МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

О.П. Чекалов

# ОСНОВИ ФУНКЦІОНУВАННЯ ОПЕРАЦІЙНИХ СИСТЕМ. ПРАКТИКУМ

Рекомендовано Міністерством освіти і науки України

Видавництво СумДУ

2010

УДК 004.451(076)
ББК 32.973-01
Ч 37

Рецензенти:
д-р техн. наук, проф. О.Ю. Соколов
(Національний аерокосмічний університет ім. М.Є.Жуковського
"Харківський авіаційний інститут", м. Харків);
д-р техн. наук, проф. Є.А. Лавров
(Національний аграрний університет, м. Суми);
д-р фіз.-мат. наук, проф. С.П. Рощупкін
(Інститут прикладної фізики НАН України, м. Суми)

*Рекомендовано Міністерством освіти і науки України*
*як навчальний посібник для студентів вищих навчальних закладів*

*(лист № 1.4/18-Г-12 від 9.01.2009 р.)*

**Чекалов О.П.**

Ч 37 Основи функціонування операційних систем. Практикум: навчальний посібник/ О.П. Чекалов. — Суми: Вид-во СумДУ, 2010.- 85 с.
ISBN 978-966-657-275-5

Матеріали посібника розширюють лекційний курс і використовуються для ілюстрації реалізації теоретичних положень на прикладі операційної системи Windows. На практиці розглядаються взаємодія операційної системи з її оточенням, організація процесів, способи їх взаємодії, керування пам'яттю та файлова система.

Призначений для студентів вищих навчальних закладів, які навчаються за напрямом "Інформатика" і вивчають сучасні інформаційні технології у рамках дисциплін "Операційні системи" та "Системне програмування", а також для викладачів зазначених дисциплін.

УДК 004.451(076)
ББК 32.973-01
ISBN 978-966-657-275-5

# Зміст

# Вступ

Предметом посібника є практичне ознайомлення з фундаментальними концепціями, які покладені в основу сучасних операційних систем Windows і UNIX/Linux.

Посібник підготовлений за матеріалами лекційних і практичних занять курсу "Операційні системи", що автор протягом багатьох років викладає для студентів, які вчаться за напрямом 6.040302 - “Інформатика”.

# Структура книги

- *Перший розділ* посібника присвячено питанню взаємодії операційної системи з її оточенням. Тут (за допомогою інструментарію сторонніх виробників) показано, як програми передають керування операційній системі для виконання найбільш загальних операцій: робота з диском, монітором і т.п. Виклад матеріалу супроводжується практичним ознайомленням із програмними перериваннями й Win32 API.
- *Другий розділ* присвячений фундаментальному поняттю операційних систем — "процесу". Це основний динамічний об'єкт, над яким системи виконують певні дії. Розглянуто проблему "перегони", що виникає за відсутності синхронізації потоків. Поняття "критична секція" й "мьютекс" у контексті розв'язання завдання синхронізації роботи процесів. Виклад матеріалу супроводжується практичним ознайомленням