }
// Метод повертає число Фібоначчі з вказаним номером.
public static long GetFibonacciNumber(int _memberNum){
if (_memberNum == 0) return 0; if (_memberNum == 1) return 1;
return GetFibonacciNumber(_memberNum - 2)+
GetFibonacciNumber(_memberNum - 1); } } }
```

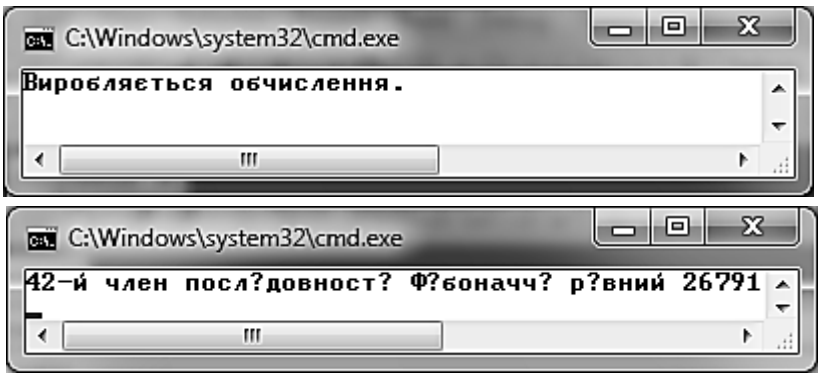


Рис. 2.9. **Обчислення послідовності Фібоначчі**

2.3. Графічне застосування

Зручність однопоточного програмування полягає в тому, що не потрібно замислюватися над синхронізацією потоків, код виходить набагато простіше і зрозуміліше. Але один істотний недолік такого підходу в додатках із графічним інтерфейсом користувача – під час виконання тривалої роботи вікно програми перестає відповідати на дії користувача і виникає відчуття, що додаток завис і не виконує жодної корисної роботи.

Для того, щоб продемонструвати, як працювати з багатопотоковим застосуванням, потрібне яке-небудь завдання, що виконується значний час. Прикладом буде завдання завантаження в компонент ListView файлів, розташованих у певній директорії. Сам процес здобуття імен файлів виконується вмить, а завантаження його в ListView займає значний час. Інтерфейс додатка виглядатиме так, як показано на рис. 2.10.

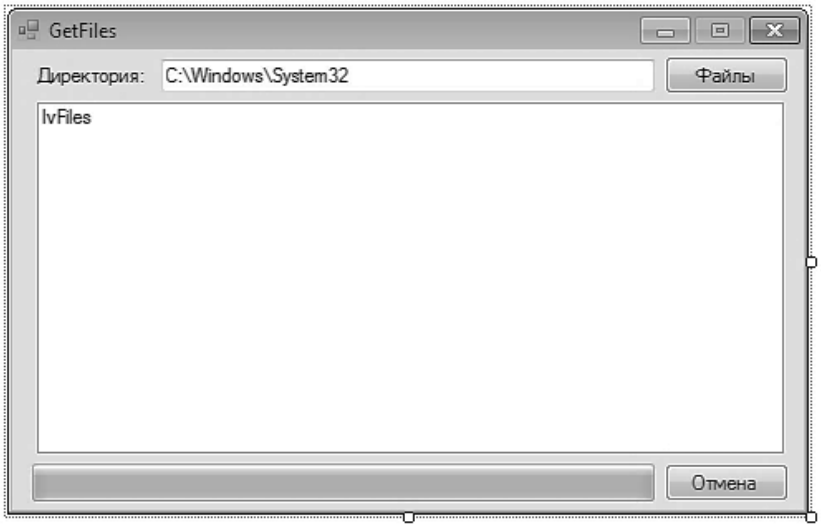


Рис. 2.10. **Інтерфейс додатка**

Як видно, на формі розташований компонент TextBox для задавання директорії, з якої буде отримано список файлів (за замовчуванням це C:\Windows\System32), Button "Файли" для запуску процесу, ListView для завантаження назв файлів, ProgressBar для відображення ходу процесу і Button "Відміна". У процесі запуску додатка кнопка "Відміна" невидима. Її властивість VIsible стає рівною true тоді, коли запускається процес перерахування файлів, а повертається в стан false після завершення процесу або ж у разі його відміни.

Простори імен, які необхідно підключити.

```
using System;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using System.Threading;
```

Потім у класі форми визначити необхідні об'єкти:

```
private delegate void AddToLV(string fileName);
private Thread newTh;
```

За допомогою другої змінної необхідно працювати з потоком, у якому здійснюватимуться тривалі за часом операції.

Для створення потоку буде використано процедуру.

```
private void WriteFiles(object _files) {
FileInfo[] files = (FileInfo[]) _files;
foreach (FileInfo fi in files) { PutInLV(fi.Name); } }
```

Сам же потік створюється натисненням кнопки "Файлів":

```
private void btnGet_Click(object sender, EventArgs e){
DirectoryInfo di = new DirectoryInfo(txtPath.Text);
if (!di.Exists) {
MessageBox.Show("Вказаної директорії не існує!");
return; }
```

```
btnCancel.Visible = true; FileInfo[] fi = di.GetFiles();
progressBar1.Maximum = fi.Count(); progressBar1.Value = 0;
lvFiles.Items.Clear();
newTh = new Thread(new ParameterizedThreadStart(WriteFiles));
newTh.Start(fi); }
```

Як видно з коду, спочатку відбувається перевірка, чи існує вказана директорія. Якщо її немає, то виводиться повідомлення і відбувається вихід із процедури. Якщо ж тека є, то кнопка відміни стає видимою, в об'єкт fi завантажується список файлів із цієї директорії, прогрес бару призначається максимальним значенням, рівним кількості файлів, встановлюється поточне положення в 0, очищається компонент ListView і нарешті створюється новий потік із указівкою на процедуру для створення потоку.

Що ж робить створений потік? Для кожного файла з переданого як аргумент масиву викликає іншу процедуру.

```
private void PutInLV(string fileName) {
if (lvFiles.InvokeRequired) {
AddToLV add = new AddToLV(PutInLV);
this.Invoke(add, new object[] { fileName }); }
Else { lvFiles.Items.Add(fileName);
progressBar1.Value++;
if (progressBar1.Value == progressBar1.Maximum)
btnCancel.Visible = false; lvFiles.Refresh(); } }
```

Оскільки в робочому потоці відбувається звертання до компонента ListView, що створений в іншому потоці, безпосереднього доступу до нього не буде. Тут і згодиться делегат, створений на самому початку. Під час виконання процедури спочатку перевіряється властивість InvokeRequired компонента ListView. Згідно з MSDN, властивість InvokeRequired дозволяє набувати значення, що показує, чи слід операторові звертатися до методу invoke під час викликів методу з елемента управління, оскільки оператор знаходиться не в тому потоці, в якому був створений елемент управління. Простіше кажучи, якщо виклик процедури відходить від робочого потоку, то виконується блок коду, що стоїть після if, в іншому випадку виконується

код після else. У цьому самому коді відбувається додавання елемента в ListView, збільшується значення лічильника прогрес бара і якщо поточний лічильник дорівнює максимальному значенню, то означає, що потік виконав свою роботу, можна приховати кнопку відміни. В кінці процедури змальовується знову компонент ListView, аби користувач побачив уже додані файли.

У робочому стані вікно додатка виглядає так, як показано на рис. 2.11.

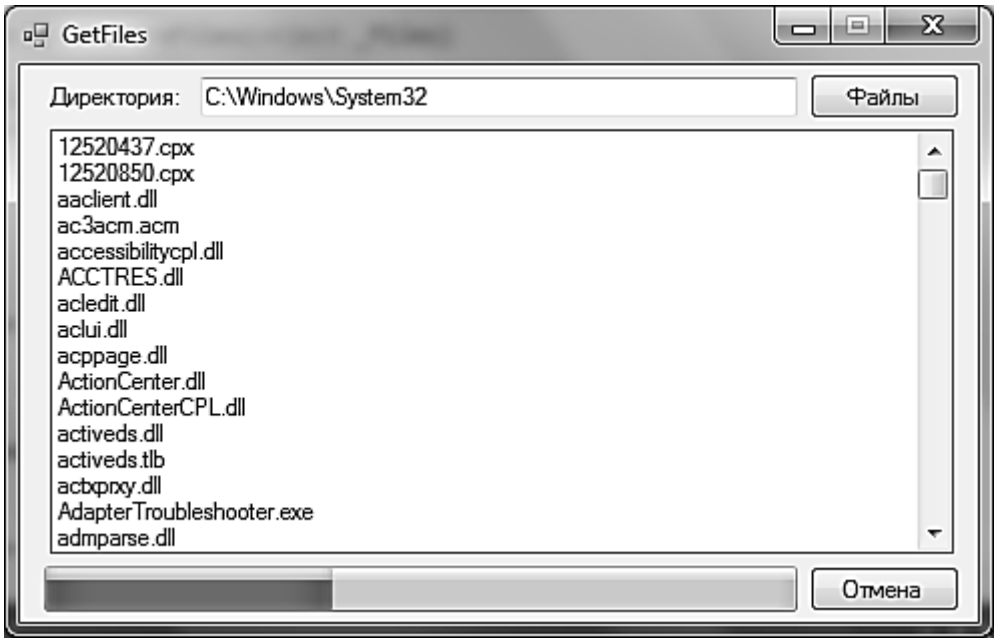


Рис. 2.11. **Вікно працюючого застосування**

Залишилося розглянути лише процедуру, що викликається кнопкою відміни.

```
private void btnCancel_Click(object sender, EventArgs e){
if (newTh.IsAlive) newTh.Abort();
btnCancel.Visible = false; }
```

Тут усе просто: якщо потік існує, то слід завершити його, викликавши його метод Abort(). Потім кнопка стає невидимою.

Також варто врахувати той момент, що, якщо закрити форму під час працюючого потоку, то саме вікно закриється, але процес залишиться в пам'яті, поки всі його потоки не завершаться. Тому в процедурі закриття вікна слід вставити код.

```
private void frmMain_FormClosing(object sender
FormClosingEventArgs e) {
if (newTh != null)
if (newTh.IsAlive)
newTh.Abort(); }
```

Усе практично так само, як і в процедурі, що викликається на подію натисненням кнопки відміни. Єдина відмінність – додається перевірка на нерівність об'єкта, що представляє потік, значенню null. Якщо цю перевірку прибрати, то у разі спроби закрити форму, коли newTh == null, виключення не виникає.

Завдання 2.1

1. Вивчити теоретичні посилання.
2. Запрограмувати і виконати налаштування консольного додатка.
3. Запрограмувати і виконати налаштування графічного додатка.
4. Провести експерименти з додатками.

Питання для самоперевірки

1. Дайте визначення поняттям "критична ділянка", "взаємовиключення" і "синхронізація процесів (потоків)".
2. Перерахуйте та охарактеризуйте основні методи синхронізації в ОС Windows.
3. Визначте критичну ділянку коду потоку в розроблювальному застосуванні.
4. Які з розглянутих методів синхронізації можуть бути використані для синхронізації процесів?
5. Що таке завдання, процес, потік, волокно?
6. Які види механізмів взаємодії процесів ви знаєте?
7. Як реалізуються процеси і потоки в ОС Windows?
8. Як плануються процеси в ОС Windows?

Лабораторна робота 3
Архітектура віртуальної пам'яті ОС Windows

Мета: здобуття практичних навичок із налаштування і використання віртуальної пам'яті ОС Windows Win32 API в процесі створення програмних проектів.

3.1. Теоретичні посилання

Об'єм основної пам'яті комп'ютера малий, і це значно обмежує можливості із запуску і використання всіх функцій додатків. Ідея віртуальної пам'яті полягає в тому, аби сформувати лінійний адресний простір, що розширює оперативну пам'ять, використовуючи зовнішню пам'ять. Таким чином, кожен процес отримав би у своє розпорядження чималий адресний простір (рис. 3.1).

Віртуальна пам'ять поділяється на частини розміром 4 Кб, звані сторінками (Page). Сторінки управляються пристроєм управління пам'яттю MMU (Memory Management Unit). Під час виконання процесу в оперативну пам'ять завантажуються лише використовувані сторінки, останні зберігаються в зовнішній пам'яті.

Адресний простір віртуальної пам'яті залежить від апаратної платформи. Так, адресний простір 32-бітових x86 систем складає 4 Гб.

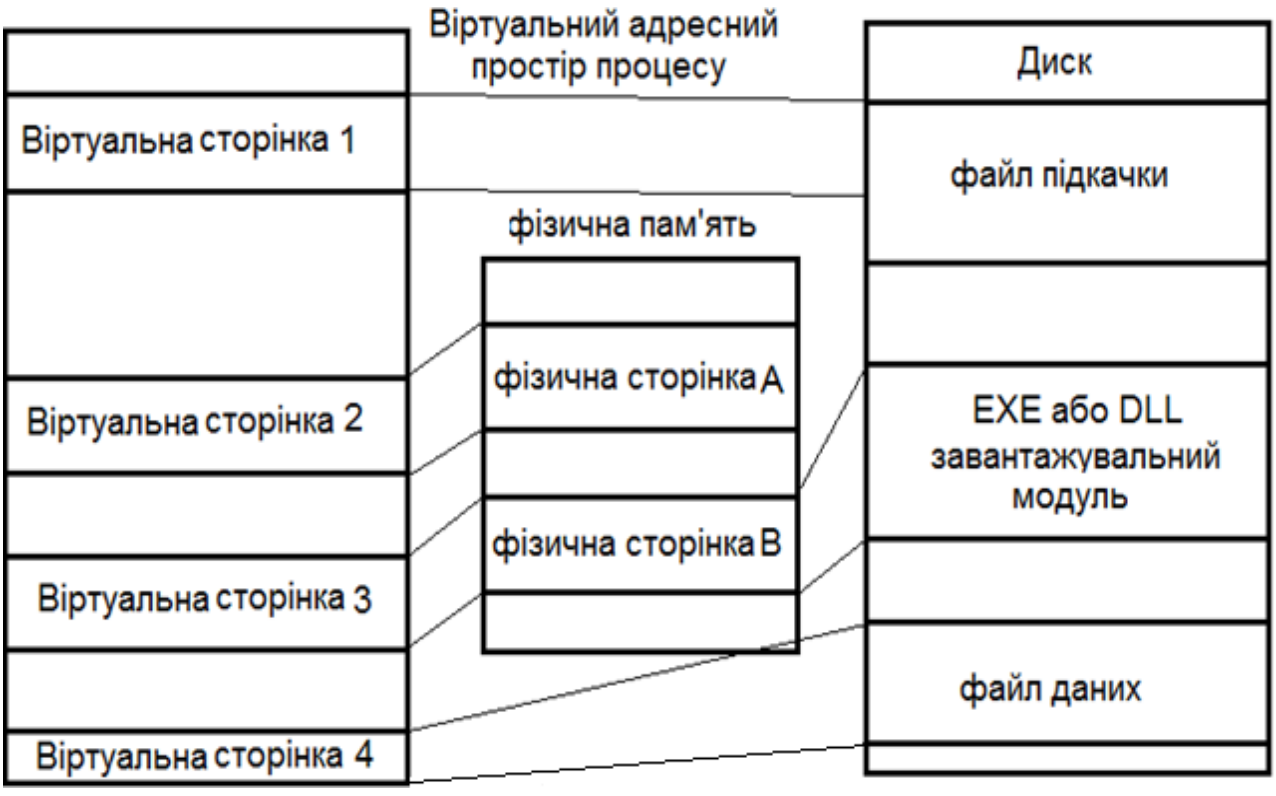


Рис. 3.1. **Структура віртуальної пам'яті**

Пристрій управління пам'яттю MMU переводить логічні адреси у фізичні. Якщо поділити 4 Гб пам'яті на сторінки розміром 4 КБ, то вийде 1 мільйон сторінок. Процесор застосовує двовимірну таблицю для звернення до цього мільйона сторінок, яку можна подати як матрицю 1024 x 1024. Перша розмірність називається каталогом сторінок (Page Directory), а друга – таблицею сторінок (Page Table). Маючи в своєму розпорядженні подібну структуру, можна створити каталог сторінок, у якому 1024 записи, і кожен запис вказує на таблицю сторінок. Кожна таблиця сторінок містить, у свою чергу, 1024 записи, кожна з яких вказує на фізичну адресу 4 Кб сторінки. Кожен запис у каталозі і таблиці сторінки має 4 байти в довжину. Так, щоб розподілити 4 Гб адресний простір на 4 Кб сторінки, потрібна структура розміром 4 x 1024 x 1024 = 4 Мб. Адреси пам'яті мають 32 біти в довжину, з яких 20 біт (10 біт на каталог сторінок + 10 біт на таблицю сторінок) є фізичною адресою сторінки, а 12 біт – індексом усередині вибраної сторінки (рис. 3.2).

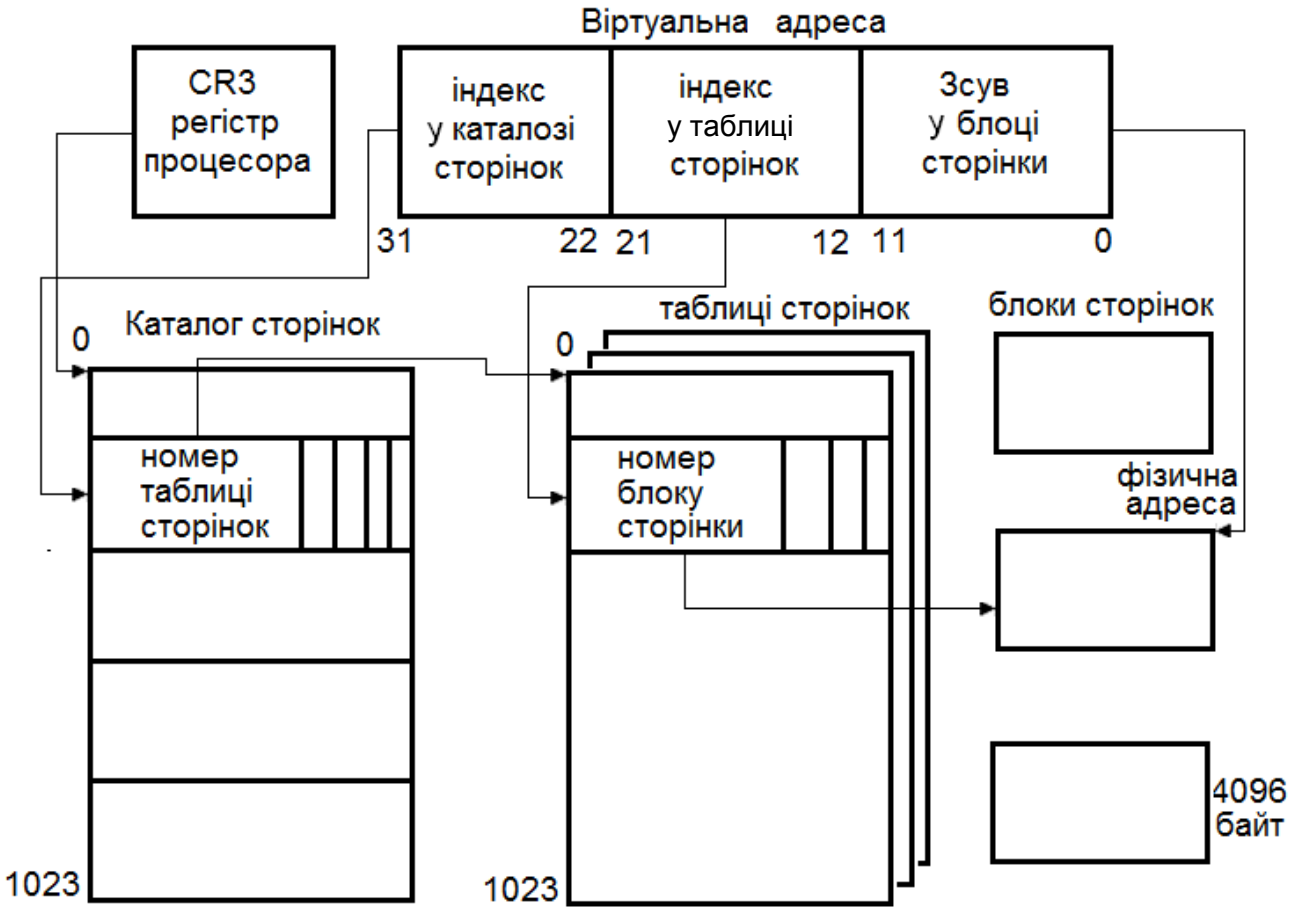


Рис. 3.2. **Принцип адресації віртуальної пам'яті**

У середовищі Windows у кожного процесу є свій каталог сторінок і таблиця сторінок, для яких ОС виділяє 4 Мб місця. Коли процес створюється, кожен запис у каталозі сторінок містить фізичну адресу таблиці сторінок. Записи в таблиці сторінок можуть бути як дійсними, так і недійсними. Дійсні записи містять фізичну адресу 4 Кб сторінок. Процес нічого не знає про фізичні адреси і під час звернення використовує лише віртуальні адреси. Перетворенням віртуальної адреси у фізичну займається менеджер пам'яті. Адреса, за якою у фізичній пам'яті знаходиться каталог сторінок, називається базовою адресою каталога сторінок (Page Directory Base Address). Ця адреса записується в спеціальний регістр процесора CR3 (у процесорах x86). У процесі перемикання контексту Windows завантажує у регістр CR3 нове значення, аби спрямувати запуск на адресу каталогу сторінок нового процесу.

Використання такої технології дозволяє процесу займати 4 Гб адресного логічного (віртуального) простору. Логічні адресні простори процесів не перетинаються, і логічна адреса одного процесу не може вказувати на фізичну адресу іншого. Це називається віртуальним адресним простором, оскільки хоч у процесу і є 4 Гб віртуального адресного простору, використовувати він може лише стільки пам'яті, скільки йому виокремлено.

Загальний об'єм пам'яті, що одночасно займається всіма процесами, не може перевищити суму основної пам'яті і файла підкочування (Pagefile). Це називається межею для виділення пам'яті (Commit Limit). Якщо процес затребує більше пам'яті (Commit), то ОС перевіряє наявність вільної пам'яті і виокремлює її процесу. Таким чином, ОС гарантує, що процесу реально виокремлена пам'ять, чи в області оперативної, чи зовнішньої пам'яті. Об'єм всієї віртуальної пам'яті, виокремленої всім процесам, називається поточною виокремленою пам'яттю (Current Commit Charge). Для визначення оптимального розміру файла підкочування необхідно знати, скільки віртуальної пам'яті процеси реально займають. Для цього потрібно знайти пік виокремленої пам'яті (Peak Commit Charge). Інструментом, за допомогою якого можна знайти пік виокремленої пам'яті, можливо, наприклад, Process Explorer:

Оперативної пам'яті комп'ютера повинно вистачити, аби в неї помістилися всі поточні блоки призначених для користувача програм.

Якщо оперативної пам'яті не вистачає, то продуктивність системи різко падає через помилки сторінок і, відповідно, постійного підкочування сторінок. Помилка сторінки (Page Fault) – це ситуація, коли процес звертається за логічною адресою, але відповідна сторінка не завантажена в оперативну пам'ять. Якщо виникає помилка сторінки, ОС запускає процедуру управління помилками сторінок, яка блокує процес, що звернувся за цією адресою, знаходить у зовнішній пам'яті необхідну сторінку і завантажує її в оперативну пам'ять. Потім процедура управління помилками оновлює запис сторінки фізичною адресою і запускає заблокований процес, аби той міг відновити свою роботу.

Якщо в оперативній пам'яті недостатньо місця для нової сторінки, ОС застосовує алгоритм заміщення сторінок, згідно з яким у пам'яті відшукується сторінка, яку в даний момент можна перенести в зовнішню пам'ять. Вибір сторінки для перенесення в зовнішню пам'ять є дуже відповідальним, оскільки, якщо буде вибрана сторінка, що скоро буде потрібною, ОС знову доведеться вирішувати проблему помилки сторінки і переносити недавно видалену сторінку назад в оперативну пам'ять. Може статися, що в системі виникає велика кількість помилок сторінок, і ОС доводиться безперервно займатися їх підкочуванням. Процеси у ході цього простоюють в очікуванні сторінок. Ця ситуація називається такою, що пробуксувала (Thrashing). Це ситуація, коли ОС витрачає велику частину часу на вирішення помилок сторінок, а процеси не можуть виконувати свої завдання. Для вирішення цієї проблеми ОС повинна скоротити кількість запущених процесів.

3.1.1. Файли, що відображуються у пам'яті

Файли, що відображуються в пам'яті, є сервісом, який Win32 надає програмістові. Його існування стирає для програміста грань між оперативною і дисковою пам'яттю. Дійсно, з точки зору класичної теорії – кеш, оперативна пам'ять і дисковий простір – це три види пам'яті, що відрізняються швидкістю доступу і розміром. Але якщо функції з переміщення даних між кешем і оперативною пам'яттю виконують процесор і ОС, то переміщення даних між оперативною пам'яттю і диском зазвичай виконує прикладний процес із використанням функцій read() і write(). Win32 діє інакше, а саме: ОС виконує функцію переміщення сторінок адресного простору процесу, що знаходяться у файлі підкочування, причому як

файл підкочування може бути використаний будь-який файл. Інакше кажучи, сторінки віртуальної пам'яті будь-якого процесу можуть бути помічені як вивантажені, а як місце, куди вони вивантажені, може бути вказаний файл. Тепер у разі звернення до такої сторінки UMM проведе її завантаження, використовуючи стандартний механізм свопінгу. Це дозволяє працювати з довільним файлом як з регіоном пам'яті. Даний механізм має в Win32 три використання:

для запуску виконуваних файлів (EXE) і динамічно зв'язуваних бібліотек (DLL);

для роботи з файлами;

для спільного використання однієї області даних двома і більш процесами.

Відображення файла даних в адресний простір процесу надає потужний механізм роботи з файлами – програма може працювати з файлом, як із масивом елементів пам'яті. За допомогою функції CreateFileMapping() створюється об'єкт ядра файл, що "відображується". У ході цього використовується дескриптор файла (handle), який повертається функцією CreateFile(). Якщо один процес змінює спільно використовувану область даних, то вона змінюється і для іншого процесу, що розділяє її. Операційна система забезпечує когерентність спільно використовуваної області даних для всіх процесів.

Завдання 3.1. Підготувати реферат за темами:

1. Створення і використання Ram Disk в ОС Windows 7.
2. Логічна організація оперативної пам'яті ОС Windows 7.
3. Налаштування віртуальної пам'яті в ОС Windows 7.
4. Оцінювання віртуальної пам'яті ОС Windows 7 за допомогою системного монітора.

3.2. Обмін даними між застосуваннями

Слід створити два консольні застосування, одне з яких посилатиме повідомлення в загальну пам'ять, а інше – прочитувати це повідомлення.

Для цього буде потрібно два класи: MemoryMappedFile – він створюватиме ділянку пам'яті, що розділяється, і MemoryMappedViewAccessor – з його допомогою буде проведено взаємодію (читання/запис) зі загальною пам'яттю.

Код застосування, що записує повідомлення в пам'ять.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.MemoryMappedFiles;
class Program {
static void Main(string[] args) {
Console.WriteLine("Введіть повідомлення :\n");
char[] message = Console.ReadLine().ToCharArray();
int size = message.Length; //Розмір введеного повідомлення
//Створення ділянки пам'яті, що розділяється
MemoryMappedFile sharedMemory =
MemoryMappedFile.CreateOrOpen("MemoryFile", size * 2 + 4);
//Створюємо об'єкт для запису в ділянку пам'яті, що розділяється
using (MemoryMappedViewAccessor writer
=sharedMemory.CreateViewAccessor(0, size * 2 + 4)) {
//запис розміру повідомлення в пам'ять, що розділяється
writer.Write(0, size);
//запис повідомлення з четвертого байта в пам'яті, що розділяється
writer.WriteArray<char>(4, message, 0, size); }
Console.WriteLine("\nПовідомлення, записане в пам'ять, що розділяється");
Console.WriteLine("Для виходу натискайте будь-яку клавішу");
Console.ReadLine(); } }
```

Код застосування, що прочитує повідомлення з пам'яті.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.MemoryMappedFiles;
class Program {
static void Main(string[] args) {
char[] message;//Масив для повідомлення із загальної пам'яті
int size; //Розмір введеного повідомлення
```

```
//Здобуття існуючої ділянки пам'яті, що розділяється
MemoryMappedFile sharedMemory =
MemoryMappedFile.OpenExisting("MemoryFile");
//Прочитуємо розмір повідомлення
using (MemoryMappedViewAccessor reader =
sharedMemory.CreateViewAccessor(0, 4,
MemoryMappedFileAccess.Read)) {
size = reader.ReadInt32(0); }
//Прочитуємо повідомлення
using (MemoryMappedViewAccessor reader =
sharedMemory.CreateViewAccessor(4, size * 2,
MemoryMappedFileAccess.Read)) {
message = new char[size];
reader.ReadArray<char>(0, message, 0, size); }
Console.WriteLine("Отримано повідомлення :\n");
Console.WriteLine(message); Console.ReadLine(); } }
```

Тепер запустити перше застосування, ввести повідомлення і відправити його в пам'ять. Потім запустити друге застосування і можна бачити повідомлення з пам'яті, яка розподіляється (рис. 3.3).

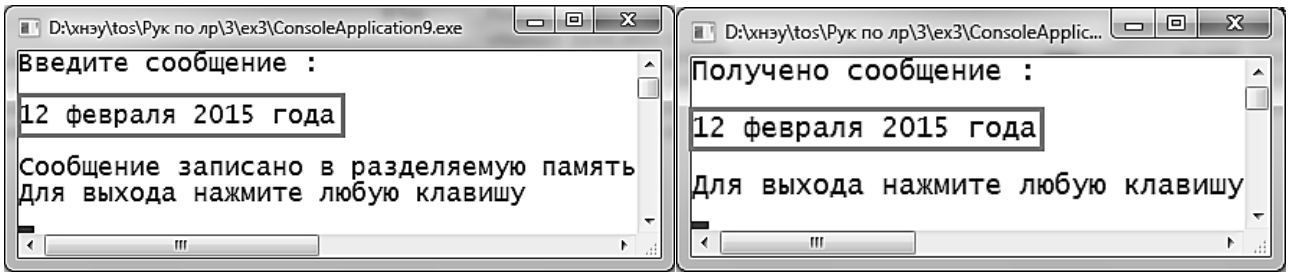


Рис. 3.3. **Обмін даними між застосуваннями через файл у пам'яті**

Завдання 3.2. Усунення помилки зміни послідовності запуску застосувань

Якщо першим запустити застосування для прочитування повідомлення з пам'яті, буде отримано помилку й аварійне завершення застосування. Потрібно, використовуючи механізм обробки виняткових ситуацій, усунути цю помилку.

3.3. Обмін даними між застосуваннями з використанням механізму обробки виняткових ситуацій і синхронізації

Даний приклад складається з трьох окремих процесів (консольних застосувань), які записують логічні значення в зіставлений у пам'яті файл. Виконується наведена далі послідовність дій.

Процес Process A створює зіставлений у пам'яті файл і записує в нього значення.

Процес Process B відкриває зіставлений у пам'яті файл і записує в нього значення.

Процес Process C відкриває зіставлений у пам'яті файл і записує в нього значення.

Процес Process A прочитує і відображує значення із зіставленого в пам'яті файла.

Після того, як процес Process A закінчить роботу із зіставленим у пам'яті файлом, цей файл негайно буде знищений.

```
using System; // Process A:
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;
class Program {
static void Main(string[] args) {
using (MemoryMappedFile mmf =
MemoryMappedFile.CreateNew("testmap", 10000)) {
bool mutexCreated;
Mutex mutex = new Mutex(true, "testmapmutex", out
mutexCreated);
using (MemoryMappedViewStream stream =
mmf.CreateViewStream()) {
BinaryWriter writer = new BinaryWriter(stream);
writer.Write(1); }
mutex.ReleaseMutex();
Console.WriteLine("Start Process B and press ENTER to
continue.");
Console.ReadLine();
Console.WriteLine("Start Process C and press ENTER to
```

```
continue.");
Console.ReadLine();
mutex.WaitOne();
using (MemoryMappedViewStream stream =
mmf.CreateViewStream()) {
BinaryReader reader = new BinaryReader(stream);
Console.WriteLine("Process A says: {0}",
reader.ReadBoolean());
Console.WriteLine("Process B says: {0}",
reader.ReadBoolean());
Console.WriteLine("Process C says: {0}",
reader.ReadBoolean()); }
mutex.ReleaseMutex(); } } }
using System; // Process B
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;
class Program {
static void Main(string[] args) {
try {
using (MemoryMappedFile mmf =
MemoryMappedFile.OpenExisting("testmap")) {
Mutex mutex = Mutex.OpenExisting("testmapmutex");
mutex.WaitOne();
using (MemoryMappedViewStream stream =
mmf.CreateViewStream(1, 0)) {
BinaryWriter writer = new BinaryWriter(stream);
writer.Write(0); }
mutex.ReleaseMutex(); } }
catch (FileNotFoundException) {
Console.WriteLine("Memory-mapped file does not exist. Run
Process A first."); } } }
using System; // Process C
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Threading;
class Program {
```

```
static void Main(string[] args) {
try {
using (MemoryMappedFile mmf =
MemoryMappedFile.OpenExisting("testmap")) {
Mutex mutex = Mutex.OpenExisting("testmapmutex");
mutex.WaitOne();
using (MemoryMappedViewStream stream =
mmf.CreateViewStream(2, 0)) {
BinaryWriter writer = new BinaryWriter(stream);
writer.Write(1); }
mutex.ReleaseMutex(); } }
catch (FileNotFoundException) {
Console.WriteLine("Memory-mapped file does not exist. Run
Process A first, then B."); } } }
```

Для реалізації цього прикладу потрібно виконати такі дії:
запустити Process A;
запустити Process B;
повернутися до Process A і натискувати клавішу ВВЕДЕННЯ;
запустити Process C;
повернутися до Process A і натискувати клавішу ВВЕДЕННЯ;
Вихідні дані Process A виглядають таким чином (рис. 3.4).

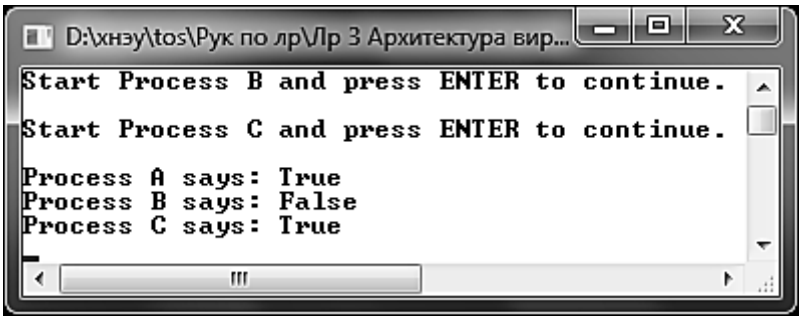


Рис. 3.4. **Вихідні дані Process A**

Завдання 3.3. Зміна напряму передачі інформації

У прикладі, що був описаний, процес A створює файл в оперативній пам'яті і прочитує з нього інформацію, передану з процесів B і C. Потрібно змінити код програм так, щоб процеси B і C прочитували інформацію з файла від процесу A.

Питання для самоконтролю

1. Чим обмежується максимальний розмір віртуального адресного простору, доступного застосуванню?

2. Що таке "віртуальна адреса", "віртуальний адресний простір"?

3. Що означає термін "плоска модель пам'яті"? У чому полягають переваги і недоліки цієї моделі?

4. Порівняйте сегментний і сторінковий способи організації ВП. Перерахуйте достоїнства і недоліки кожного.

5. Які завдання вирішує механізм VAD в ОС Windows 2000?

6. Чи може прикладний процес використовувати системну частину віртуальної пам'яті?

7. Яке з цих двох тверджень правильне:

1) усі віртуальні адреси замінюються на фізичні під час завантаження програми в оперативну пам'ять;

2) віртуальні адреси замінюються на фізичні під час виконання програми у момент звернення за даною віртуальною адресою?

8. Що таке віртуальна пам'ять?

9. Який з таких методів розподілу пам'яті може розглядатися як окремий випадок віртуальної пам'яті:

а) розподіл фіксованими розділами;

б) розподіл динамічними розділами;

в) сторінковий розподіл;

г) сегментний розподіл;

д) сегментно-сторінковий розподіл?

10. Поясніть різні значення терміна "свопінг".

11. Як величина файла підкочування впливає на продуктивність системи?

12. Чому розмір сторінки вибирається рівним мірі двійки? Чи можна прийняти таке ж обмеження для сегмента?

13. На що впливає розмір сторінки? Які переваги і недоліки великого розміру сторінки?

14. Розкажіть про підтримку сегментної організації віртуальної пам'яті в МП 80386.

15. Розкажіть про сторінковий механізм МП 80386.

16. Де зберігаються таблиці сторінок і таблиці сегментів?

Лабораторна робота 4
Підсистема введення – виведення і файлові системи

Мета: вивчення файлової системи і системи введення – виведення ОС Windows. Використання системних викликів управління файловою системою для вирішення практичних завдань.

4.1. Основи управління файловою системою ОС Windows

Файл – це іменована послідовність байтів довільної довжини. Створення файла полягає в привласненні йому імені і реєстрації його у файловій системі. Ім'я файла складається з двох частин: імені і розширення (тип файла), відокремлених через крапку.

Теки (каталоги) – це важливі елементи ієрархічної структури, необхідні для забезпечення зручного доступу до файлів, якщо файлів на носієві надто багато.

Файлова система – це частина операційної системи, що управляє розміщенням і доступом до файлів текам на диску.

Операції з файловою структурою:

навігація файловою структурою;

запуск програм і відкриття документів;

створення тек;

копіювання файлів і тек;

переміщення файлів і тек;

видалення файлів і тек;

перейменування файлів і тек;

створення ярликів.

Каталог – це ім'я групи файлів, об'єднаних за якою-небудь ознакою і зберігаються на одному диску. У каталозі містяться імена всіх файлів, що належать до нього. У системах Windows каталог називається текою, яка є точнішим поняттям, що розкриває його сенс.

Класи роботи з файловою системою знаходяться в просторі імен System.IO (збірка mscorlib.dll). Простір імен System.IO містить класи для роботи з каталогами і дисками, такі, як: DriveInfo, Directory, DirectoryInfo. Класи для роботи з файлами: File, FileInfo. Клас для роботи з шляхами: Path. Окрім цього System.IO містить основні класи для роботи з потоками Stream, FileStream, StreamReader, StreamWriter, StringReader, StringWriter, TextReader, TextWriter, BinaryReader, BinaryWriter і MemoryStream.

Для використання всіх перерахованих класів, у проект необхідно імпортувати простір імен System.IO, щоб не писати повний шлях до кожного класу using System.IO.

4.1.1. Шляхи

Клас Path дозволяє формувати шляхи до каталогів, файлів і має методи для аналізу імен файлів.

Найчастіше використовується функція Combine, яка дозволяє об'єднати фрагменти шляхів в один шлях. Усе це показано в такому прикладі (рис. 4.1).

```
string a = @"C:\Папка"; string b = "файл.exe";
string p = Path.Combine(a, b);
MessageBox.Show(String.Format("Фрагмент 1: {0}\r\nФрагмент 2: {1}\r\nРезультат об'єднання фрагментів: {2}", a, b, p));
```

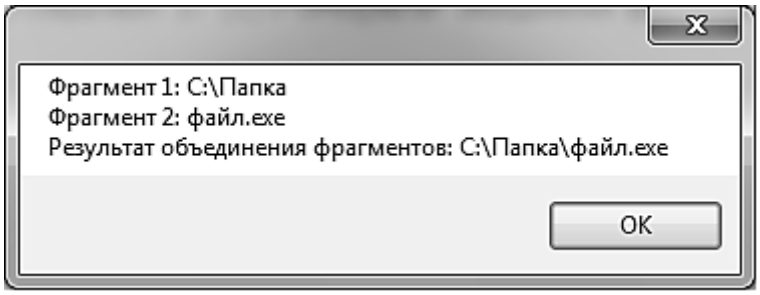


Рис. 4.1. **Результат об'єднання фрагментів шляхів в один**

У .NET Framework 3.5 функція Combine може об'єднати лише два фрагменти шляху, а в .NET Framework 4 кількість фрагментів уже нічим не обмежується.

```
string a = @"C:\Папка"; string b = "Мамочка"; string з = "Дитя.wav";
string p = Path.Combine(a, b, з);
MessageBox.Show(String.Format("Фрагмент 1: {0}\r\nФрагмент 2: {1}\r\nФрагмент 3: {2}\r\nРезультат об'єднання фрагментів: {3}", a, b, з, p));
```

У ранніх версіях .NET Framework такої функції взагалі не було і програмістам доводилося вирішувати проблеми об'єднання шляхів самостійно (рис. 4.2).

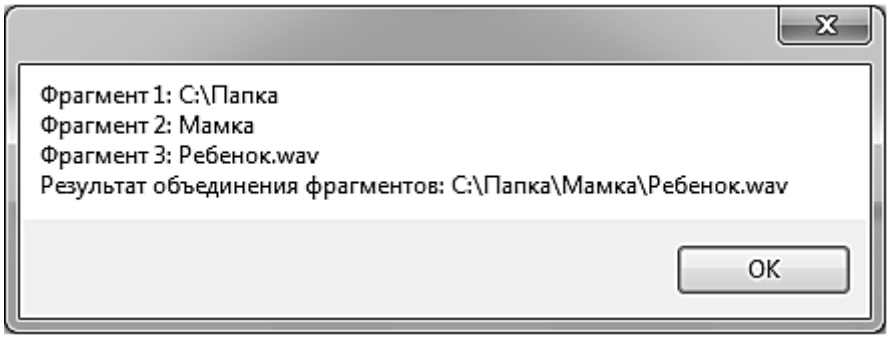


Рис. 4.2. **Результат об'єднання фрагментів шляхів в один за допомогою функції Combine класу Path**

Окрім цього клас Path має функції для відділення каталогів від файлів у вказаному шляху. Наприклад, наявний шлях "C:\Каталог 1\Каталог 2\Каталог 3\Файл.exe" і потрібно з цього шляху отримати каталог, в якому знаходиться "Файл.exe", тобто "C:\Каталог 1\Каталог 2\Каталог 3". Зробити це можна за допомогою функції GetDirectoryName.

```
string a = @"C:\Каталог 1\Каталог 2\Каталог 3\Файл.exe";
string p = Path.GetDirectoryName(a);
MessageBox.Show(p);
```

А якщо необхідно отримати лише ім'я файла – "Файл.exe", то в цьому допоможе функція GetFileName.

```
string a = @"C:\Каталог 1\Каталог 2\Каталог 3\Файл.exe";
string p = Path.GetFileName(a);
MessageBox.Show(p);
```

Функція GetFileName повертає останній елемент указаного шляху, яка зовсім не зобов'язана бути файлом. Отже, якщо необхідно отримати ім'я останнього каталога шляху, для цього теж можна використовувати функцію GetFileName.

```
string a = @"C:\Каталог 1\Каталог 2\Каталог 3\Файл.exe";
string b = Path.GetDirectoryName(a); // отримуємо шлях
string p = Path.GetFileName(b); // ім'я останнього каталога в шляху
MessageBox.Show(p); // Каталог 3
```

Аналогічно відбувається і з функцією GetDirectoryName, яка фактично повертає шлях, просто відсікаючи останній елемент.

Під час аналізу імен файлів можна піти ще далі й отримати окремо ім'я файла і розширення за допомогою функцій GetExtensionи, відповідно, GetFileNameWithoutExtension.

4.1.2. Диски

Для роботи з дисками призначений клас DriveInfo. Цей клас має всього один статичний метод – GetDrives, який повертає масив дисків комп'ютера. Кожен елемент масиву є екземпляром класу DriveInfo.

```
StringBuilder driveList = new StringBuilder();
foreach (DriveInfo d in DriveInfo.GetDrives()) {
if (d.IsReady) driveList.AppendLine(String.Format("Диск: {0}; мітка тому: {1}; файлова система: {2}; тип: {3}; об'єм: {4} байт; вільно: {5} байт", d.Name, d.VolumeLabel, d.DriveFormat, d.DriveType, d.TotalSize, d.AvailableFreeSpace));}
MessageBox.Show(driveList.ToString());
```

Примітка. Клас StringBuilder належить простору імен System.Text, у разі необхідності, слід імпортувати його в проект (рис. 4.3).

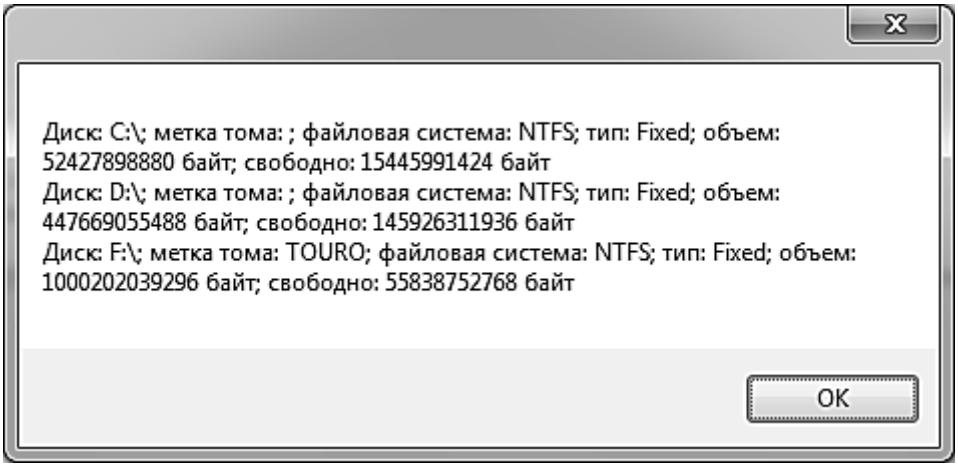


Рис. 4.3. **Виведення списку дисків**

Кожен екземпляр пристрою DriveInfo дозволяє отримати додаткову інформацію про нього. Є можливість створити екземпляр класу для будь-якого доступного диска в системі.

```
DriveInfo d = new DriveInfo("C:");
```

Перш ніж починати працювати з яким-небудь диском, необхідно перевірити його доступність. Для цього призначена властивість IsReady.

```
DriveInfo d = new DriveInfo("C:");
if (d.IsReady) {MessageBox.Show("Диск готовий до роботи!");}
else {MessageBox.Show("Диск недоступний!");}
```

У системі є мережеві диски, здобуття інформації про них може зайняти значний час, упродовж якого програма може бути недоступна для користувача (підвисатиме).

Якщо пристрій не готовий до роботи (наприклад, є CD-ROM, але в ньому немає диска), під час спроби отримати детальну інформацію про диск відбудеться вимкнення (помилка).

Клас DriveInfo дуже простий. Букву диска можна отримати у властивості Name. Властивість VolumeLabel містить мітку тому. Отримати інформацію про файлову систему можна у властивості DriveFormat. Фактичний об'єм диска і вільний простір, що залишився, знаходяться у властивостях TotalSize і AvailableFreeSpace. Тип пристрою можна дізнатися у властивості DriveType.

Букву диска (Name) і його тип (DriveType) можна отримати, навіть якщо диск не готовий до роботи (рис. 4.4).

```
StringBuilder driveList = new StringBuilder();
foreach (DriveInfo d in DriveInfo.GetDrives()) {
driveList.AppendLine(String.Format("Диск: {0}; тип: {1}; ", d.Name,
d.DriveType));
}
MessageBox.Show(driveList.ToString());
```

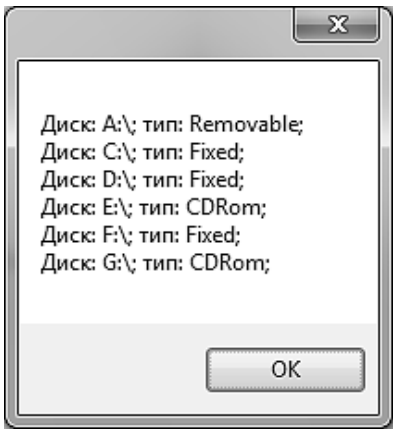


Рис. 4.4. **Список типів пристроїв**

4.1.3. Каталоги

Робота з каталогами в .NET Framework здійснюється за допомогою двох класів. Основний клас – DirectoryInfo, він не має статичних методів і призначений для роботи з конкретними каталогами. Другий клас – Directory, є допоміжним і містить часто використовувані статичні методи, функціонал яких, по суті, реалізований через клас DirectoryInfo.

Клас DirectoryInfo під час ініціалізації обирає шлях до каталога, який він реалізує. Причому можна вказувати шлях як до реально існуючого каталога, так і до каталога, який ще не створений.

```
DirectoryInfo d = new DirectoryInfo("D:\\Нова тека");
```

Перевірити існування каталога можна за допомогою властивості Exists.

```
DirectoryInfo d = new DirectoryInfo("D:\\Нова тека");
if (d.Exists) {MessageBox.Show("Каталог існує!");}
else {MessageBox.Show("Такого каталогу ще немає! ");}
```

Якщо каталога немає, то його можна створити за допомогою методу Create.

```
d.Create();
```

Примітно, що метод Create може створювати не лише одну теку, але і всі відсутні теки, вказані в шляху. Наприклад, якщо шлях "C:\Тека 1\Тека 2\Тека 3" і жоден з цих каталогів не існує, то будуть створена всі три теки.

У існуючу директорію можна запросто додати нові теки методом CreateSubdirectory.

```
DirectoryInfo d = new DirectoryInfo(@"D:\Нова тека");
d.CreateSubdirectory("Тека");
d.CreateSubdirectory("Тека1");
d.CreateSubdirectory("Тека2");
```

Інколи виникає необхідність отримати повний шлях до каталога. Для цього можна використовувати властивість FullName. Аби отримати лише ім'я поточного каталога, досить поглянути у властивість Name. Окрім цього, екземпляр каталога містить посилання на батьківську директорію у властивості Parent, а також посилання на кінцеву теку – Root. Посилання є екземплярами класу DirectoryInfo, що дає можливість маніпулювати цими каталогами (рис. 4.5).

```
DirectoryInfo d = new DirectoryInfo(@"D:\Новая папка\Просто папка");
MessageBox.Show(String.Format("Полный путь: {0}\r\nТекущий каталог: {1}\r\nРодитель: {2}\r\nКорневой каталог: {3}", d.FullName, d.Name, d.Parent.Name, d.Root.Name));
```

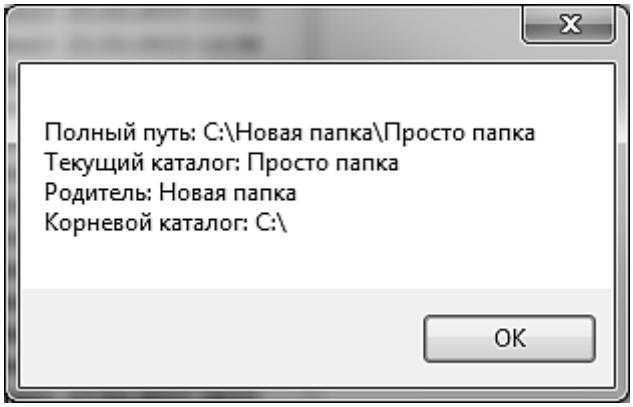


Рис. 4.5. **Виведення інформації про поточний, батьківський і кореневий каталоги**

Звичайного методу для зміни імені каталога в класі DirectoryInfo немає. Проте директорію можна перемістити в інше місце методом MoveTo. Цей метод, окрім переміщення каталога, цілком можна використовувати для зміни імені теки.

```
DirectoryInfo d = new DirectoryInfo(@"D:\Нова тека");
d.MoveTo(@"D:\Стара тека");
```

Для видалення каталога призначений метод Delete. Та варто пам'ятати, що директорію, в якій знаходяться вкладені теки і файли, просто так видалити не можливо. Видалити можна лише абсолютно порожній каталог, інакше станеться вимкнення (помилка). У .NET

Framework 4.0 метод Delete має додатковий параметр, який дозволяє рекурсивно видалити всі вкладені в теку об'єкти.

```
DirectoryInfo d = new DirectoryInfo(@"D:\Нова тека");
d.Delete(true);
```

Отримати список вкладених каталогів і файлів можна за допомогою функцій GetDirectories і GetFiles. Функція GetDirectories повертає масив DirectoryInfo, кожен елемент якого реалізує вкладену теку. Аналогічно функція GetFiles повертає масив файлів, де кожен елемент є екземпляром класу FileInfo. Обидві функції можуть виводити як усі файли/каталоги, так і проводити вибірку певних об'єктів за шаблоном. Наприклад, можна вивести список файлів із розширенням .cmd, що знаходяться в теці "C:\Windows\System32\", вказавши шаблон пошуку "*.cmd" (рис. 4.6).

```
DirectoryInfo d = new DirectoryInfo(@"C:\Windows\System32");
StringBuilder fileList = new StringBuilder();
foreach (FileInfo f in d.GetFiles("*.cmd"))
{
fileList.AppendLine(f.Name);
}
MessageBox.Show(fileList.ToString());
```

Рис. 4.6. **Список файлів з розширенням .cmd, C:\Windows\System32, що знаходяться в теці**

Окрім цього, клас DirectoryInfo містить властивості для визначення дати створення каталога – CreationTime, дати останнього доступу до каталога – LastAccessTime і дату останньої модифікації каталога – LastWriteTime.

Отримати список дозволу для директорії можна за допомогою функції GetAccessControl, яка повертає об'єкт типу DirectorySecurity (рис. 4.7).

```
DirectoryInfo d = new DirectoryInfo(@"D:\Нова тека");
StringBuilder result = new StringBuilder();
DirectorySecurity ds = d.GetAccessControl();
foreach (FileSystemAccessRule permissions in
ds.GetAccessRules(true, true, typeof(NTAccount))) {
result.AppendLine(String.Format("Користувач: {0}",
permissions.IdentityReference.Translate(typeof(NTAccount)).Value));
result.AppendLine(String.Format("Права: {0}",
permissions.FileSystemRights.ToString())); result.AppendLine();}
MessageBox.Show(result.ToString());
```

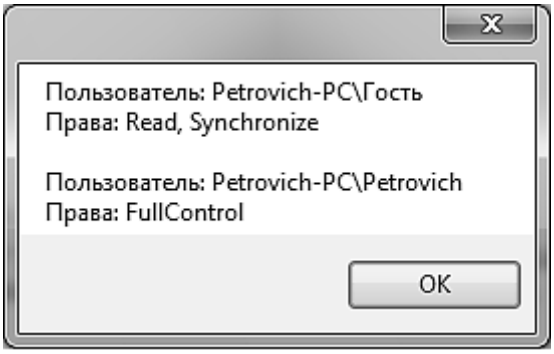


Рис. 4.7. **Виведення списку дозволів для каталога**

Що стосується допоміжного класу Directory, то він підходить для виконання простих операцій над директоріями, хоча в принципі має все ті ж можливості, що і клас DirectoryInfo. Клас Directory зручно використовувати для перевірки існування каталога, якщо більше жодних дій над цим каталогом проводити не потрібно.

```
if (Directory.Exists(@"D:\Нова тека")) MessageBox.Show("Каталог
існує!");
```

За допомогою цього класу можна легко створити нову теку методом CreateDirectory, перемістити методом Move або видалити – Delete.

```
Directory.CreateDirectory(@"D:\Нова тека")
Directory.Move(@"D:\Нова тека" @"D:\Стара тека");
Directory.Delete(@"D:\Стара тека", true);
```

Якщо ж потрібно виконувати безліч операцій над певним каталогом, то краще використовувати для цього клас DirectoryInfo, оскільки кожна функція класу Directory виконує всі операції саме створюючи екземпляри DirectoryInfo. Уявіть, скільки разів ініціалізується клас DirectoryInfo, якщо постійно застосовувати методи Directory для одного і того ж каталога. Може статись марна трата ресурсів, хоча звичайно, в більшості випадків очевидних негативних наслідків для продуктивності від цього не буде.

Завдання 4.1

Вибрати свій варіант завдання. Уточнити його у викладача. Написати консольне застосування в середовищі Microsoft Visual Studio і прокоментувати фізичний сенс усіх системних викликів управління файловою системою, використовуваних у програмі.

Варіанти завдань

1. Дано два фрагменти шляху. Написати програму об'єднання фрагментів шляху в один.
2. Задано шлях до файла. Написати програму реалізації шляху до каталога, виключаючи ім'я файла.
3. Задано шлях до файла. Написати програму реалізації імені файла або останнього каталога із указаного шляху.
4. Задано ім'я файла. Написати програму реалізації імені файла без розширення і розширення файла без імені.
5. Задано ім'я файла. Написати програму зміни розширення файла.
6. Задано ім'я. Написати програму визначення, чи є вказане ім'я текою або файлом.
7. Написати програму визначення шляху до тимчасового каталога Windows.
8. Написати програму здобуття списку всіх дисків ЕОМ.
9. Написати програму перевірки існування директорії.
10. Написати програму створення нової директорії.
11. Написати програму обмеження права доступу до директорії.

12. Написати програму визначення прав доступу до директорії.

13. Написати програму зміни імені теки.

14. Написати програму видалення теки і видалення теки і всіх підпапок, а так само всіх вкладених файлів.

15. Написати програму здобуття списку вкладених у теку каталогів і файлів.

16. Написати програму перевірки існування файла.

4.2. Система введення – виведення ОС Windows

4.2.1. Основи файлового введення – виведення

Передача даних із зовнішнього пристрою в оперативну пам'ять називається читанням, або введенням, зворотний процес – записом, або виводом.

Уведення – виведення виконується за допомогою підсистеми введення – виведення і класів бібліотеки .NET.

Потік (stream) – це абстрактне поняття, що належить до будь-якого перенесення даних від джерела до приймача. Потоки забезпечують надійну роботу як зі стандартними, так і з визначеними користувачем типами даних, а також одноманітний і зрозумілий синтаксис.

Потік визначається як послідовність байтів і не залежить від конкретного пристрою, з яким проводиться обмін (оперативна пам'ять, файл на диску, клавіатура або принтер). Обмін із потоком для підвищення швидкості передачі даних проводиться, як правило, через спеціальну область оперативної пам'яті – буфер. Буфер виділяється для кожного відкритого файла. Під час запису у файл вся інформація спочатку прямує в буфер і там накопичується до того часу, поки весь буфер не заповниться. Лише після цього або після спеціальної команди скидання відбувається передача даних на зовнішній пристрій. У прцесі читання з файла дані спочатку прочитуються в буфер, причому не стільки, скільки запрошується, а скільки поміщається в буфер.

Механізм буферизації дозволяє швидше й ефективніше обмінюватися інформацією із зовнішніми пристроями.

Для підтримки потоків бібліотека .NET містить ієрархію класів, основна частина якої наведена на рис. 4.8. Ці класи визначені в просторі імен System.IO.

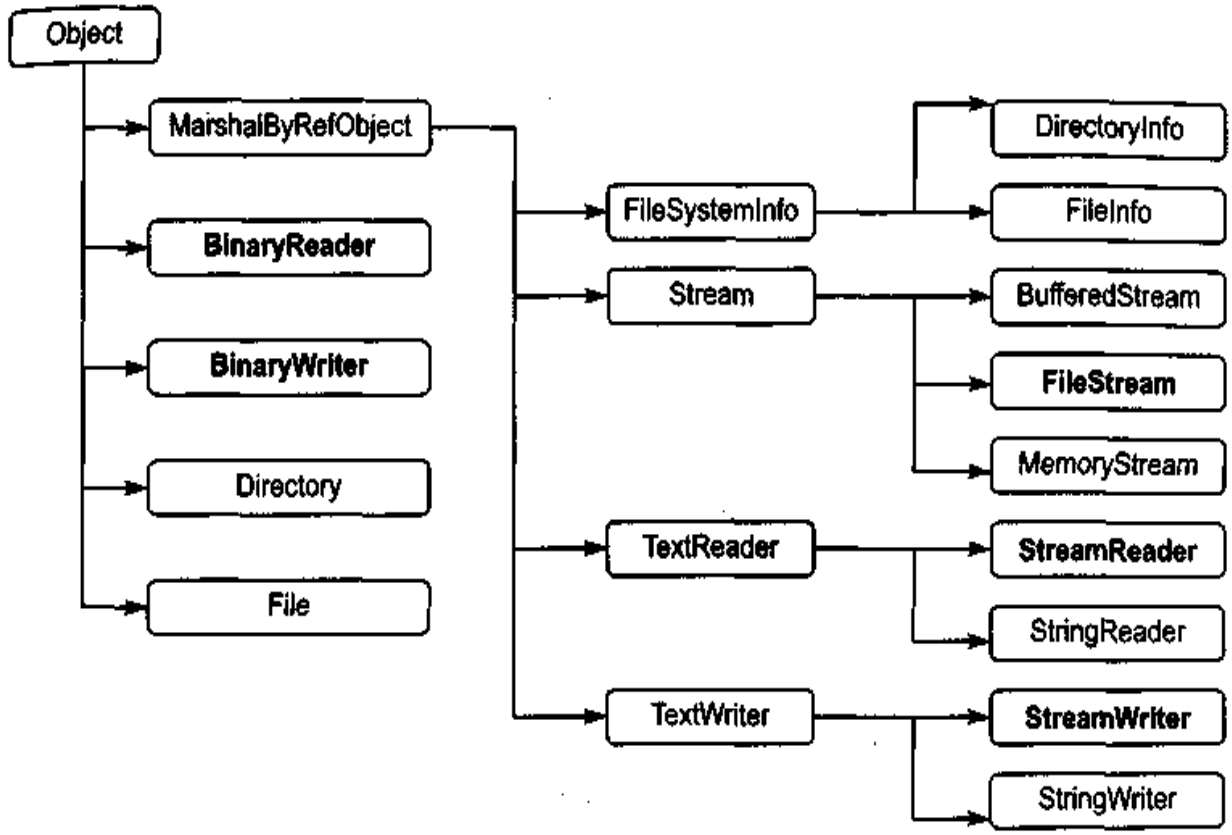


Рис. 4.8. **Класи бібліотеки .NET для роботи з потоками**

Класи бібліотеки дозволяють працювати в різних режимах із файлами, каталогами й областями оперативної пам'яті. Короткий опис класів наведений в табл. 4.1.

Таблиця 4.1

Основні класи простору імен System.IO

Клас	Опис
1	2
BinaryReader, BinaryWriter	Читання і запис значень простих вбудованих типів (цілочисельних, логічних, строкових тощо) у внутрішній формі подання
BufferedStream	Тимчасове зберігання потоку байтів (наприклад, для подальшого перенесення в постійне сховище)
Directory, DirectoryInfo File, FileInfo	Робота з каталогами або фізичними файлами: створення, видалення, набуття властивостей. Можливості класів File і Directory реалізовані в основному у вигляді статичних методів. Аналогічні класи DirectoryInfo і FileInfo використовують звичайні методи
FileStream	Довільний (прямий) доступ до файла, поданого як потік байтів

Закінчення табл. 4.1

1	2
MemoryStream	Довільний доступ до потоку байтів в оперативній пам'яті
StreamWriter, StreamReader	Читання з файлу і запис у файл текстової інформації (довільний доступ не підтримується)
StringWriter, StringReader	Робота з текстовою інформацією в оперативній пам'яті

Як можна бачити з табл. 4.1, виконувати обмін із зовнішніми пристроями можна на рівні:

двійкового подання даних (BinaryReader, BinaryWriter);

байтів (FlleStream);

тексту, тобто символів (StreamWriter, StreamReader).

У .NET використовується кодування Unicode, в якому кожен символ кодується двома байтами. Класи, що працюють з текстом, є оболонками класів, що використовують байти, й автоматично виконують те, що перекодувало з байтів у символи і назад.

Двійкові і байтові потоки зберігають дані в тому ж вигляді, в якому вони подані в оперативній пам'яті, тобто під час обміні з файлом відбувається побітове копіювання інформації. Двійкові файли застосовуються не для перегляду їх людиною, а для використання в програмах.

Доступ до файлів може бути послідовним, коли черговий елемент можна прочитати (записати) лише після аналогічної операції з попереднім елементом, і довільним, або прямим, за якого виконується читання (запис) довільного елемента за заданою адресою.

Текстові файли дозволяють виконувати лише послідовний доступ, в двійкових і байтових потоках можна використовувати обидва методи.

Прямий доступ у поєднанні з відсутністю перетворень забезпечує високу швидкість здобуття потрібної інформації.

Методи форматованого введення, за допомогою яких можна виконувати введення з клавіатури або з текстового файла значень арифметичних типів, у C# не підтримуються. Для перетворення з символьного в числову виставу використовуються методи класу Convert або метод Parse.

Форматоване виведення, тобто перетворення з внутрішньої форми подання числа в символьну, зрозумілу людині, виконується за допомогою переобтяжених методів ToString, результати виконання яких передаються в методи текстових файлів.

Окрім перерахованих класів у бібліотеці .NET є класи XmlTextReader.

Використання класів файлових потоків у програмі передбачає такі операції:

створення потоку і скріплення його з фізичним файлом;

обмін (введення – виведення);

закриття файла.

Кожен клас файлових потоків містить декілька варіантів конструкторів, за допомогою яких можна створювати об'єкти цих класів різними способами і в різних режимах.

Наприклад, файли можна відкривати лише для читання, лише для запису або для читання і запису. Ці режими доступу до файла містяться в перерахуванні FileAccess, визначеному в просторі імен System.IO. Константи перерахування наведені в табл. 4.2.

Таблиця 4.2

Значення перерахування FileAccess

Значення	Опис
Read	Відкрити файл лише для читання
ReadWrite	Відкрити файл для читання і запису
Write	Відкрити файл лише для запису

Можливі режими відкриття файла визначені в перерахуванні FileMode (табл. 4.3).

Таблиця 4.3

Значення перерахування FileMode

Значення	Опис
Append	Відкрити файл, якщо він існує, і встановити поточний покажчик у кінець файла. Якщо файл не існує, то створити новий файл
Creat	Створити новий файл. Якщо в каталозі вже існує файл з таким же ім'ям, він буде стертий
CreateNew	Створити новий файл. Якщо в каталозі вже існує файл з таким же ім'ям, виникає виключення IOException
Open	Відкрити існуючий файл
Truncate	Відкрити існуючий файл. Після відкриття він має бути обрізаний до нульової довжини

Режим FileMode.Append можна використовувати лише спільно з доступом типа FileAccess.Write, тобто для файлів, що відкриваються для запису.

Режими спільного використання файла різними користувачами визначає перерахування FileShare (табл. 4.4).

Таблиця 4.4

Значення перерахування FileShare

Значення	Опис
None	Спільне використання відкритого файла заборонене. Запит на відкриття даного файла завершується повідомленням про помилку
Read	Дозволяє відкривати файл для читання одночасно декільком користувачам. Якщо цей прапор не встановлений, запити на відкриття файла для читання завершуються повідомленням про помилку
ReadWrite	Дозволяє відкривати файл для читання і запису одночасно декільком користувачам
Write	Дозволяє відкривати файл для запису одночасно декільком користувачам

4.2.2. Потоки байтів

Уведення – виведення у файл на рівні байтів виконується за допомогою класу FileStream, який є спадкоємцем абстрактного класу Stream, що визначає набір стандартних операцій з потоками. Елементи класу Stream описані в табл. 4.5.

Таблиця 4.5

Елементи класу Stream

Елемент	Опис
1	2
BeginRead, BeginWrite	Почати асинхронне введення або виведення
CanRead, CanSeek, CanWrite	Властивості, що визначають, які операції підтримує потік: читання, прямий доступ і запис
Close	Закрити поточний потік і звільнити пов'язані з ним ресурси (сокети, покажчики на файли і т. п.)
EndRead, EndWrite	Чекати завершення асинхронного введення; закінчити асинхронний виведення
Flush	Записати дані з буфера в пов'язане з потоком джерело даних і очистити буфер. Якщо для даного потоку буфер не використовується, то цей метод нічого не робить

Закінчення табл. 4.5

1	2
Length	Повернути довжину потоку в байтах
Position	Повернути поточну позицію в потоці
Read, ReadByte	Рахувати послідовність байтів (або один байт) з поточного потоку і перемістити покажчик у потоці на кількість перерахованих байтів
Seek	Установити поточний покажчик потоку на задану позицію
SetLength	Установити довжину поточного потоку
Write, WriteByte	Записати послідовність байтів (або один байт) у поточний потік і перемістити покажчик у потоці на кількість записаних байтів

Клас FileStream реалізує ці елементи для роботи з дисковими файлами. Для визначення режимів роботи з файлом використовуються стандартні перерахування FileMode, FileAccess і FileShare. Значення цих перерахувань наведені в табл. 4.2 – 4.4. Приклад роботи з потоком байтів наведений у розділі 2.1.

4.2.3. Потоки символів

Символьні потоки StreamWriter і StreamReader працюють з Unicode-символами, отже, ними найзручніше користуватися для роботи з файлами, призначеними для сприйняття людиною. Ці потоки є спадкоємцями класів TextWriter і TextReader відповідно, які забезпечують їх переважно функціональності. У табл. 4.6 і 4.7 наведені найбільш важливі елементи цих класів. Довільний доступ для текстових файлів не підтримується.

Таблиця 4.6

Найбільш важливі елементи базового класу TextWriter

Елемент	Опис
Close	Закрити файл і звільнити пов'язані з ним ресурси. Якщо в процесі запису використовується буфер, він буде автоматично очищений
Flush	Очистити всі буфери для поточного файла і записати накопичені в них дані в місце їх постійного зберігання. Сам файл у ході цього не закривається
NewLine	Використовується для завдання послідовності символів, що означають початок нового рядка. За замовчуванням використовується послідовність "повернення каретки" – "переклад рядка" (\r\n)
Write	Записати фрагмент тексту в потік
WriteLine	Записати рядок в потік і перейти на інший рядок

Таблиця 4.7

Найбільш важливі елементи класу TextReader

Елемент	Опис
Peek	Повернути наступний символ, не змінюючи позицію покажчика у файлі
Read	Рахувати дані з вхідного потоку
ReadBl ock	Рахувати з вхідного потоку вказану користувачем кількість символів і записати їх у буфер, починаючи із заданої позиції
ReadLine	Зчитати рядок з поточного потоку і повернути його як значення типа string. Порожній рядок null означає кінець файла (EOF)
ReadToEnd	Рахувати всі символи до кінця потоку, починаючи з поточної позиції, і повернути перераховані дані як один рядок типу string

4.2.4. Асинхронне введення – виведення

Клас Stream (і, відповідно, FileStream) підтримує два способи виконання операцій введення – виведення: синхронний і асинхронний. За замовчуванням файли відкриваються в синхронному режимі, тобто подальші оператори виконуються лише після завершення операцій введення – виведення.

Для тривалих файлових операцій ефективніше виконувати введення – виведення асинхронно, в окремому потоці виконання. У ході цього в первинному потоці можна виконувати інші операції.

Для асинхронного введення – виведення необхідно відкрити файл в асинхронному режимі, для цього використовується відповідний варіант переобтяженого конструктора. Асинхронна операція введення ініціюється за допомогою методу BeginRead. Окрім характеристик буфера, в який виконується введення, в цей метод передається делегат, задаючий метод, що виконується після завершення введення.

Цей метод може ініціювати обробку отриманої інформації, відновити операцію читання або виконати будь-які інші дії, наприклад, перевірити успішність введення і повідомити про його завершення. Зазвичай в цьому методі викликається метод EndRead, який завершує асинхронну операцію.

Аналогічно виконується і асинхронний вивід. У розділі наведений приклад асинхронного читання з файла великого об'єму і паралельного виконання діалогу з користувачем.

4.2.5. Двійкові потоки

Двійкові файли зберігають дані в тому ж вигляді, в якому вони подані в оперативній пам'яті, тобто у внутрішній формі подання. Двійкові файли застосовуються не для перегляду їх людиною, а для використання в програмах.

Вихідний потік BinaryWriter підтримує довільний доступ, тобто є можливість виконувати запис у довільну позицію двійкового файла. Двійковий файл відкривається на основі базового потоку, як який найчастіше використовується потік FileStream. Вхідний двійковий потік містить перевантажені методи читання для всіх простих убудованих типів даних.

Основні методи двійкових потоків наведені в таблицях 4.8 і 4.9.

Таблиця 4.8

Найбільш важливі елементи класу BinaryWriter

Елемент	Опис
BaseStream	Базовий потік, з яким працює об'єкт BinaryWriter
Close	Закрити потік
Flush	Очистити буфер
Seek	Встановити позицію в поточному потоці
Wri te	Записати значення в поточний потік

Таблиця 4.9

Найбільш важливі елементи класу BinaryReader

Елемент	Опис
BaseStream	Базовий потік, із яким працює об'єкт BinaryReader
Close	Закрити потік
PeekChar	Повернути наступний символ без переміщення внутрішнього покажчика в потоці
Read	Рахувати потік байтів або символів і зберегти в масиві, передаваному як вхідний параметр
READXXXX	Рахувати з потоку дані певного типу (наприклад, ReadBoolean, ReadByte, ReadInt32 і т. д.)

4.2.6. Методи класу Directory

Клас Directory має такі методи (всі вони є статичними і доступні без створення об'єкта):

CreateDirectory() – створює всі вкладені теки у вказаному як параметр шляху;

Delete () – видаляє вказану теку;

Exists () – дозволяє визначити, чи існує вказана як параметр тека;

GetAccessControl () – повертає права доступу для вказаної теки;

GetCreationTime () – повертає час створення вказаної теки;

GetCurrentDirectory() – повертає поточну робочу теку застосування, в яку зберігатимуться файли, для яких не вказаний конкретний шлях;

GetDirectoryRoot () – повертає інформацію про том;

GetFiles() – повертає імена файлів у вказаній теці;

GetFileSystemEntries() – повертає всі імена файлів і вкладених тек у вказаній теці;

GetLastAccessTime () – час останнього доступу до теки;

GetLastWriteTime () – час останнього запису в теці;

GetLogicaiDrives () – повертає імена логічних дисків у системі;

Getparent () – повертає батьківську теку;

Move () – переміщає файл або директорію зі всім вмістом в нове місце;

SetAccessControl () – призначити права доступу;

SetCreationTime () – змінити час створення;

SetCurrentDirectory() – змінити поточну теку;

SetLastAccessTime() – змінити час останнього доступу;

SetLastwriteTime() – змінити час останнього запису.

4.2.7. Методи класу File

Клас File має такі методи, які також є статичними і доступні без створення об'єкта:

AppendAllText () – додати вказаний текст у файл. Якщо файл не існує, то він буде створений;

AppendText() – додати текст у файл, що поверне об'єкт StreamWriter, який можна використовувати для дописування в той же файл додаткової інформації;

Create () – створити файл за вказаним шляхом;

Createtext () – створити файл для запису інформації у форматі UTF-8;

Decrypt () – дешифрувати вміст файла;

Delete () – видалити вказаний файл;

Encrypt () – зашифрувати вказаний файл;

Exists () – перевірити існування вказаного файла;

GetAccessControl () – повернути права доступу до файла;

GetAttributes () – повернути атрибути файла;

GetCreationTime () – час створення вказаного файла;

GetLastAccessTime () – час останнього доступу до файла;

GetLastWriteTime () – час останнього запису у файл;

Move () - перемістити файл в нову теку;

Open () – відкрити для читання і зміни вказаний файл;

OpenRead() – відкрити вказаний файл для читання;

OpenWrite () – відкрити вказаний файл для запису;

OpenText () – відкрити файл для роботи із ним як з текстом у форматі UTF-8;

ReadAllLines () – прочитати всі рядки файла;

SetAccessControl () – призначити права доступу;

SetCreationTime () – змінити час створення;

SetLastAccessTime() – змінити час останнього доступу;

SetLastwriteTime () – змінити час останнього запису;

WriteAllLines() – створити файл і записати в нього вказаний масив рядків.

4.2.8. Робота з потоком байтів

Слід розглянути роботу з файлами в C# на прикладі застосування, наведеного на рис. 4.9.

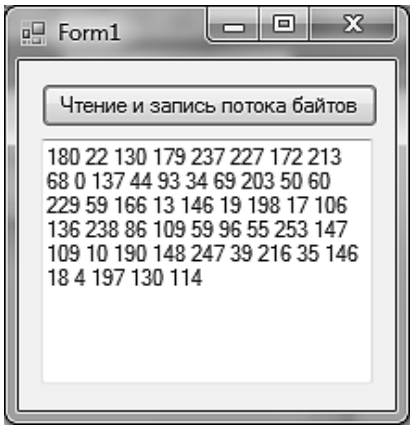


Рис. 4.9. **Читання і запис потоку байтів**

Для події Click кнопки "Читання-запис потоку байтів" написати такий програмний код:

```
private void button1_Click_1(object sender, EventArgs e){
textBox1.Clear();
```

```
FileStream f = new FileStream("file1.txt", FileMode.Create,
FileAccess.ReadWrite); //Створення файла
Random r = new Random();
byte a;
// Запис у файл п'ятдесяти байт
for (int i = 0; i < 50; ++i) { a =(byte) r.Next(); f.WriteByte(a); }
// встановити покажчик на початок файла
f.Seek(0, SeekOrigin.Begin);
// отримати довжину файла в байтах
long j =f.Length;
// читання і виведення на екран вмісту файла
Do { a = (byte)f.ReadByte();
textBox1.Text += Convert.ToString(a) + " ";
j = j - 1;
} while (j!=0);
f.Close(); }
```

Поточна позиція в потоці спочатку встановлюється на початок файла (для будь-якого режиму відкриття, окрім Append) і зрушується на одну позицію під час запису кожного байта.

Для установки бажаної позиції читання використовується метод Seek, що має два параметри: перший задає зсув у байтах відносно точки відліку, що задається другим. Точки відліку задаються константами перерахування SeekOrigin: початок файла – Begin, поточна позиція – Current і кінець файла – End.

У даному прикладі файл створювався в поточному каталозі. Можна вказати і повний шлях до файла, при цьому зручніше використовувати дослівні літерали, наприклад:

```
FileStream f = new FileStream(@"D:\lr5\test.txt", FileMode.Create,
FileAccess.ReadWrite);
```

У дослівних літералах не потрібно дублювати обернену косу риску.

Операції з відкриття файлів можуть завершитися невдало, наприклад, у разі помилкі в імені існуючого файла або за відсутності вільного місця на диску, тому рекомендується завжди контролювати результати цих операцій.

У разі непередбачених ситуацій середовище виконання генерує різні виключення, обробку яких слід передбачити в програмі, наприклад:

FileNotFoundException, якщо файла зі вказаним ім'ям у вказаному каталозі не існує;

DirectoryNotFoundException, якщо не існує вказаний каталог;

ArgumentException, якщо неправильно заданий режим відкриття файла;

IOException, якщо файл не відкривається через помилки введення – виведення.

Можливі й інші виняткові ситуації.

Зручно обробляти найбільш вірогідні помилки окремо, аби надати користувачеві програми в повідомленні, що виводиться, найбільш точну інформацію. У даному прикладі помилки не обробляються.

Під час закриття файла звільняються всі пов'язані з ним ресурси, наприклад, для файла, відкритого для запису, у файл вивантажується вміст буфера. Тому рекомендується завжди закривати файли після закінчення роботи, особливо файли, відкриті для запису. Якщо буфер потрібно вивантажити, не закриваючи файл, використовується метод Flush.

4.2.9. Робота з потоком символів

Вікно для роботи з потоком символів наведене на рис. 4.10.

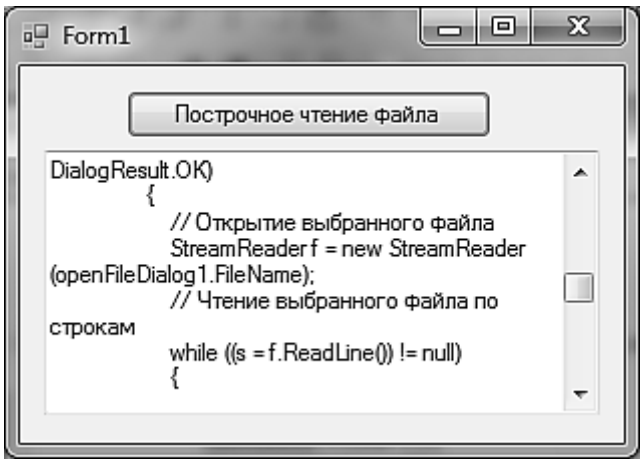


Рис. 4.10. **Прогресивне читання файла**

Для події Click кнопки "Відрядкове читання текстового файла" написати такий програмний код:

```
private void button1_Click(object sender, EventArgs e){
String s;
```

```
textBox1.Clear();
try {
// Діалог вибору файла
if (openFileDialog1.ShowDialog() == DialogResult.OK) {
// Відкриття вибраного файла
StreamReader f = new
StreamReader(openFileDialog1.FileName);
// Читання вибраного файла по рядках
while ((s = f.ReadLine()) != null) {
textBox1.Text += s + (char)13 + (char)10; }
f.Close(); } }
catch (FileNotFoundException e1) {
textBox1.Text += e1.Message + (char)13 + (char)10;
textBox1.Text += "Перевірте правильність імені файла";
return; }
catch (Exception e1) {
textBox1.Text += "Помилка:" + e1.Message; } }
```

4.2.10. Робота з двійковим потоком

Вікно для роботи з потоком символів наведене на рис. 4.11.

Для події Click кнопки "Формування двійкового файла" написати такий програмний код:

```
private void button1_Click(object sender, EventArgs e){
try {
BinaryWriter f = new BinaryWriter(new
FileStream(@"D:\binary.file", FileMode.Create));
double d = 0;
while (d < 20) {
f.Write(d); d += 0.33; };
f.Seek(16, SeekOrigin.Begin);
f.Write(8888d); f.Close(); }
catch (Exception e1) {
textBox1.Text += "Помилка:" + e1.Message;
return; } }
```

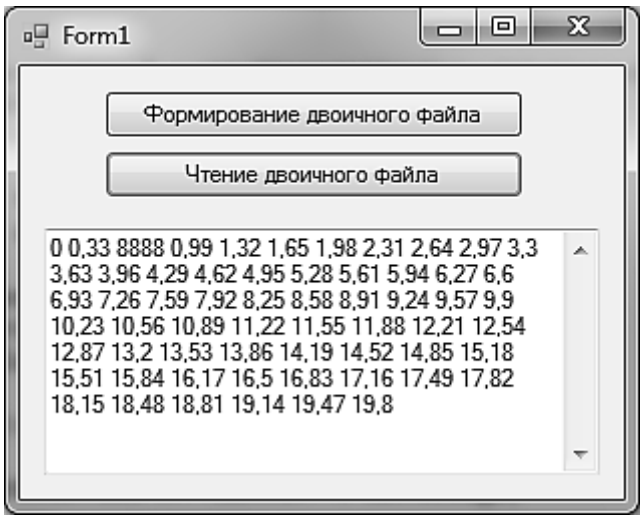


Рис. 4.11. **Формування і читання двійкового файла**

Під час створення двійкового потоку в нього передається об'єкт базового потоку. У процесі установки покажчика поточної позиції у файлі враховується довжина кожного значення типу double – 8 байт.

Для події Click кнопки "Прочитування двійкового файла" написати такий програмний код:

```
private void button2_Click(object sender, EventArgs e) {
try {
FileStream f =new FileStream(@"D:\binary.file",FileMode.Open);
BinaryReader f1 = new BinaryReader(f);
long n = f.Length / 8; // кількість чисел у файлі
double[] x = new double[n];long i = 0;
try { while (true) x[i++]= f1.ReadDouble(); }
catch (EndOfStreamException e2) { }
foreach (double d in x) textBox1.Text += Convert.ToString(d) + " ";
f1.Close(); f.Close(); }
catch (FileNotFoundException e1) {
textBox1.Text += e1.Message + (char)13 + (char)10;
textBox1.Text += "Перевірте правильність імені файла";
return; }
catch (Exception e1) {
textBox2.Text += "Помилка:" + e1.Message; } }
```

Спроба перегляду сформованого програмою файла в текстовому редакторові не інформативна, тому в програмі допомогою екземпляра BinaryReader прочитує вміст файла в масив дійсних чисел, а потім виводить цей масив на екран. Під час читання береться до уваги той факт, що метод ReadDouble у ході виявлення кінця файла генерує виключення EndOfStreamException. Оскільки в даному випадку це не помилка, тіло обробника виключень порожнє.

Завдання 4.2

1. Сформувати файл послідовності 15 чисел, в якій кожен і-й компонент визначається за формулою

y = sin(i * p / 8) якщо i < 8; y = 4 cos(i (p + 1) / 5) якщо i > 8.

Визначити кількість позитивних значень, що містяться в сформованому файлі.

Скласти програму, що визначає правильність дотримання дужок у рядку символів, використовуючи для цієї мети стік на основі файла.

2. Сформувати файл послідовності 20 чисел, у якій кожен і-й компонент визначається за формулою

y = sin(i * p / 8) якщо i < 8; y = 4cos(i (p + 1) / 5) якщо i > 8.

Визначити кількість негативних значень, що містяться у сформованому файлі.

Підрахувати кількість здвоєних символів 'її', 'нн', 'лл' у тексті, розташованому в текстовому файлі.

3. Сформувати файл із значень випадкових величин:

0,324; 0,524; 0,789; 0,556; 0,761; 0,248; 0,345; 0,911; 0,216,

Визначити для даної послідовності середнє арифметичне компонентів, значення яких менше 0,5.

Розподілити довільний текст, що знаходиться у файлі, на рядки певної довжини. Під час перенесення слова передбачити виведення дефіса.

4. Відкоректувати текст, розташований у текстовому файлі, замінивши в ньому всі входження однієї букви на іншу.

Провести сортування файла цілих чисел методом "бульбашки".

5. Сформувати файл, що містить прізвища декількох студентів. Додати до отриманого файла прізвища ще 2 – 3 студентів.

Знайти в текстовому файлі довге та коротке слова.

6. Записати у файл оцінки (у балах), отримані певним студентом на іспитах протягом усіх сесій. Додати в початок файла оцінки, отримані на вступних іспитах.

Із рядка, розташованого в текстовому файлі, виключити всі символи, що входять у нього більше одного разу.

7. Записати у файл оцінки (у балах), отримані певним студентом на іспитах протягом усіх сесій, і визначити середній бал.

Перевірити, чи правильно розставлені в тексті, розташованому в текстовому файлі, круглі дужки.

8. Сформувати два файли. У один із них помістити прізвища п'яти ваших знайомих, а в іншій – номери їх телефонів. Скласти програму, яка за прізвищем вашого знайомого визначає номер його телефону.

У послідовності символів, заданій в текстовому файлі, підрахувати загальну кількість символів '+', '-', '*'.

9. Сформувати два файли. У один із них помістити прізвища п'яти ваших знайомих, а в іншій – номери їх телефонів. Скласти програму, яка за номером телефону вашого знайомого визначає його прізвище.

Слова тексту, розташованого в текстовому файлі, вивести на екран у вигляді рядка й у вигляді стовпчика.

10. У текстовому файлі, в пропозиції, що містить не менше двох слів, поміняти місцями перше і останнє слова.

Для програми, записаної у файлі у вигляді безперервного тексту, перетворити файл так, щоб кожен оператор розташовувався на окремому рядку.

11. У текстовому файлі два рядки тексту. Необхідно сформувати третій рядок, що складається з символів, що входять одночасно в обидва вихідні рядки, і дописати його в текстовий файл.

У стовпцях матриці довільного розміру, розміщеної в зовнішньому файлі, провести перестановку його елементів так, щоб максимальний елемент кожного стовпця виявився на головній діагоналі.

12. Сформувати файл із значень випадкових величин:

0,324; 0,524; 0,789; 0,556; 0,761; 0,248; 0,345; 0,911; 0,216,

Визначити для даної послідовності суму компонентів, значення яких більше 0,5.

Підрахувати число слів у пропозиції, записаній у текстовому файлі.

13. Переписати текстовий файл так, щоб всі слова вихідного тексту були перевернуті.

Перемножити два найдовші цілі числа, записані у файлі. Результат записати в той же файл.

14. У довільний текстовий файл додати у кінець перше і третє слова з вихідного тексту.

У відсортованому файлі цілих чисел методом бінарного пошуку знайти заданий елемент і видалити його.

15. Сформувати файл, компонентами якого є дійсні значення, що обчислюються за формулою:

$$a_i = (i + 1)\ 2 \sin(i\ p / 10),$$

де i – номер компонента файла.

Визначити, скільки в отриманому файлі міститься позитивних значень.

У довільний текстовий файл додати в кінець своє прізвище.

16. Сформувати файл, компонентами якого є дійсні значення, що обчислюються за формулою:

$$a_i = (i + 1)\ 2 \sin(i\ p / 10),$$

де i – номер компонента файла.

Визначити, скільки в отриманому файлі міститься негативних значень.

У вихідному текстовому файлі X замінити всі входження підрядка P на підрядок Q.

17. Сформувати файл цілих чисел. Вивести на екран лише ті компоненти файла, значення яких лежать в інтервалі від 0 до 25.

Для заданого символу визначити, скільки разів він зустрічається у введеному тексті файла.

18. Сформувати файл цілих чисел. Вивести на екран лише парні значення компонентів файла.

Із тексту, розташованого у файлі, виключити групи символів, розташованих між круглими дужками.

19. Із тексту, розташованого у файлі, виключити однобуквенні слова. Результат записати в інший файл.

Нехай деякий файл містить числа Фібоначчі U_0, U_1..., U_n. Отримати і доповнити цей файл черговим компонентом U_{n+1}.

20. Сформувати файл, елементами якого є значення функції $y = \sin(x_i) + 2\cos(x_i)$ в точках X = (0,1; 0,2; 0,25; 0,33; 1,78; 2,05; 2,23). Визначити компонент файла, що має мінімальне значення.

Із тексту, розташованого у файлі, видалити зайві пропуски, що розділяють слова.

21. Сформувати файл, елементами якого є значення функції $y = \sin(x_i) + 2\cos(x_i)$ у точках X = (0,1; 0,2; 0,25; 0,33; 1,78; 2,05; 2,23). Визначити компонент файла, що має максимальне значення.

З'ясувати, чи правильно, що серед символів рядка довільної довжини, розташованої у файлі, є всі символи, що входять у слово ДЕНЬ.

22. Сформувати файл цілих чисел. Вивести на екран лише непарні значення компонентів файла.

Для кожного зі слів пропозиції, розташованої в одному текстовому файлі, вказати, скільки разів воно зустрічається в пропозиції, розташованій в іншому текстовому файлі.

23. Записати у файл оцінки (у балах), отримані студентами групи за певним предметом, визначити середній бал.

У довільному текстовому файлі виключити перше і останнє слова.

24. Записати у файл оцінки (у балах), отримані студентами групи за певним предметом, визначити відсоток успішності.

Визначити, чи можна з символів заданого у файлі рядка скласти ваше прізвище.

25. Із довільної послідовності символів, розташованої в текстовому файлі, виключити спеціальні символи.

У зовнішньому файлі створити чергу довільної довжини. Видаляти або доповнювати її довільною кількістю елементів.

26. У файлі цілих чисел виключити повторне входження одних і тих же чисел.

Даний файл цілих чисел. Перетворити його так, щоб спочатку йшли всі від'ємні числа, а потім – додатні.

Питання для самоперевірки

1. Який об'єкт вибраний як зберігання інформації в ЕОМ?
2. Із яких частин складається ім'я файла?
3. Як розрізняються файли залежно від розширення?
4. У чому полягає унікальність імені файла?
5. Чим утворена файлова структура?
6. Як позначаються імена зовнішніх носіїв інформації?
7. Що таке повне ім'я файла? Наведіть приклад.
8. Дайте характеристику фізичної структури зберігання інформації.
9. Що таке файлова система?
10. Які основні файлові системи виділяють і в чому їх відмінність?
11. Чому створення підсистеми введення – виведення вважається однією з найскладніших областей проектування операційних систем?

12. Чому операції введення – виведення в операційних системах оголошуються привілейованими?

13. Перерахуйте основні завдання, що покладаються на супервізор введення – виведення?

14. У яких випадках пристрій введення – виведення називається ініціативним?

15. Які режими управління введенням – виведенням ви знаєте? Опишіть кожен із них.

16. Що означає термін spooling і що означає термін swapping?

17. Чим забезпечується незалежність призначених для користувача програм від пристроїв введення – виведення, підключених до ком-п'ютера?

18. Що таке синхронне й асинхронне введення – виведення?

19. Опишіть структуру магнітного диска (розбиття дисків на розділи). Наскільки до (і яких) розділів може бути на магнітному диску?

20. Як у загальному випадку здійснюється завантаження операційної системи після включення комп'ютера? Що таке початковий, системний і позасистемний завантажувачі? Де вони розташовуються?

21. Розкажіть про кешування операцій введення – виведення під час роботи з накопичувачами на магнітних дисках.

Змістовий модуль 2
Налаштування і забезпечення безпеки функціонування ОС

Лабораторна робота 5
Робота з реєстром ОС Windows 7

Мета: вивчити призначення реєстру, його структуру, редактори реєстру, прийоми відновлення системи у разі пошкодження реєстру; сформувати навички й уміння працювати з редактором реєстру.

5.1. Реєстр операційної системи

Реєстр Windows (Windows Registry, системний реєстр) – ієрархічно побудована база даних параметрів і налаштувань у більшості ОС

Windows. Реєстр містить інформацію і налаштування для апаратного забезпечення, програмного забезпечення, профілів користувачів, передустановки.

Системний реєстр Windows 7 побудований за принципами динамічності, ієрархічності і захищеності.

Динамічність полягає у можливості зміни в процесі роботи з ОС.

Ієрархічність виражається в її деревовидній структурі, що дозволяє швидко відшукати потрібні файли.

Захищеність побудована так, що реєстр має власну службу безпеці і не дозволяє доступ першому зустрічному користувачеві.

Відмінністю реєстру Windows 7 від попередніх версій є те, що в нього вносяться лише найважливіші записи й об'єм реєстру не впливає на швидкодію ОС.

Файли реєстру створюються в процесі установки операційної системи і зберігаються в теці %SystemRoot%\system32\config. Для операційних систем Windows – це файли з іменами default, sam, security, software і system.

У ОС Windows Windows 7 файли реєстру розташовуються також у каталозі \Windows\system32\config і мають такі ж імена, проте в цих ОС додався новий розділ реєстру для зберігання даних конфігурації завантаження (Boot Configuration Data) з ім'ям BCD00000000. Файл із даними цього розділу має ім'я bcd і знаходиться в прихованій теці Boot активного розділу (розділу, з якого виконується завантаження системи). Зазвичай, під час стандартної установки Windows 7, створюється активний розділ невеликого розміру (близько 100 мегабайт), який прихований від користувача і містить лише службові дані для завантаження системи, – завантажувальні записи, менеджер завантаження bootmgr, сховище конфігурації завантаження BCD, файли локалізації і програми тестування пам'яті. Розташування куща bcd залежить від того, як конфігурований завантажувач системи під час її установки, і може знаходитися на тому ж розділі, де і каталог Windows.

Місце розташування файлів реєстру в будь-якій версії Windows можна проглянути за допомогою редактора реєстру. До складу ОС Windows 7 входить програма для редагування реєстру regedit.exe (рис. 5.1).

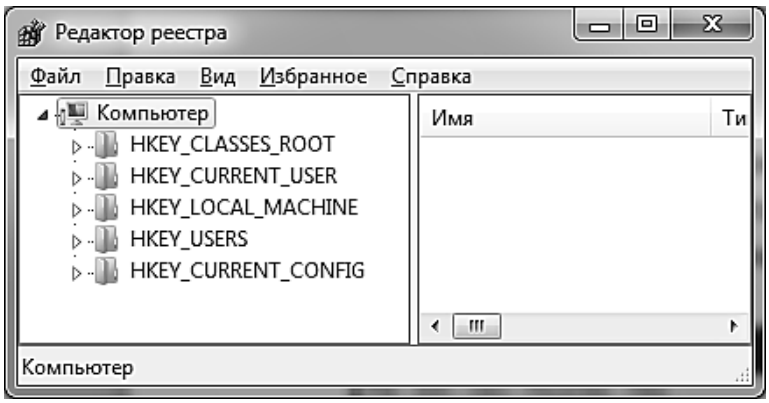


Рис. 5.1. **Редактор реєстру ОС**

Реєстр має ієрархічну структуру, що складається з 5 кореневих ключів:

HKEY_CLASSES_ROOT – інформація про асоціації файлів із застосуваннями, ярликами, об'єктами;

HKEY_CURRENT_USER – інформація налаштувань для зареєстрованого в системі користувача;

HKEY_LOCAL_MACHINE – інформація про встановлене апаратне програмне забезпечення;

HKEY_USERS – інформація про конфігурації призначених для користувача профілів, системні змінні середовища, розкладку клавіатури і тому подібне;

HKEY_CURRENT_CONFIG – поточна конфігурація програмного і апаратного забезпечення.

У лівій частині вікна редактора реєстру у вигляді дерева відображуються піддерева і розділи реєстру, в правій – параметри конкретного розділу реєстру. Повний шлях до вибраного розділу виводиться в рядку стану (рис. 5.2).

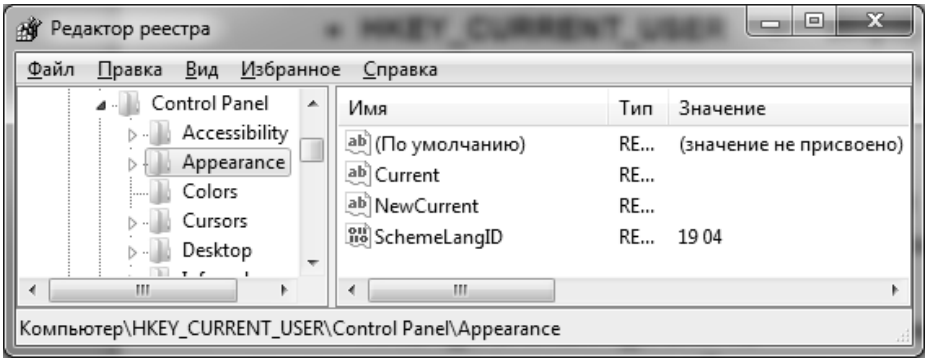


Рис. 5.2. **Структура реєстру**

Редактори реєстру дозволяють переглядати і модифікувати реєстр. Треба враховувати, що редактори реєстру не розпізнають помилки, не попереджають про них користувача і не мають команди відмінити.

5.2. Налаштування операційної системи

Налаштування ОС у більшій частині може бути виконане за допомогою редактора реєстру. Існує безліч програм для редагування реєстру. Проте, не дивлячись на те, що всі програми, які дозволяють здійснювати налаштування реєстру, прості у використанні і володіють обширними можливостями, все те, що вміють ці утиліти, можна зробити і вручну. До того, як почати роботу з реєстром, рекомендується зробити його резервну копію, створити точку відкоту в Windows 7 або створити образ диска з ОС. Маючи резервну копію, можна відновити вихідні значення всіх ключів реєстру. Якщо заздалегідь не зберегти реєстр, то єдиний вихід із ситуації, що склалася, це переустановлення ОС.

Робота з редактором реєстру зводиться до виконання таких завдань:

управління розділами реєстру;

управління параметрами;

експорт і імпорт даних.

Аби створити новий підрозділ, слід знайти у дереві реєстру потрібний розділ, виділити його і в меню "Правка" вибрати "Создать > Раздел". Ввести ім'я нового розділу.

Аби видалити розділ, слід знайти його в дереві реєстру, виділити і в меню "Правка" вибрати "Удалить". Після додаткового підтвердження розділ, включаючи всі підрозділи і параметри у всіх підрозділах будуть видалені.

Під час видалення розділів реєстру видаляються всі його підрозділи і параметри у всіх підрозділах. Операція видалення не може бути скасована.

5.3. Управління параметрами і їх значеннями

Редактор реєстру regedit.exe підтримує роботу з різними типами параметрів (табл. 5.1).

Таблиця 5.1

Типи параметрів

Тип даних	Опис
1	2
REG_BINARY	Необроблені двійкові дані. Більшість відомостей про апаратні компоненти зберігаються у вигляді двійкових даних і виводяться в редакторові реєстру в шістнадцятковому форматі

Закінчення табл. 5.1

1	2
REG_DWORD	Дані, наведені цілим числом (4 байти). Багато параметрів служб і драйверів пристроїв мають цього типу і відображуються в двійковому, шістнадцятковому або десятковому форматах
REG_EXPAND_SZ	Рядок даних змінної довжини. Цей тип даних включає змінні, що обробляються під час використання даних програмою або службою
REG_MULTI_SZ	Багаторядковий текст. Цей тип, як правило, мають списки й інші записи у форматі, зручному для читання. Записи розділяються пропусками, комами або іншими символами
REG_SZ	Текстовий рядок фіксованої довжини
REG_FULL_RESOURCE_DESCRIPTOR	Послідовність вкладених масивів, розроблена для зберігання списку ресурсів апаратного компонента або драйвера

Аби створити новий параметр у розділі, слід знайти в дереві реєстру потрібний розділ, виділити його і натиснути праву кнопку мишки. У контекстному меню вибрати "Создать", а в ньому – "Строковый параметр", "Двоичный параметр", "Параметр DWORD", "Мульти строковый параметр" або "Расширяемый строковый параметр", залежно від того, параметр якого типу необхідно створити. Буде створений новий параметр з ім'ям "Новый параметр #1", яке можна відразу змінити. Після натиснення клавіші Enter параметр буде збережений з новим ім'ям.

Аби змінити ім'я параметра, треба знайти в дереві реєстру потрібний розділ, виділити його і знайти потрібний параметр у списку справа. Виділити цей параметр і натиснути праву кнопку мишки. У контекстному меню вибрати "Переименовать". Замінити ім'я параметра також можна, натиснувши клавішу F2. Тепер можна змінити ім'я параметра безпосередньо в списку параметрів. Після зміни натиснути клавішу Enter, і нова назва параметра буде збережена.

Аби видалити параметр, треба знайти у дереві реєстру потрібний розділ, виділити його і знайти потрібний параметр у списку справа. Виділити цей параметр і натиснути праву кнопку мишки. У контекстному меню вибрати "Удалить". Крім того, можна використовувати клавішу Delete . Після додаткового підтвердження параметр буде видалений.

Аби змінити значення параметра, треба знайти в дереві реєстру потрібний розділ, виділити його і знайти потрібний параметр у списку справа. Виділити цей параметр і натиснути праву кнопку мишки. У контекстному меню вибрати "Изменить". Крім того, можна використовувати клавішу Enter. У вікні, що з'явилося, виправити або ввести нове значення параметра і клацнути кнопку ОК.

5.4. Копія реєстру

Резервну копію можна зберегти в потрібному місці, наприклад, у теці на жорсткому диску або на переносному носієві. Потім цю резервну копію можна імпортувати, аби відмінити внесені зміни.

Відкрити редактор реєстру. Для цього натиснути кнопку "Пуск", ввести regedit у поле пошуку і потім натиснути клавішу ВВЕДЕННЯ. Якщо відображується запит на введення паролю адміністратора або його підтвердження, то вказати пароль або надати підтвердження.

Слід знайти і обрати розділ або підрозділ реєстру, резервну копію якого необхідно створити.

Відкрити вкладку "Файл" і натиснути кнопку "Експорт".

У полі "Папка" обрати розташування, в яке слід зберегти резервну копію, і в полі "Имя файла" ввести ім'я файла.

Клацнути "Сохранить".

Завдання 5.1

1. Вивчення довідки WINDOWS про те, що таке реєстр, "Восстановление реестра".

2. Виконання експорту реєстру в текстовий файл на свій розділ диска. Ім'я файла – MYREG.REG.

3. Додавання повідомлення, що відображується під час реєстрації користувача в системі:

викликати редактор regedit;

розкрити ключ реєстру HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\WinLogon;

знайти параметр LegalNoticeCaption, розкрити його і ввести "Заголовок окна";

знайти параметр LegalNoticeText, розкрити його і ввести "Вас приветствует администратор";

закрити вікно редактора і перезавантажити систему, продемонструвати результат роботи;

виконати кроки 1 – 5, очистивши значення LegalNoticeCaption і LegalNoticeText;

проаналізувати зміни.

4. Зміна значка сміттєвої корзини (порожньою і заповненою):

знайти у реєстрі ключ;

HKEY_CURRENT_USER\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\CLSID\645FF040-5081-101B-9F08-00AA002F954E;

Замінити значення елементів FULL і EMPTY ключа Defaulticon з 31 і 32 на 64 і 65 відповідно;

створити на робочому столі нову теку і видалити її і поглянути, чи змінилася піктограма корзини;

продемонструвати викладачеві результат зміни і повернути колишні піктограми для корзини.

5. Видалення стрілок з ярликів:

створити на робочому столі 2 будь-яких ярлика і переконатися, що на ярличках є маленькі стрілки;

викликати редактор реєстру;

знайти ключ HKEY_CLASSES_ROOT\lnkfile;

записати тип параметра IsShortcut у зошит (для подальшого відновлення), видалити цей параметр;

знайти ключ HKEY_CLASSES_ROOT\piffile;

видалити параметр IsShortcut;

перезавантажити Windows і переконатися, що стрілки в ярличків відсутні;

продемонструвати викладачеві результат роботи;

повернути колишні установки (параметри IsShortcut).

6. Зміна фонового малюнка екрана входу в ОС Windows:

зайти у реєстр і пройти до запису HKEY_LOCAL_MACHINE/Software/Microsoft/Windows/CurrentVersion/Authentication/LogonUI/Background;

знайти у правій панелі ключ "OEMBackground". Якщо цього ключа немає, то створити його. Для цього клікнути правою кнопкою мишки на правій панелі, вибрати "Создать" і потім "Параметр DWORD" (32 біта). Створений ключ необхідно назвати відповідно "OEMBackground";

двічі клікнути по ключику, аби відкрити його;

тепер у полі "Значение" ввести 1;

клікнути OK;

За допомогою провідника Windows пройти в теку Windows/ System32/oobe. Якщо в цій папці є папка з назвою info, то потрібно увійти в неї. Якщо ж в info є тека з назвою backgrounds, то увійти і в неї. Якщо дві останні теки не існують, то створити їх;

скопіювати бажане зображення (це має бути JPEG-файл із розміром менше 256 Кб) у теку info/backgrounds;

потім перейменувати скопійований файл в backgroundDefault.jpg. Якщо розмір зображення відрізняється від роздільної здатності робочого столу, то зображення буде підігнано під ваш стіл – із можливою втратою його якості. Тека info/background також підтримує 12 інших файлів під певні роздільні здатності. Файли повинні мати назву backgroundXXXXX.jpg, де замість XXXXX слід вставити роздільні здатності 900 x 1440, 960 x 1280, 1024 x 1280, 1280 x 1024, 1024 x 768, 1280 x 960, 1600 x 1200, 1440 x 900, 1920 x 1200, 1280 x 768 і 1360 x 768. Так, наприклад, файл background1920 x 1200.jpg використовуватиметься на роздільній здатності 1920 x 1200.

Після перезавантаження комп'ютера у вікні входу Windows LogOn буде видно нову обрану картинку. Якщо вибрана картинка заважає читати написи на кнопках екрану, то варто спробувати налаштувати вид кнопок. Для цього:

пройти до ключа HKEY_LOCAL_MACHINE/Software/Microsoft/ Windows/CurrentVersion/Authentication/LogonUI;

у правій панелі створити параметр DWORD з назвою ButtonSet;

змінити його значення на 1 (тіні тексту у ході цього стануть темніші, а кнопка – світліша. Параметр призначений для світлих малюнків) або на 2 (немає тіней і непрозорі кнопки – для темних малюнків) або на 0, що Windows приймає за замовчуванням.

7. Персоналізація рядка заголовка IE8:

перейти до запису HKEY_CURRENT_USER/Software/Microsoft/ Internet Explorer/Main;

клікнути правою кнопкою мишки на правій панелі, вибрати "Создать" і потім "Строковый параметр";

назвати тільки що створений параметр "Window Title";

двічі клікнути по ньому мишкою;

увести в поле значення свій "апендикс" і клікнути OK.

8. Зміна поведінки стекової кнопки на панелі завдань.

За замовчуванням панель завдань групує декілька вікон одного застосування під однією кнопкою і потім під час кліку по цій кнопці показує їх мініатюри. Якщо ж ви думаєте, що було б краще, аби під час кліку по кнопці ОС автоматично переходила на останнє відкрите вікно, то ви можете це зробити. Для цього:

пройти до запису HKEY_CURRENT_USER/Software/Microsoft/Windows/CurrentVersion/Explorer/Advanced;

клікнути правою кнопкою мишки по правій панелі, вибрати "Создать" і потім "Параметр DWORD" (32 біта);

перейменувати створений параметр у "LastActiveClick";

двічі клікнути по "LastActiveClick", аби відкрити його;

змінити число в полі значення на 1;

клікнути OK.

9. Зміна розміру кнопок панелі завдань.

За замовчуванням Windows 7 завжди об'єднує кнопки панелі завдань від однієї програми і ніколи не відображує їх лейбли. Якщо ж ви тільки що повністю відключили об'єднання вікон або змусили операційну систему об'єднувати їх лише під час заповнення панелі завдань, то ви також можете змінити розмір іконок, аби приховати їх лейбли. Для цього:

пройти до запису HKEY_CURRENT_USER/ControlPanel/Desktop/WindowMetrics;

знайти у правій панелі ключ "MinWidth". Якщо його там немає, то створити його самостійно. Клікнути правою кнопкою мишки на правій панелі, вибрати "Создать" і потім "Строковый параметр". Після чого перейменувати створений параметр в MinWidth;

двічі клікнути по MinWidth, аби відкрити його;

змінити число в полі, значення з 38 (невеликі іконки) на 52 (крупні іконки);

клікнути OK.

Питання для самоконтролю

1. Які кореневі ключі має реєстр, у чому їх призначення?

2. Як зберегти реєстр перед редагуванням?

3. Як відновити реєстр?

4. Запишіть назви ключів, параметрів і значень, які ви використовували під час виконання завдань, у чому їх призначення?

5. Як деінсталювати програми, які не відображуються в меню "Установка или удаление программ"? (Напишіть шлях в реєстрі).

6. Чому перед редагуванням створюється точка відкоту в Windows?

Лабораторна робота 6
Моніторинг оптимізації та аудиту ОС Windows 7

Мета: практичне вивчення можливостей утиліт Windows, набуття навичок управління моніторингом ОС.

6.1. Теоретичні посилання

Утиліти – обслуговуючі програми, які надають користувачеві сервісні функції. Багато утиліт володіють розвиненим діалоговим інтерфейсом із користувачем і наближаються за рівнем спілкування до оболонок. Останні ж використовуються шляхом їх запуску з певними аргументами.

Програми-утиліти – це спеціальні комп'ютерні програми для виконання особливих функцій, призначені для розширення можливостей ОС. Це можуть бути безповоротне видалення файлів, відновлення даних, оптимізація Windows, очищення реєстру й інші важливі завдання.

Зазвичай програми-утиліти мають не дуже великий розмір і значно збільшують діапазон взаємодії з Windows.

Утиліти можуть бути вбудованими до ОС і бути частиною оболонки або бути самостійними програмами.

Утиліти, що існують у даний час, забезпечують реалізацію таких функцій:

а) обслуговування дисків, а саме:

форматування дисків у декількох режимах; відновлення помилково видалених файлів, а також у разі руйнування;

дефрагментація файлів на диску, унаслідок чого час доступу до файлів скорочується до 30 % і полегшується відновлення інформації в разі руйнування;

надійне затирання конфіденційної інформації.

б) утиліти друку;

в) шифрування інформації;

г) захист від комп'ютерних вірусів;

д) архівація даних.

Для виконання лабораторної роботи використовується пакет програм SysinternalSuite, який можна завантажити за посиланням: http://technet.microsoft.com/ru-ru/sysinternals. Пакет містить більше 60 утиліт для аналізу і моніторингу системи, роботи з файлами і дисками, мережеві і діагностичні програми.

Сервісні програми Sysinternal допомогають як фахівцям із інформаційних технологій, так і розробникам управляти, знаходити і усувати несправності і виконувати діагностику застосувань й операційної системи Windows.

6.2. ProcesExplorer

Дана програма призначена для моніторингу системних ресурсів. Володіє багатшим функціоналом, ніж стандартний "Диспетчер завдань". Це системна утиліта, що дозволяє відстежувати процеси, що відбуваються на комп'ютері. З її допомогою завжди можна взнати, що відбувається в операційній системі в даний момент.

Серед основних відмінностей ProcesExplorer від "Диспетчера завдань" визначають такі:

відображення запущених процесів у вигляді деревовидної структури;

наявність засобів графічного відображення даних (для підвищення якості сприйняття інформації);

вживання різнокольорового підсвічування даних (для зручності сприйняття);

виведення інформації про використовувані системою бібліотеки, що динамічно підключаються;

можливість запуску програм із різними правами користувача.

Інтерфейс програми є двома вікнами, у яких відображуються поточні процеси. У верхньому вікні програми відображуються всі процеси зі вказівкою облікових записів, яким ці процеси відповідають. У нижньому вікні, залежно від режиму роботи, можуть відображуватися всі запущені в даний момент дескриптори, а в режимі DLL – усі динамічні бібліотеки, завантажені в пам'ять тим або іншим процесом. Це дозволяє з точністю визначити, якому процесу відповідає інформація, що відображується. Це вельми корисно для користувачів, які мають навички поводження з операційними системами за системною частиною.

Крім усього іншого, програма володіє якісним пошуком. Завжди точно можна взнати, за допомогою якого застосування був відкритий той або інший файл, які бібліотеки завантажені тим або іншим дескриптором.

Можливості програми передбачають зміну будь-якого процесу, або ж, його повну зупинку і вивантаження з пам'яті комп'ютера. Крім усього іншого, за допомогою ProcesExplorer передбачена можливість запускати будь-які застосування, вимикати, перезавантажувати і навіть заблокувати комп'ютер, зберігати в текстовому LOG-файлі список усіх процесів із подібним описом процесів і розміром зайнятої кожним із них пам'яті, знаходити використовувані бібліотеки, включати підсвічування кольором певних процесів і багато що інше. Це, у свою чергу, допомагає контролювати працюючі процеси й отримувати детальну інформацію за кожним із них, а також характеристики використання системних ресурсів.

Викачати ProcesExplorer можна за посиланням: http://download.sysinternals.com/Files/ProcessExplorer.zip.

Після запуску програми в області повідомлень "Панелі завдань" з'являється характерний значок, під час наведення на нього курсора мишки з'являється спливаюча підказка з інформацією про завантаження центрального процесора і про те, який процес більш всього завантажує CPU (рис. 6.1).

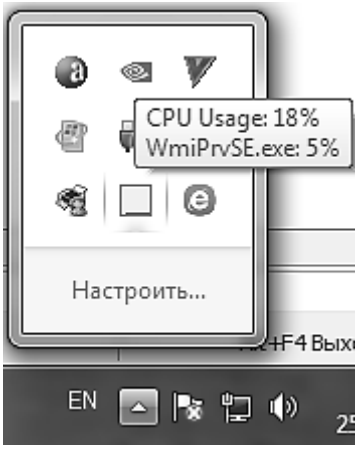


Рис. 6.1. **Знак ProcesMonitor на панелі завдань**

6.3. ProcesMonitor

Ця програма призначена для моніторингу файлової системи, реєстру і процесів в оперативній пам'яті. Ця утиліта об'єднує можливості програм Filemon (моніторинг файлової системи) і Regmon (моніторинг

реєстру), але у ході цього ProcesMonitor володіє ширшими можливостями. В утиліті доступний перегляд усіх завантажених бібліотек і внутрішніх kernel-драйверів різних пристроїв, виведення детальної інформації про кожен процес (повний шлях до файла, ID сесії, ім'я користувача).

Кожна колонка з даними, що відображуються в програмі, може переміщатися і віддалятися, присутня можливість встановлювати на кожен тип даних фільтр.

Інші можливості ProcesMonitor:

відстежує запуск і завершення роботи процесів і потоків, включаючи інформацію про код завершення;

дозволяє встановлювати фільтри, які не наводитимуть до втрати даних;

дозволяє в більшості випадків визначити вихідну причину виконання операції;

дерево процесів відображує стосунки між всіма процесами, перерахованими у відомостях трасування;

зберігає всі дані основного формату журналу, аби їх можна було завантажити в іншому екземплярі програми ProcesMonitor;

дає підказки до процесів для простого перегляду інформації про образ процесу;

дає детальні підказки, які дозволяють дістати зручний доступ до форматованих даних;

здійснює пошук, що припиняється.

Завдання 6.1

1. Запустити утиліту ProcesExplorer і виконати такі завдання:

Записати в звіт поточну загальну завантаженість процесора й об'єм зайнятої оперативної пам'яті (окремо розмір своп-файла і фізичної пам'яті).

Відкрити будь-який текстовий файл. Згорнути вікно процесу winword за допомогою програми ProcesExplorer.

Призначити для поєднання клавіш [Ctrl+Alt+Delete] виконання програми ProcesExplorer, а не "Диспетчера завдань".

Визначити, скільки потоків у процесів explorer і procexp.

З'ясувати, звідки запускається програма ProcesExplorer, скільки % ресурсів процесора і оперативної пам'яті займає дана програма.

Розташувати стовпці вікна програми в наступному порядку: CPU, PID, Working set, Private bytes. Зберегти такий вигляд програми.

Визначити кількість запущених процесів.

Відсортувати за стовпцем Working set і визначити, який процес займає більш всього оперативної пам'яті (ім'я процесу і ID).

Замалювати в звіт дерево процесів, підрахувати кількість батьківських процесів і процесів-сиріт.

Визначити, який процес більш усього завантажує процесор у даний момент.

Записати, які процеси стали "сиротами", після завершення процесу explorer.

2. Запустити утиліту ProcesMonitor і виконати такі завдання:

вивчити панель інструментів утиліти;

проаналізувати результат виконання запитів за всіма процесами в таблиці ProcesMonitor;

Запустити "Диспетчер завдань", знайти відповідний процес у вікні даних ProcesMonitor, залишити в списку процесів лише його в режимі перегляду активності реєстру і проаналізувати, до якого ключа було останнє звернення; тип звернення; перейти до даного об'єкта в редакторові реєстру.

Набудувати вигляд так, щоб відображувалася лише активність файлової системи, вимкнути запис подій і проаналізувати, до якого файла було останнє звернення; тип звернення. Відкрити в провіднику цей файл.

Набудувати фільтр, що працює за таким правилом: "Не відображувати події, в полі імені компанії яких присутнє значення Microsoft Corporation".

3. Запустити утиліту Desktop і виконати такі завдання:

Створити додатковий робочий стіл і набудувати гарячі клавіші перемикання між ними.

Запустити одну і ту ж програму (наприклад, Paint) на різних робочих столах і проаналізувати, яким чином відображуються процеси різних робочих столів у "Диспетчерові завдань"?

4. Запустити утиліту ZoomIt і виконати такі завдання:

збільшити область системного трея, обвести маркером значок ZoomIt, зробити скриншот і додати в звіт;

включити таймер на 1 хвилину в правому верхньому кутку екрана на тлі затемненого робочого столу.

5. Методика виконання завдання 7.1 ProcesExplorer.

Робоча область програми ProcesExplorer складається з двох вікон (рис. 6.2). У верхньому вікні відображується список активних процесів, включаючи назви облікових записів, яким належать ці процеси.

Інформація, яка відображується в нижньому вікні, залежить від вибраного режиму роботи програми. У режимі дескрипторів у нижньому вікні відображуються всі відкриті дескриптори (Handles) вибраного у верхньому вікні процесу, а в режимі бібліотек DLL – усі завантажені процесом динамічні бібліотеки і файли, що відображуються в пам'яті.

Для відображення (приховання) нижньої панелі натиснути піктограму Show/Hide Lower Pane на панелі інструментів, або вибрати меню View/Show(Hide) Lower Pane, або натиснути поєднання гарячих клавіш [Ctrl+L].

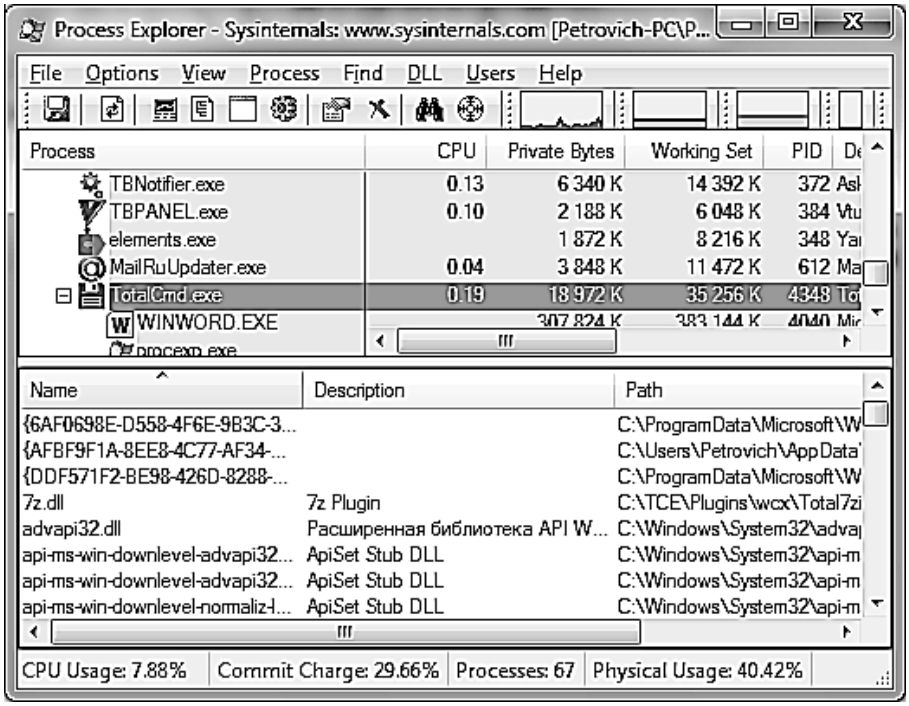


Рис. 6.2. **Робоча область програми ProcesExplorer**

Для включення режиму дескрипторів вибрати меню View / Lower Pane View /Handle (або натиснути [Ctrl+H]).

Для включення режиму бібліотек DLL вибрати меню View /Lower Pane View / DLL (або натиснути Ctrl+D).

Процеси в головному вікні відображують у вигляді дерева ієрархій. Тобто процеси поділені на батьківські і дочірні. Материнські процеси породжують дочірні. Є процеси-"сироти", вони відсортовані за лівим краєм і не мають вкладених процесів, їх "батьки" завершили роботу.

Вигляд списку процесів набудовується як у будь-якій програмі, використовуючою WINAPI. Стовпці можна перетягувати, сортувати за збільшенням і спаданням. Змінений зовнішній вигляд можна зберегти через меню View / Save Column Set.

Виділивши будь-який процес у верхньому вікні ProcesExplorer і клацнувши його правою кнопкою мишки (або вибравши меню Process), можна вибрати з контекстного меню потрібну дію (рис. 6.3.).

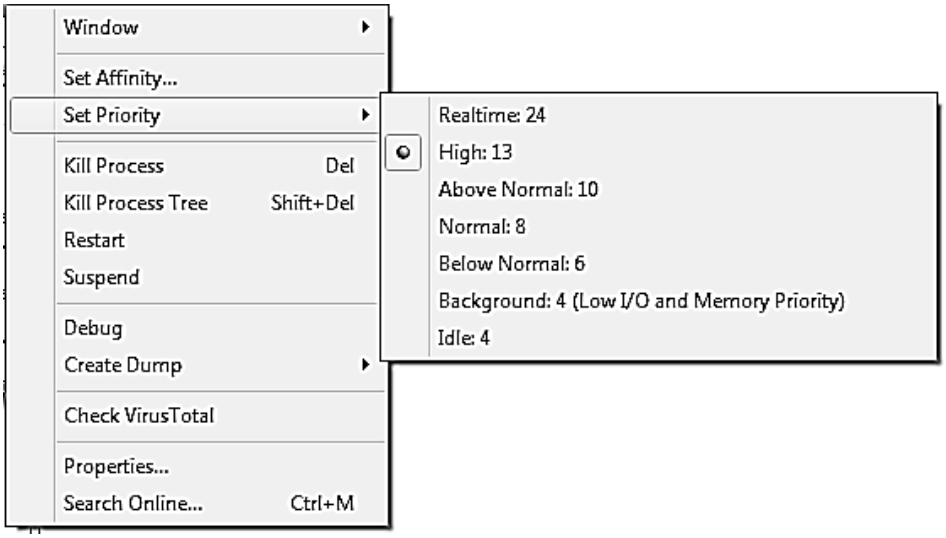


Рис. 6.3. **Контекстне меню процесу**

Window:
SMinimize (звернути)
SRestore (відновити)
SMaximize (розвернути на весь екран)
Bring to front (зробити активним)
Close (закрити).
Set Priority (встановити пріоритет процесу):
Realtime:24 (реального часу)
High:13 (високий)
Above Normal:10 (вище середнього)
Normal:8 (середній)
Below Normal:6 (нижче середнього)
Idle:4 (низький);
Kill Proces (завершити процес);
Kill ProcesTree (завершити дерево процесів);
Restart (перезапустити процес);
Suspend (Припинити) – для продовження роботи виберіть Resume (відновити);
Propertie (властивості) – викликається вікно властивостей процесу з вкладками (рис. 6.4):
Image (файл, з якого запускається даний процес), Performance;

Performance Graph (поточне використання ресурсів системи), Threads, TCP/IP, Security, Environment, Strings;

Search Online (пошук в Інтернеті) – пошук додаткових відомостей про програми/процеси в Інтернеті.

Аби проглянути потоки (Threads), що виконуються в рамках процесу, необхідно викликати діалогове вікно властивостей процесу, а в ньому – вкладку Thread (потоки).

Аби проглянути стек потоку процесу (Stack for thread) – у вкладці Thread (потоки) натиснути кнопку Stackp (рис. 6.5).

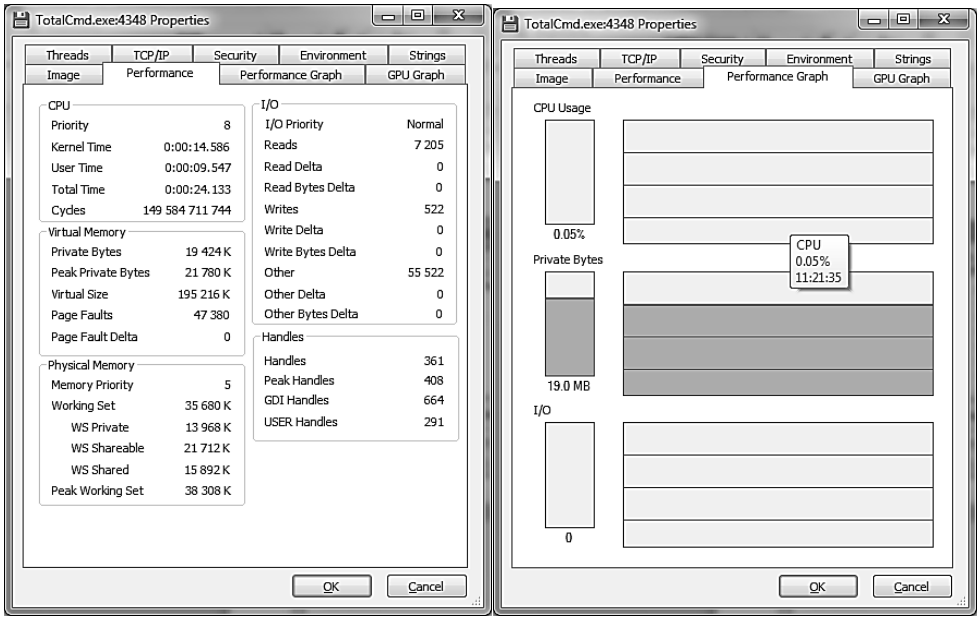


Рис. 6.4. **Вкладки Performance і Performance Graph**

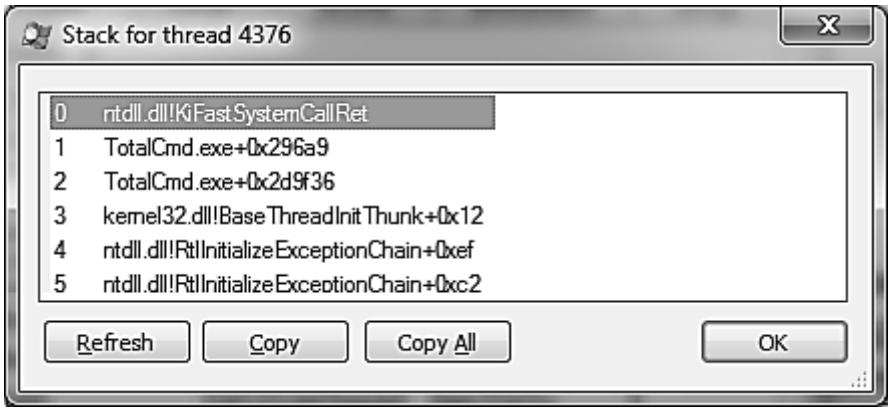


Рис. 6.5. **Стек процесу**

Детальна інформація про систему буде доступна, якщо вибрати в меню View/System Information (або натиснути відповідну піктограму на

панелі інструментів), у результаті відкриється вікно System Information (рис. 6.6 і 6.7).

Завантаження пам'яті відображується у вкладці Memory окремо для віртуальної пам'яті (своп-файл) – System commit і фізичною (безпосередньо ОЗП) – Physical memory (рис. 6.7).

Аби замінити стандартний "Диспетчер завдань" Windows на ProcesExplorer потрібно запустити ProcesExplorer і вибрати меню Option/ Replace Task Manager.

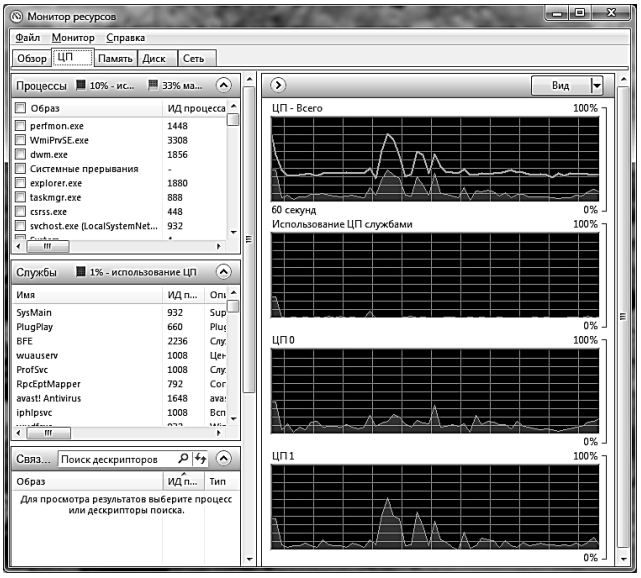


Рис. 6.6. **Вкладка CPU**

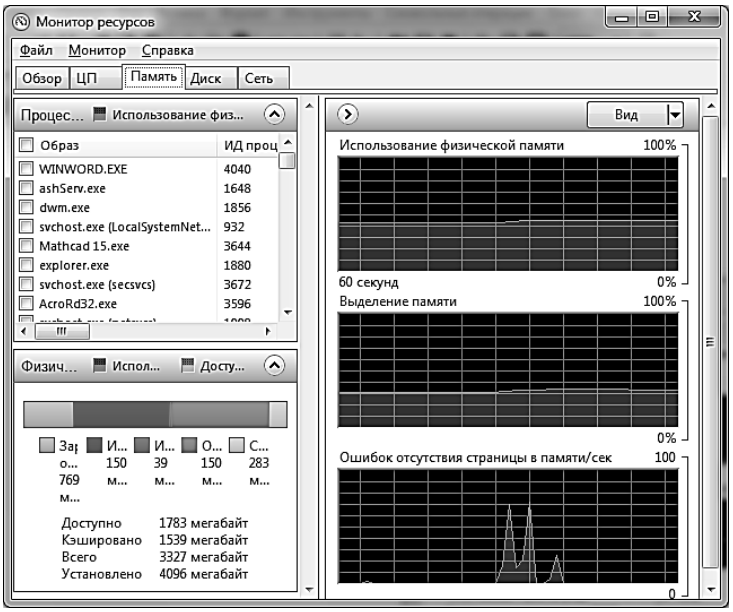


Рис. 6.7. **Вкладка Memory**

Після цього під час натиснення комбінації клавіш [Ctrl+Alt+Delete] автоматично запускатиметься не "Диспетчер завдань" Windows, а ProcesExplorer.

Аби повернути "Диспетчер завдань" Windows, необхідно запустити ProcesExplorer і вибрати меню Option/ Restore Task Manager.

6.4. ProcesMonitor

Утиліта ProcesMonitor виконує перехоплення контрольованих монітором системних функцій і збір даних тих, що підлягають моніторингу. Спостереження виконується для таких класів операцій:

звернення до файлової системи (file system);

звернення до реєстру (Registry);

робота з мережею (Network);

активність процесів (Process).

У ході першого запуску на екран буде видано ліцензійну угоду, що вимагає підтвердження користувача. Потім після старту програми ProcesMonitor, виводиться вікно з фільтрами для виключення з процесу спостереження подій стандартної активності системи і самого монітора.

Інтерфейс програми складається з 3-х частин – рядок меню (menu bar), панель інструментів (toolbar) і область виведення даних у вигляді списку (рис. 6.8).

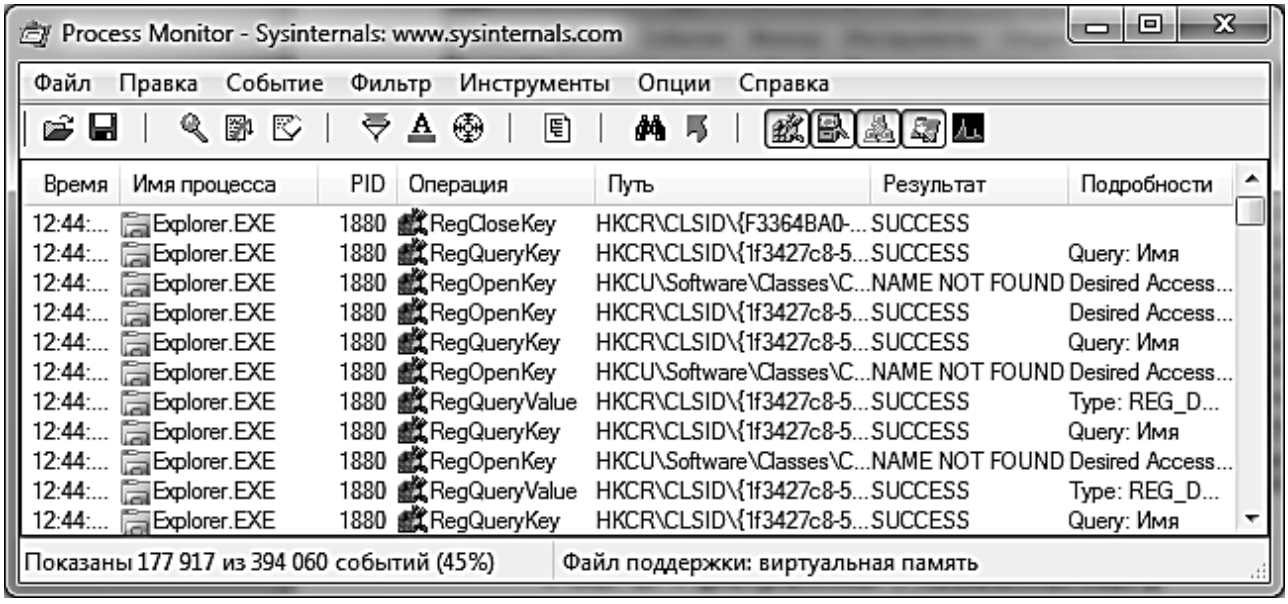


Рис. 6.8. **Програми ProcesMonitor**

Програма перехоплює відстежувані події, пов'язані з активністю процесів і видає дані відповідно до заданих критеріїв фільтрації і призначеними для користувача налаштуваннями колонок, що відображуються. Для зупинки моніторингу потрібно клацнути мишкою по кнопці з лупою на панелі інструментів так, щоб її зображення стало перекресленим червоною лінією. Повторне клацання поверне режим перехоплення.

Вкл\Викл запис подій, авто скролінг, очистити екран. Якщо значок лупи перекреслений – моніторинг активності процесів зупинений.

Автоскролінг – це режим автоматичної прокрутки екрану даних. Під час включення у вікні даних відображуватимуться останні за часом виконання операції. Якщо значок перекреслений – прокрутка не виконуватиметься. Інші піктограми панелі завдань:

"Открыть\Сохранить" поточні дані моніторингу у файл. Є можливість зберегти всі події (All Events), події, відфільтровані поточними фільтрами (Eventdisplayed using current filter) або події виділені підсвічуванням (Highlighted events). Можна також використовувати комбінацію клавіш [Ctrl+S].

Налаштування фільтру відображення, підсвічування рядків, пошук процесу серед вікон на екрані.

Задати умови фільтрації можна відразу після старту утиліти ProcesMonitor (рис. 6.9) або у будь-який момент часу з використанням меню програми, комбінації клавіш [CTRL+L] контекстного меню, що викликається правою кнопкою мишки.

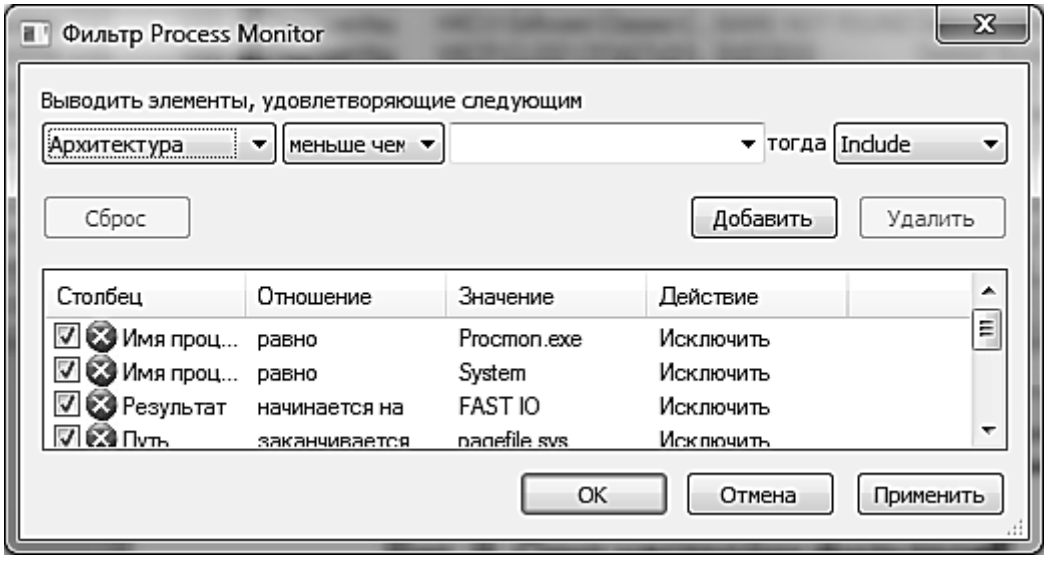


Рис. 6.9. **Вікно налаштування фільтрів**

Запис правила фільтрації складається з 4-х колонок: Column – колонка (вміст поля опису події) запису події. Можна вибрати одне з можливих полів, що відображується у вікні даних програми події. Relation – логічне вираження: I – рівне, набуває значення; Inot – нерівне; Lesthen – менше ніж; More than – більше ніж; Begin with – починається з; Endwith – закінчується на; Contain – містить; Exclude – не містить; Value – значення. Залежить від властивостей вибраного поля в першій колонці (Columns) Action – дія:

Exclude – вимкнути подію, відповідну умовам даного фільтра;

Include – увімкнути подію, відповідну умовам даного фільтра.

Для скидання фільтра використовується комбінація клавіш [CTRL+R].

Відображувати дерево процесів (як у ProcesExplorer), пошук на сторінці, перейти до об'єкта (файла, ключа реєстру).

У ході відображення дерева процесів (рис. 6.10) для кожного з них виводиться інформація з ім'ям, описом, шляхом виконуваного файла, власником, командним рядком запуску і часом старту. Під час установки покажчика мишки на рядок конкретного процесу, в нижній частині вікна буде виведена детальна інформація про нього.

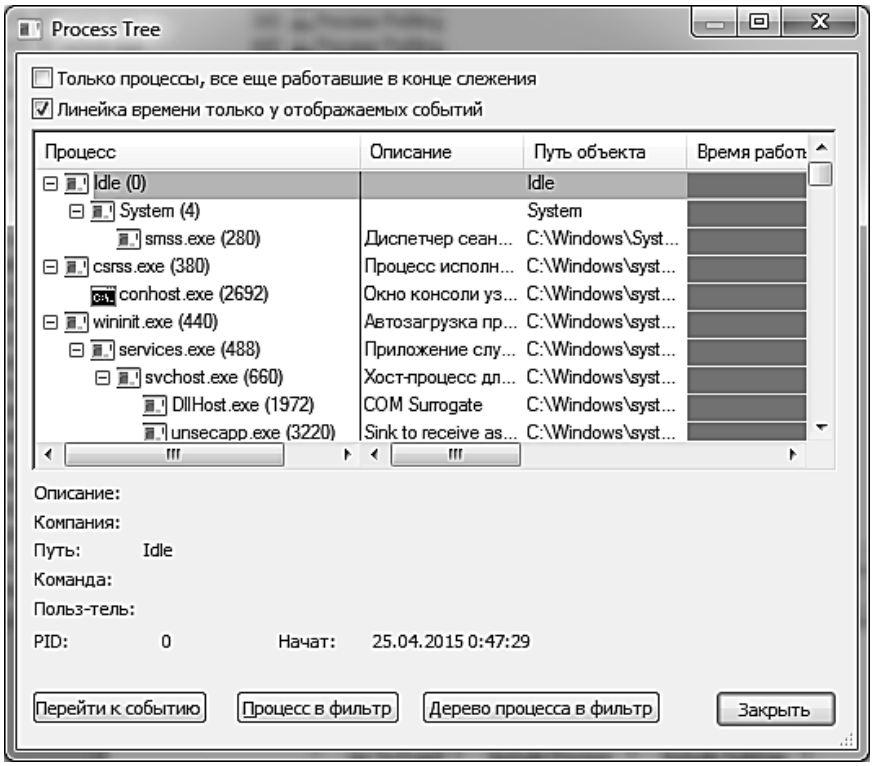


Рис. 6.10. **Дерево процесів**

Інформація відображується у вигляді ієрархічної структури (рис. 6.10), що відображує залежності між батьківськими і дочірніми процесами. Процеси, що мають одного і того ж батька, відображуються в порядку, відповідному часу запуску. Так, наприклад, процес winlogon.exe з ідентифікатором (PID) рівним 1772, запустив services.exe з ідентифікатором 1816, який, у свою чергу, породив декілька процесів svchost.exe і так далі.

Jump To Object (перейти до об'єкта) – це можливість швидкого переходу до досліджуваного об'єкта – ключа або розділу в редакторові реєстру, теки або файла в провіднику Windows; також можна використовувати комбінацію клавіш [Ctrl+J].

Вкл\Викл види активності (реєстр, файлова система, мережева активність, активність процесів і ниток).

Кожній події, перехопленій програмою ProcesMonitor, відповідає один рядок у вікні виведення даних. Подвійне клацання на окремому рядку викличе вікно перегляду властивостей події (Event Properties).

Порядок дотримання рядків відповідає послідовності виконання операцій. Інформація у вікні виведення даних розподілена на декілька стовпців, склад яких можна вибрати за допомогою контекстного меню Select Columns, що викликається правою кнопкою мишки на полі опису колонок або через головне меню Option/Select Columns (рис. 6.11).

Можливе виведення колонок, поділених на 3 категорії:

Application Detail – відомості про процес;

Event Detail – відомості про подію;

ProcesManagement – дані про батьківський процес, породжувані потоки і контекст облікового запису безпеки досліджуваного процесу.

Інформація, що дає уявлення про те, який процес, яку операцію виконав, і з яким результатом можна вивести, набудувавши такі поля:

1) Sequence – номер рядка (порядок дотримання події за часом) з початку сесії перехоплення відстежуваних подій;

2) ProcesName – ім'я процесу, що викликав подію;

3) Operation – виконувана операція. Значення залежить від типу звернення і є коротким описом, як, наприклад, відкриття ключа реєстру – RegOpenKey, відправка TCP пакета – TCP Send, читання файла – ReadFile і так далі;

4) Path – шлях, пов'язаний із використовуваним ресурсом. Це може бути файл, ключ реєстру, дані TCP з'єднання і тому подібне;

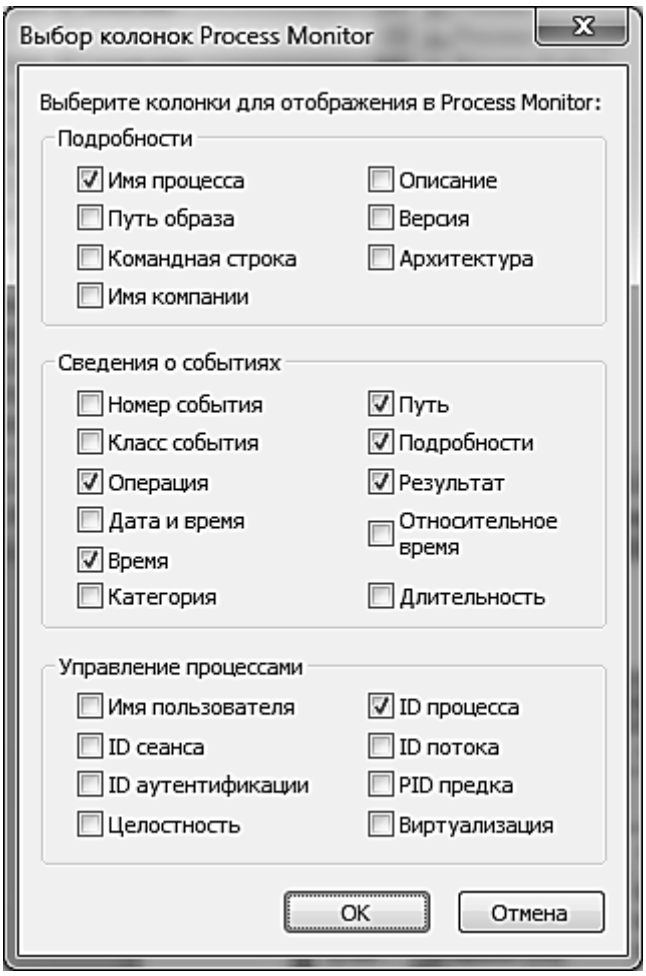


Рис. 6.11. **Налаштування колонок таблиці виведення даних**

5) Result – результат виконання запиту: end of file – виявлена ознака кінця файла (EOF) name not found – файл, каталог або дані реєстру не знайдені, name collision – була спроба створити новий файл, але файл з таким ім'ям вже існує;

file locked – файл відкритий для монопольного доступу;

Succes – операція виконана успішно;

invalid device request – неправильний запит до пристрою;

fast i/o disallowed – операція введення/виведення заборонена;

6) Detail – додаткова інформація про подію, що описує типа запиту, права доступу, властивості файла або каталога, типа даних, значення ключа реєстру і тому подібне.

6.5. Desktops

Desktop – програма для створення віртуальних робочих столів. Після запуску утиліти в треї з'являється іконка програми.

Desktop натисненням лівою кнопкою мишки відкриває вибір робочих столів, натиснення правої кнопки мишки викликає меню, звідки можна пройти в налаштування програми (Options) (рис. 6.12).

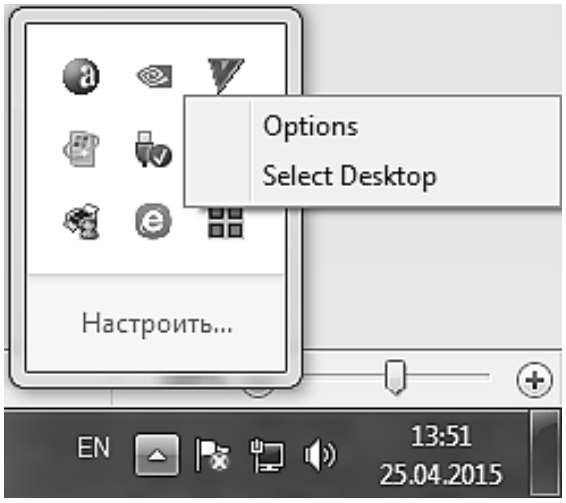


Рис. 6.12. **Контекстне меню Desktop**

У налаштуваннях можна вибирати типи запуску програми і гарячі клавіші для перемикання між робочими столами (рис. 6.13).

Рис. 6.13. **Вікно налаштувань програми Desktops**

Для включення автозапуску програми під час завантаження ОС необхідно поставити галочку в полі Run automatically at logon.

6.6. Zoomit

Zoomit – програма для збільшення області екрана. Після запуску, в треї з'являється іконка програми (рис. 6.14).

Під час натиснення на значку будь-якою кнопкою мишки, з'являється контекстне меню, з якого можна відкрити налаштування програми (Options), включити один із режимів збільшення або закрити програму (Exit).

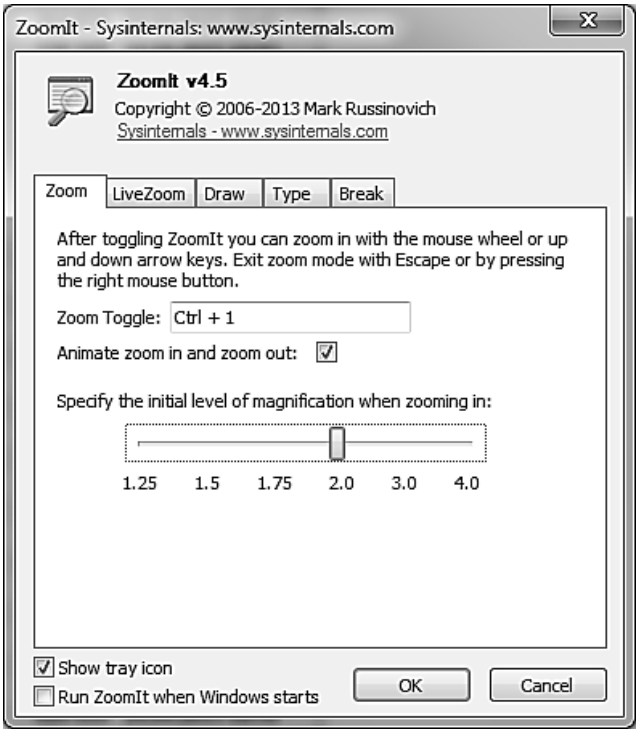


Рис. 6.14. **Вікно налаштувань програми**

У програмі є декілька варіантів збільшення екрану:

Zoom – включення режиму статичного збільшення, обертання колесом мишки регулює масштаб збільшення. Натиснення лівої кнопки мишки включає режим Draw, натиснення правої кнопки мишки вимикає збільшення.

LiveZoom – включення режиму динамічного збільшення. Мишка залишається активною, область збільшення слідує за мишкою. Виключення режиму здійснюється тією ж комбінацією клавіш, як і включення.

Draw – режим маркера, обертання колесом мишки регулює масштаб збільшення.

Type – написання тексту поверх картинки.

Break – включення таймера зворотного відліку, інтервал і інші налаштування таймера знаходяться у вкладці Break у налаштуваннях програми Zoomit. Додаткові налаштування (місце розташування таймера, фон і так далі) доступні натисненням кнопки Advanced.

Завдання 6.2

Зайти на сайт Sysinternals за адресою https://technet.microsoft.com/ru-ru/sysinternals і вивчити призначення і практичне застосування програм ProcesMonitor, Desktops і Zoomit.

Результат оформити у вигляді есе.

Питання для самоконтролю

1. У чому відмінність ProcesExplorer від "Диспетчера завдань"?
2. Як встановити двовіконний режим перегляду ProcesExplorer?
3. Як викликати діалогове вікно властивостей процесу, яку інформацію можна отримати у вкладці Performance?
4. Для чого потрібний авто скролінг?
5. Для яких операцій виконується спостереження за допомогою утиліти ProcesMonitor?
6. Яку інформацію містить кожен рядок у вікні виведення даних програми ProcesMonitor?
7. Як зберегти події, відфільтровані поточними фільтрами?
8. Якими трьома способами можна відображувати дерево процесів в ProcesMonitor?
9. Якими клавішами за замовчуванням перемикаються робочі столи?

Лабораторна робота 7
Система захисту і безпеки комп'ютера

Мета: вивчення можливостей захисника Windows і використання його для організації безпеки роботи ОС Windows 7.

7.1. Захисник Windows

Останніми роками шахрайство в Інтернеті почало нестримно розвиватися. Найчастіше шахраї розсилають величезну кількість повідомлень, які відправлені нібито надійними веб-вузлами і містять запит особистих відомостей із метою відмивання грошей, обману з боку платіжних

систем, відкриття електронних проектів і т. д. Зловмисники щодня намагаються вкрасти будь-які дані, якими вони можуть скористатися для відкриття нових рахунків кредитних карт або виконання через Інтернет інших дій від чужого імені. Фішинг – це вид Інтернет-шахрайства, метою якого є здобуття доступу до конфіденційних даних користувачів – логінів і паролів. Це досягається шляхом проведення масових розсилань електронних листів від імені популярних брендів, наприклад, від імені соціальних мереж, банків, а також інших сервісів.

Шпигунське програмне забезпечення, або іншими словами Spyware – це загальний термін, що використовується для опису програмного забезпечення, що виконує певні дії, такі, як реклама, збір особистої інформації або зміна конфігурації комп'ютера, як правило, без належного здобуття згоди користувача. Для боротьби зі шпигунським програмним забезпеченням розроблена безліч програмних продуктів.

Продукт компанії Microsoft, створений для відстежування шпигунського програмного забезпечення, який допомагає захистити комп'ютер від спливаючих вікон, зниження продуктивності і загроз безпеки, викликаних програмами-шпигунами й іншими небажаними програмами. Цей програмний продукт попереджає, коли шпигунське або потенційно небезпечне програмне забезпечення спробує встановитися або запуститися самостійно на комп'ютері. Захисник Windows також попереджає у тому випадку, коли програми намагаються змінити важливі параметри операційної системи.

У зв'язку з тим, що захисник Windows є компонентом системи, оновлення для цього програмного забезпечення можна завантажити з сайта Windows Update або за допомогою Windows Server Update Services (WSUS).

7.1.1. Відкриття програми захисника Windows

На відміну від операційної системи Windows Vista, в Windows 7 захисник Windows не входить до складу всіх програм меню "Пуск". Відкрити захисник Windows можна такими способами:

Натиснути на кнопку "Пуск" для відкриття меню, в полі пошуку ввести захисник і відкрити застосування в знайдених результатах;

Натиснути на кнопку "Пуск" для відкриття меню, відкрити "Панель управління", із списку компонентів панелі управління вибрати "Захисник Windows";

Скористатися комбінацією клавіш +R для відкриття діалогу "Виконати". У діалоговому вікні "Виконати", в полі "Відкрити" ввести

%windir%\system32\control.exe /name Microsoft.WindowsDefender і натиснути на кнопку ОК (рис. 7.1).

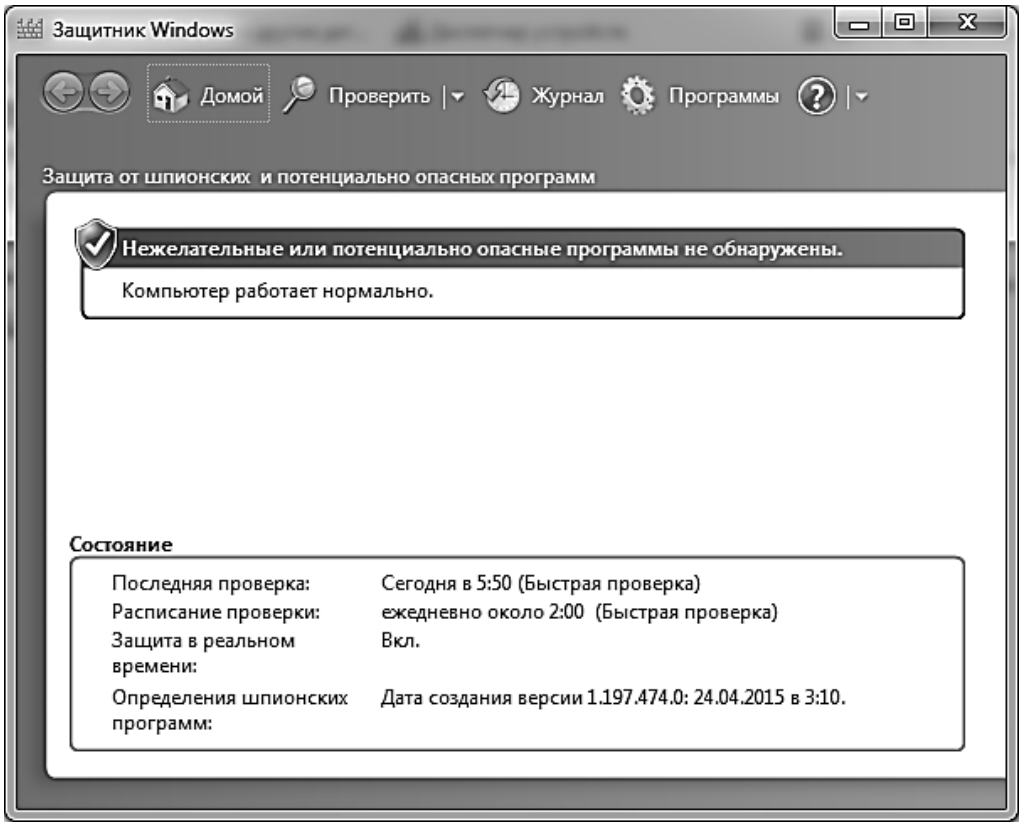


Рис. 7.1. **Захисник Windows**

7.1.2. Оновлення і перевірка комп'ютера

Програма Захисник Windows дозволяє захищати комп'ютер від шпигунського і небезпечного програмного забезпечення двома способами.

За допомогою перевірки операційної системи на наявність шпигунських програм, які могли встановитися на комп'ютері;

Шляхом захисту системи в режимі реального часу для сповіщення користувача про те, що шпигунська програма намагається встановитися або запуститися на комп'ютері.

Якщо оновлення захисника Windows не були завантажені через "Центр оновлення Windows" або вручну, то під час відкриття програми Захисника Windows буде видно, що для подальшої роботи потрібно відновити програму. Для цього натиснути на кнопку "Перевірити наявність оновлень". Також можна перевірити на наявність оновлень уручну, натиснувши на стрілку поряд із кнопкою довідки і вибравши опцію "Перевірити наявність оновлень" (рис. 7.2).

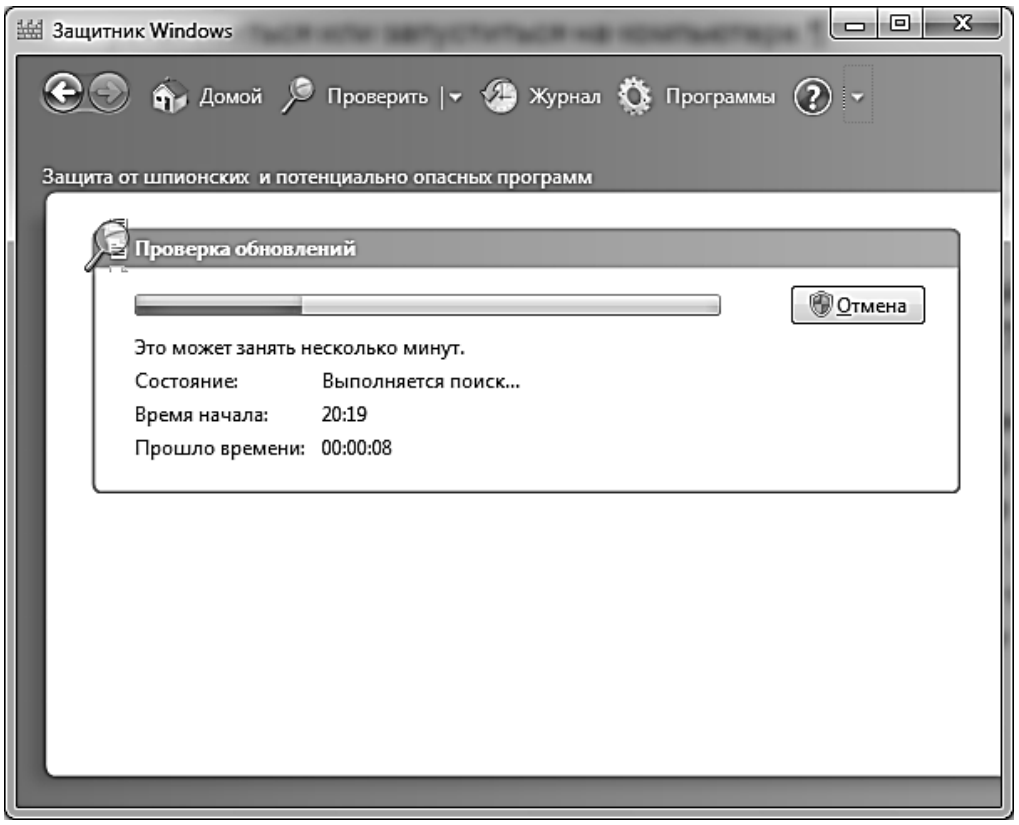


Рис. 7.2. **Перевірка наявності оновлень захисника Windows**

Після того, як вибрано цю команду, захисник Windows з'єднається з сервером оновлень Windows. Завантаживши останні оновлення, захисник Windows починає їх встановлювати. Операцію установки оновлень відмінити неможливо.

Після того, як програма встановить останні оновлення, в головному вікні захисника Windows можна буде побачити:

сповіщення про виявлення шпигунського або потенційно небезпечного програмного забезпечення;

коротку інформацію про стан оновлення, де відображується час закінчення установки оновлення, а також час, за яке оновлення було завантажене і встановлене;

детальну інформацію про стан захисника Windows та іншу інформацію.

У тому випадку, якщо програма була відкрита вперше, відразу після встанови оновлень, потрібно буде провести перевірку комп'ютера. Для цього натиснути на кнопку "Перевірити зараз".

У програмі захисника Windows передбачено три типи перевірки:

швидка перевірка;

повна перевірка;

вибіркова перевірка.

Швидка перевірка. Під час швидкої перевірки скануються лише ті ділянки системи, де найбільш висока вірогідність появи шпигунського або потенційно небезпечного програмного забезпечення. Ця перевірка за часом займає всього декілька хвилин. За замовчуванням у програмі захисника Windows вибраний саме цей тип перевірки, і він може бути запущений у будь-який час натисканням на посилання "Перевірити".

Повна перевірка. Під час повної перевірки на наявність шпигунського і потенційно небезпечного програмного забезпечення скануються всі файли на жорсткому диску, системна пам'ять, а також всі запущені в даний момент застосування. Окрім цього, під час повної перевірки виконується повне сканування всіх тек на комп'ютері, що може значно вплинути на швидкодію системи. Залежно від характеристик комп'ютера й об'єму записаної на нього інформації, повна перевірка може займати від однієї години і більш. Для запуску повної перевірки комп'ютера треба розвернути меню "Перевірити" і зі списку вибрати пункт "Повна перевірка".

Вибіркова перевірка. Під час вибіркової перевірки скануються лише ті диски і теки, які буде вказано. Тривалість сканування безпосередньо залежить від вказаної кількості дисків і тек. Для запуску вибіркової перевірки розвернути меню "Перевірити" і вибрати пункт "Вибіркова перевірка" (рис. 7.3).

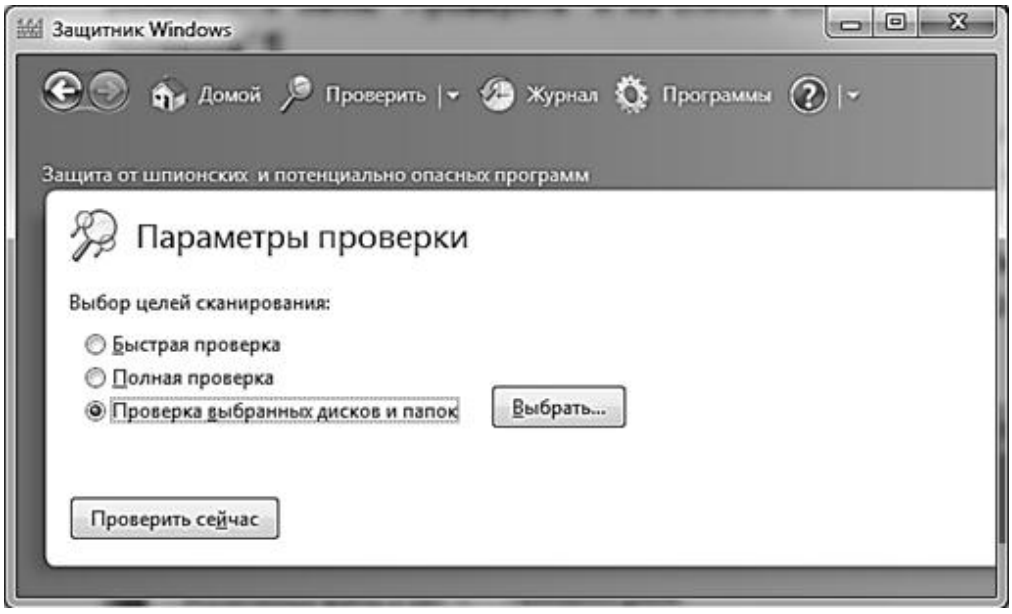


Рис. 7.3. **Вибіркова перевірка**

У цьому вікні можна вибрати два попередні типи перевірки або, встановивши перемикач "Перевірка вибраних дисків і тек", і натиснувши на кнопку "Вибрати", відзначити прапорцями лише ті диски теки, які потрібні для перевірки. Після того, як потрібні диски і теки будуть вибрані, натиснути на кнопку ОК (рис. 7.4).

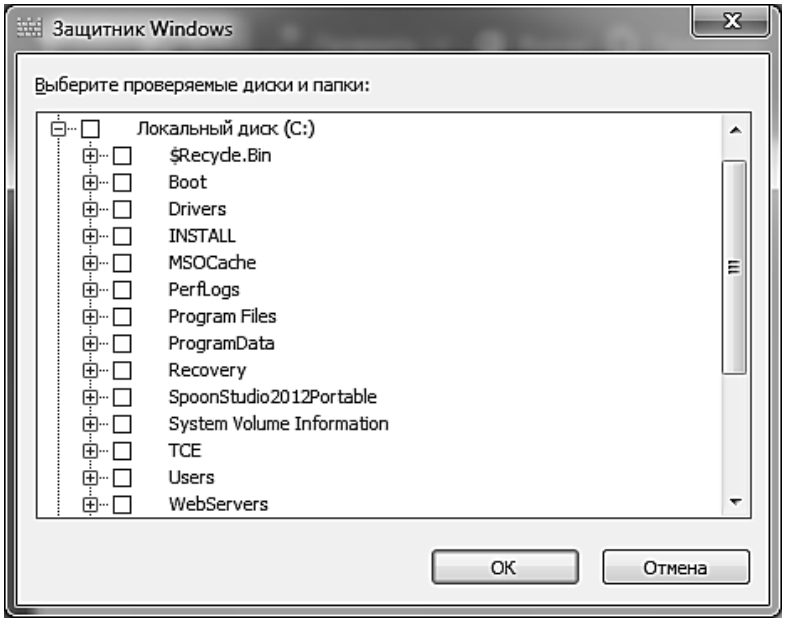


Рис. 7.4. **Вибір файла або теки для перевірки**

Натиснути на кнопку "Перевірити зараз" для запуску сканування. Під час перевірки комп'ютера на наявність шпигунського і потенційно небезпечного програмного забезпечення система, можливо, працюватиме повільніше, ніж зазвичай, у зв'язку з тим, що сканер програми перевіряє кожен файл на об'єкт зараження. Для того, щоб відмінити сканування комп'ютера, що робити не рекомендується, можна натискувати на кнопку "Відміну". Хід виконання перевірки відображується протягом всього процесу аж до завершення.

Після перевірки комп'ютера, в основному вікні захисника Windows можна побачити інформаційні повідомлення, схожі на ті, які можна бачити після оновлення програми. В області "Небажані або потенційно небезпечні програми не виявлені" відображується стан комп'ютера. В області статистики перевірки можна побачити тип останньої перевірки, час старту сканування, час, за який комп'ютер просканувався, а також кількість перевірених об'єктів. Область стану ідентична тій, яка відображується у вікні захисника Windows після оновлення баз.

У тому випадку, коли під час перевірки або після неї в програмі станеться яка-небудь помилка, пов'язана із зупинкою служби захисника

Windows, у вікні програми з'явиться повідомлення про зупинку служби, після чого потрібно буде натиснути на кнопку "Запустити" для забезпечення безпеки.

7.1.3. Журнал захисника Windows і об'єкти, поміщені на карантин

У журналі захисника Windows можна побачити всі дії, які проводилися над шпигунськими і потенційно небезпечними програмами, виявленими на комп'ютері. У полі "Дії над програмами" відображується список усіх шкідливих програм і дій, які захисник Windows з ними виконував. Для того, щоб проглянути детальну інформацію про яку-небудь дію, слід виділити і натиснути на кнопку "Перегляд". У разі появи запиту контролю облікових записів користувачів надати підтвердження. Аби видалити всі елементи списку, треба натиснути на кнопку "Очистити журнал". Для того, щоб проглянути всі об'єкти, запуск яким було дозволено, треба перейти за посиланням "Дозволені об'єкти". Для перегляду, видалення або відновлення всіх об'єктів, запуск яких був заборонений захисником Windows, слід перейти за посиланням "Об'єкти в карантині". На скриншоті можна побачити лише зовнішній вигляд журналу захисника Windows з порожнім списком дій над програмами (рис. 7.5), оскільки комп'ютер не містить шпигунського чи потенційно небезпечного програмного забезпечення.

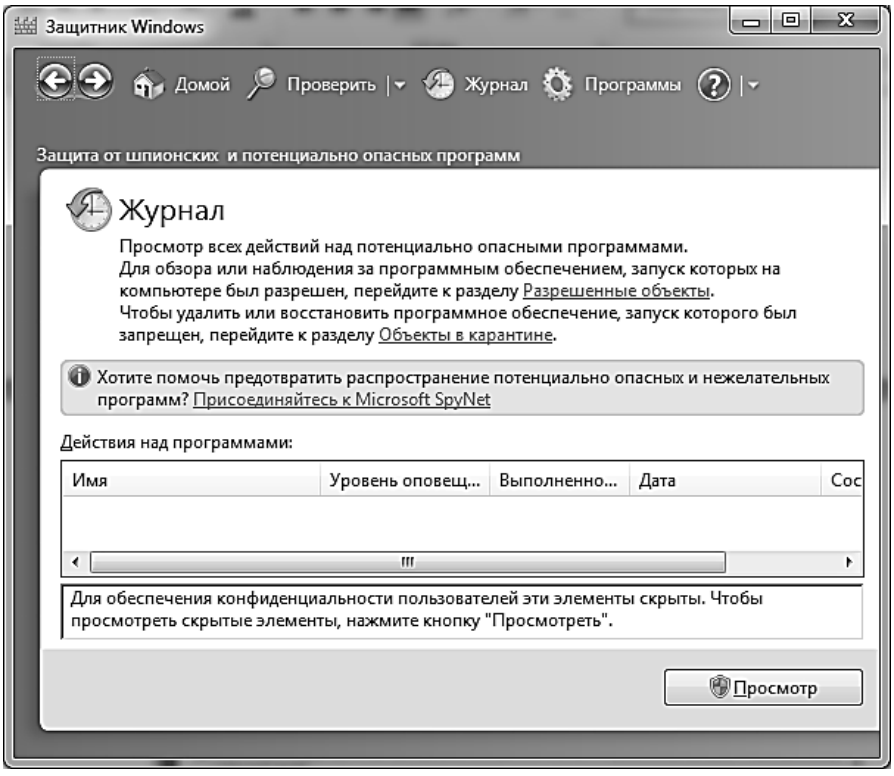


Рис. 7.5. **Журнал захисника Windows із порожнім списком дій з програмами**

На сторінці "Об'єкти в карантині" можна проглянути всі застосування, запуск яких був заборонений захисником Windows. У списку "Виберіть вживану дію" відображуються всі об'єкти, які програма вважає небезпечними. Для перегляду певної програми, натиснути на ній двічі лівою кнопкою мишки. Для відображення всіх елементів списку, натиснути на кнопку "Перегляд". Під час появи запиту контролю облікових записів користувачів дати підтвердження. Поміщаючи програму в карантин, захисник Windows переміщає її в особливе місце на комп'ютері. Для кожного з об'єктів можна натискувати на кнопку "Видалити" або "Відновити". Також для видалення всіх заражених об'єктів, можна натискувати на кнопку "Видалити все".

Для кожної програми існують різні рівні сповіщення. У програмі захисника Windows існує три види сповіщень:

критичний або високий;

середній;

низький.

Критичний або високий – це програми, які можуть виконувати збір особистих даних і негативно впливати на конфіденційність або можуть пошкодити комп'ютер.

Середній рівень – це програми, які можуть вплинути на конфіденційність і внести зміни, які можуть негативно позначитися на продуктивності системи.

Низький рівень – це небажані програми, які можуть виконувати збір відомостей про користувача або комп'ютер або змінювати характер роботи системи, але що працюють відповідно до ліцензійної угоди.

Завдання 7.1. Приєднання до співтовариства Microsoft SpyNet

Співтовариство Microsoft SpyNet є Інтернетом-співтовариством, що допомагає вибрати способи захисту від шпигунських і потенційно небезпечних програм. Це співтовариство також сприяє запобіганню поширення нових шпигунських програм. Захисник Windows дозволяє приєднатися до цього співтовариства і передавати на сервери Microsoft дані про виявлення на комп'ютері нового небезпечного програмного забезпечення. Деякі відомості відправлятимуться автоматично. Приєднатися до цього співтовариства можна такими способами. Перейти у вікно "Приєднатися до співтовариства Microsoft SpyNet". Це можна зробити за допомогою головного меню захисника Windows:

перейти у пункт меню "Журнал";

перейти у пункт меню "Програми" і натиснути на посилання;

перейти у пункт меню "Програми" і натиснути на посилання "Дозволені об'єкти";

перейти у пункт меню "Програми" і натиснути на посилання "Microsoft SpyNet".

У вікні "Приєднатися до співтовариства Microsoft SpyNet" вказати рівень своєї участі і натиснути на кнопку "Зберегти".

Вікно "Приєднання до співтовариства Microsoft SpyNet" показане на рис. 7.6.

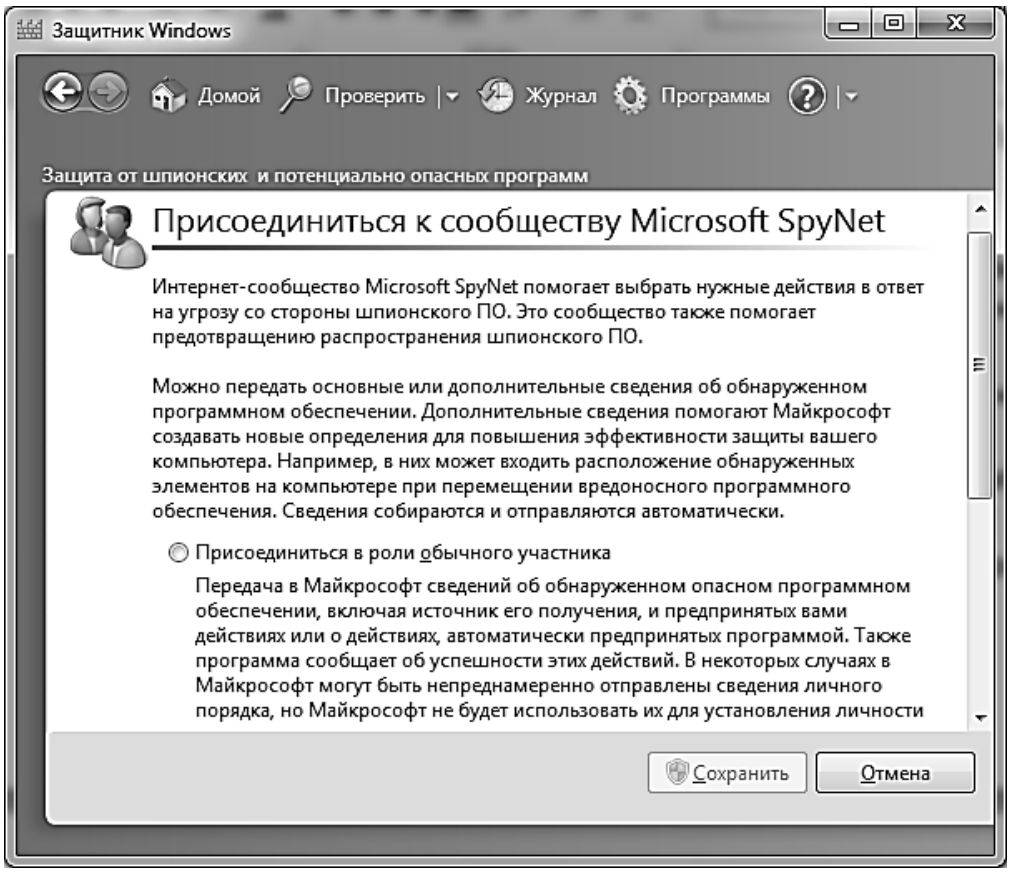


Рис. 7.6. **Вікно "Приєднання до співтовариства Microsoft SpyNet"**

Налаштування приєднання до співтовариства Microsoft SpyNet можна також вказати за допомогою системного реєстру.

```
;Приєднатися до співтовариства SpyNet
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\SpyNet]
"SpyNetReporting"=dword:00000001
```

;Приєднатися в ролі звичайного учасника
;"SpyNetReporting"=dword:00000001
;Приєднатися в ролі дослідного учасника
;"SpyNetReporting"=dword:00000002
;Не приєднуватися
"SpyNetReporting"=dword:00000000

Завдання 7.2. Налаштування захисника Windows

Діалогове вікно програм і параметрів (рис. 7.7) захисника Windows відкривається натисненням на кнопку "Програми" і дозволяє:

перейти до діалогу зміни параметрів захисника Windows;

приєднатися до співтовариства Microsoft SpyNet;

проглянути об'єкти, що знаходяться в карантині і дозволені об'єкти;

перейти на веб-сайт програми захисника Windows і в "Центр компанії Microsoft із захисту від шкідливих програм".

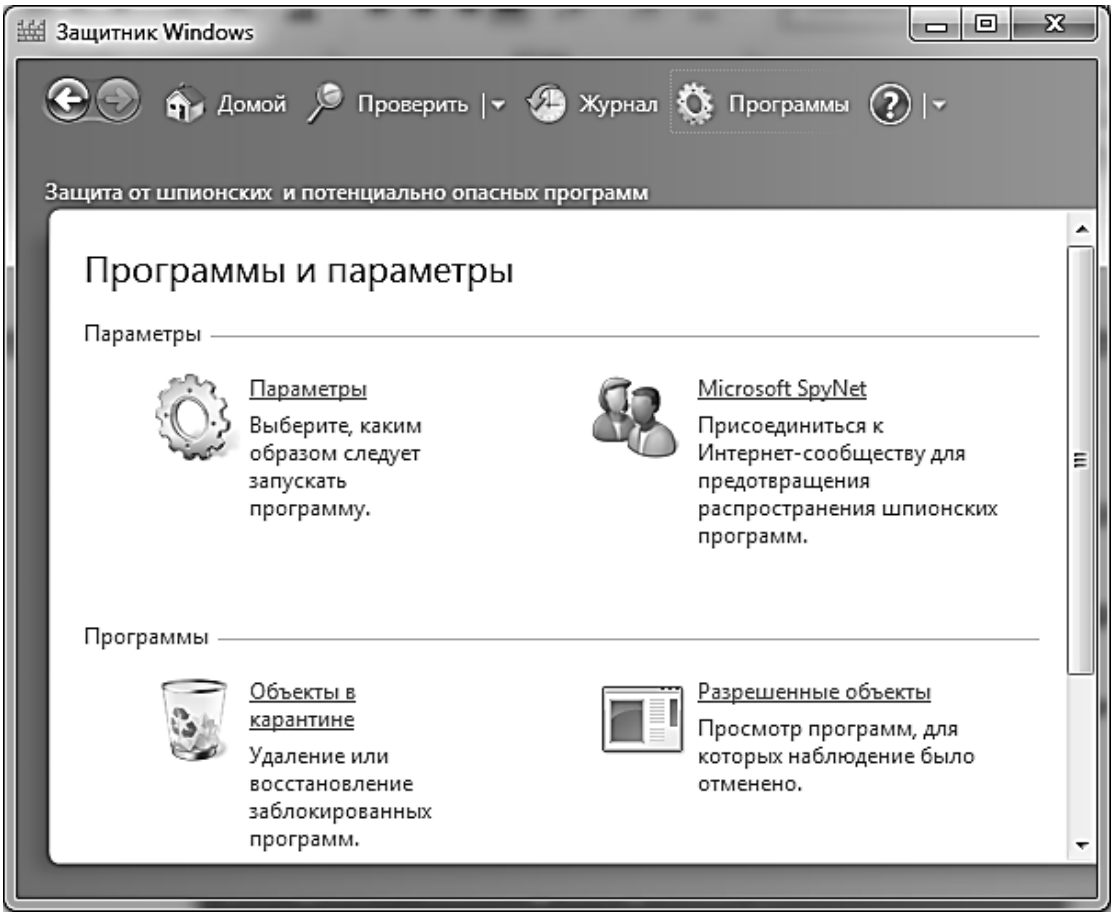


Рис. 7.7. **Діалогове вікно програм і параметрів захисника Windows**

За замовчуванням програма захисник Windows виконує швидку перевірку щодня о двадцятій годині вечора. Можна змінити періодичність, час і ще безліч налаштувань, перейшовши у вікно параметрів програми. Слід розглянути детально всі вкладки параметрів захисника Windows.

На вкладці "Автоматична перевірка" можна задати тип і частоту виконання перевірки на комп'ютері. Всі параметри вкладки "Автоматична перевірка" відображені на рис. 7.8.

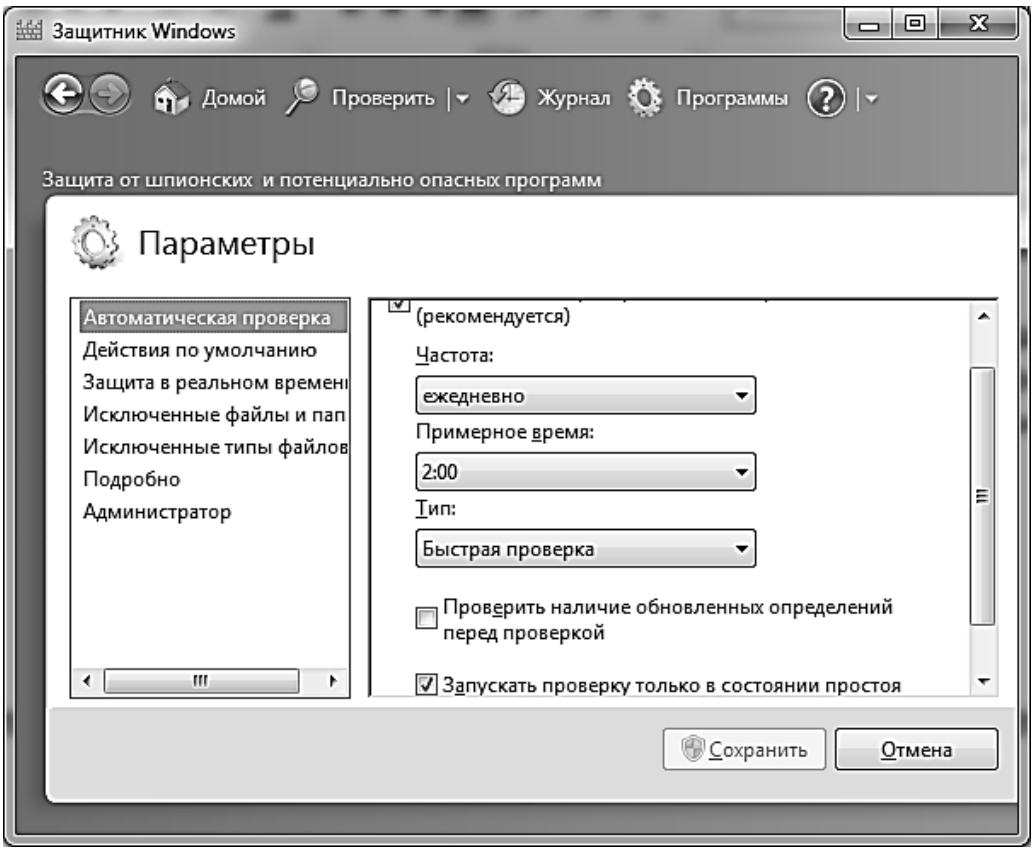


Рис. 7.8. **Параметри вкладки "Автоматична перевірка"**

Зняттям прапорця "Автоматично перевіряти комп'ютер" повністю відключається захист комп'ютера. У списках, що розкриваються, "Частота", "Приблизний час" і "Тип" можна змінювати значення, які вказані за умовчанням. Такого ж результату можна досягти, змінивши такі параметри в системному реєстрі.

```
;Автоматично перевіряти комп'ютер – частота
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"ScheduleDay"=dword:00000000
```

```
;Щонеділі
;"ScheduleDay"=dword:00000000
;Понеділок
;"ScheduleDay"=dword:00000002
;Вівторок
;"ScheduleDay"=dword:00000003
;Середа
;"ScheduleDay"=dword:00000004
;Четвер
;"ScheduleDay"=dword:00000005
;П'ятниця
;"ScheduleDay"=dword:00000006
;Субота
;"ScheduleDay"=dword:00000007
;Неділя
;"ScheduleDay"=dword:00000001
;Приблизний час (Приклад - 09-00)
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"ScheduleTime"=dword:0000021c
;Тип
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"ScanParameters"=dword:00000001
;Швидка перевірка
;"ScanParameters"=dword:00000001
;Повна перевірка
;"ScanParameters"=dword:00000002
```

Якщо встановити прапорець "Перевірити на наявність встановлених оновлень перед перевіркою", то перед автоматичною перевіркою комп'ютера запускатиметься оновлення. Також цю опцію можна встановити за допомогою реєстру.

```
;Перевірити на наявність встановлених оновлень перед перевіркою
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"CheckForSignaturesBeforeRunningScan"=dword:00000001
```

Установивши прапорець на опції "Запускати перевірку лише в стані простою", перевірка виконуватиметься лише в тому випадку, якщо на момент автоматичної перевірки не виконуватиметься жодних активних дій. Це також можна виконати засобами системного реєстру.

```
;Запускати перевірку лише в стані простою
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"ScanOnlyIfIdle"=dword:00000001
```

На вкладці дії за замовчуванням можна встановити дії, які виконуватимуться у разі виявлення шпигунських або потенційно небезпечних програм для всіх передбачених у захиснику Windows рівнів небезпеки. Для кожного з рівнів доступні такі дії: Дія, що "рекомендується, на основі визначень", "Видалити" або "Карантин". Також доступна установка дій засобами реєстру.

```
;Високий рівень небезпеки
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Threats\ThreatSeverityDefaultAction]
"5"=dword:00000003
;Дія, що рекомендується, на основі визначень
;"5"=-
;Видалити
;"5"=dword:00000003
;Карантин
;"5"=dword:00000002
;Високий рівень небезпеки
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Threats\ThreatSeverityDefaultAction]
"4"=dword:00000003
;Дія, що рекомендується, на основі визначень
;"4"=-
;Видалити
;"4"=dword:00000003
;Карантин
```

;"4"=dword:00000002
;Середній рівень небезпеки
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Threats\ThreatSeverityDefaultAction]
"2"=dword:00000003
;Дія, що рекомендується, на основі визначень
;"2"=-
;Видалити
;"2"=dword:00000003
;Карантин
;"2"=dword:00000002
;Вирішити
;"2"=dword:00000006
;Низький рівень небезпеки
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Threats\ThreatSeverityDefaultAction]
"1"=dword:00000003
;Дія, що рекомендується, на основі визначень
;"1"=-
;Видалити
;"1"=dword:00000003
;Карантин
;"1"=dword:00000002
;Дозволити
;"1"=dword:00000006

Установивши прапорець "Застосувати дії", що рекомендуються, програма захисник Windows не видаватиме запит, а почне відразу виконувати ті дії, які були виставлені в цьому діалозі.

Також можна це зробити засобами реєстру.

;Застосувати дії, що рекомендуються
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender]
"DisableRoutinelyTakingAction"=dword:00000001

http://www.oszone.net/figs/u/72715/091221114127/def-11.jpg

На вкладці "Захист у реальному часі", для того, щоб запобігти запуску шпигунських і потенційно небезпечних програм на комп'ютері,

слід встановити прапорець "Використовувати захист у реальному часі". Засобами реєстру це реалізовується таким чином.

```
;Використовувати захист у реальному часі
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection]
"DisableRealtimeMonitoring"=dword:00000000
```

Окрім цього можна встановити прапорці "Перевірка завантажених файлів і вкладень" і "Перевіряти виконувані на комп'ютері програми" для виконання відповідних дій. Також це можна зробити за допомогою системного реєстру.

```
;Перевірка завантажених файлів і вкладень
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection]
"DisableIOAVProtection"=dword:00000000

;Перевіряти виконувані на комп'ютері програми
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection]
"DisableOnAccessProtection"=dword:00000000
```

На вкладці "Виключені файли і теки" натиснути на кнопку "Додати" і вибравши з діалогу "Огляд файлів і тек" певні файли або теки, можна виключити їх з об'єктів для сканування. Це також можна зробити за допомогою системного реєстру.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths]
"ІМ’Я_ПАПКИ"=dword:00000000
```

Де в назві параметра "ІМ’Я_ПАПКИ" потрібно ввести повний шлях до теки або файла, який буде виключений із сканування.

У вкладці "Виключених типів файлів" можна додавати розширення, які не сканuватимуться. Додаючи лише розширення файлів у форматі "*.jpg" операційна система автоматично визначить типи файлів і додасть їх у список. За допомогою реєстру виключати типів файлів можна таким чином:

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Exclusions\Extensions]

"JPG"=dword:00000000

"PNG"=dword:00000000

"PS1"=dword:00000000

На вкладці "Детально" можна вибрати п'ять додаткових варіантів для перевірки комп'ютера:

"Перевіряти архівні файли" – сканування архівних файлів і тек, проте це може зайняти більше часу;

"Перевіряти електронну пошту" – перевірка вмісту повідомлень електронної пошти і вкладених файлів;

"Перевіряти знімні носії" – перевірка вмісту знімних носіїв;

"Використовувати евристику" – використання евристичного пошуку шпигунського і потенційно небезпечного програмного забезпечення. Отже, в журнал програми можуть потрапити навіть ті програми, відомості про які відсутні в оновленнях;

"Створення точки відновлення" – створення точки відновлення перед кожним видаленням програм із карантину.

Усі ці параметри можна налаштувати за допомогою системного реєстру таким чином.

;Перевіряти архівні файли

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]

"DisableArchiveScanning"=dword:00000000

;Перевіряти повідомлення електронної пошти

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]

"DisableEmailScanning"=dword:00000000

;Перевіряти знімні носії

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]

"DisableRemovableDriveScanning"=dword:00000000

;Використовувати евристику

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]

"DisableHeuristics"=dword:00000000

;Створити точку відновлення
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\Scan]
"DisableRestorePoint"=dword:00000000

На останній вкладці параметрів захисника Windows "Адміністратор" можна повністю відключити програму, знявши прапорець на опції "Використовувати цю програму", а також у журналі адміністративного облікового запису відображувати всі шпигунські і потенційно небезпечні програми для всіх користувачів поточного комп'ютера за допомогою опції "Показати елементи всіх користувачів комп'ютера". Ці дії можна виконати за допомогою реєстру.

;Використовувати цю програму
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender]
"DisableAntiSpyware"=dword:00000000

;Показати елементи всіх користувачів комп'ютера
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender\UX Configuration]
"DisablePrivacyMode"=dword:00000001
http://www.oszone.net/figs/u/72715/091221114127/def-16.jpg

Питання для самоконтролю

1. Призначення захисника Windows.
2. Функціональні можливості захисника Windows.
3. Як відкрити захисника Windows?
4. Чим займається співтовариство Microsoft?
5. Як долучитись до співтовариства Microsoft SpyNet?

Рекомендована література

Основна

Безбогов А. А. Безопасность операционных систем : учеб. пособ. / А. А. Безбогов, А. В. Яковлев, Ю. Ф. Мартемьянов. – М. : "Издательство Машиностроение-1", 2007. – 220 с.

Кокорева О. Реестр Windows 7 / О. Кокорева. – СПб. : БХВ-Петербург, 2010. – 704 с.

Колисниченко Д. Н. Секреты, настройка и оптимизация реестра Windows 7. – СПб. : БХВ-Петербург, 2010. – 320 с.

Мак-Федрис П. Microsoft Windows 7. Полное руководство / П. Мак-Федрис ; пер. с англ. – М. : ООО "ИД Вильямс", 2011. – 800 с.

Руссинович М. Внутреннее устройство Microsoft Windows / М. Руссинович, Д. Соломон. – 6-е изд. – СПб. : Питер, 2013. – 800 с.

Станек У. Windows 7 для продвинутых. Настройка, работа и администрирование / У. Станек. – СПб. : Питер, 2011. – 576 с.

Додаткова

Вавилов С. К. Самоучитель Windows 7 / С. Вавилов. – СПб. : Питер, 2010. – 272 с.

Жвалевский А. В. Windows 7 без напряга / А. В. Жвалевский. – СПб. : Питер, 2010. – 237 с.

Колисниченко Д. Н. Первые шаги с Windows 7: руководство для начинающих / Д. Н. Колисниченко. – СПб. : БХВ-Петербург, 2010. – 396 с.

Лебедев А. Н. Windows 7 u MS Office 2010: компьютер для начинающих : завтра на работу! / А. Н. Лебедев. – СПб. : Питер, 2010 – 250 с.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "ОПЕРАЦІЙНІ СИСТЕМИ"

для студентів спеціальності 7.05150102 "Технології електронних мультимедійних видань" усіх форм навчання

Самостійне електронне текстове мережне видання

Укладач **Гаврилов** Володимир Петрович

Відповідальний за випуск *О. І. Пушкар*

Редактор *В. О. Бутенко*

Коректор *М. А. Ковальчук*

План 2015 р. Поз. № 144 ЕВ. Обсяг 129 с.

Видавець і виготівник – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Леніна, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***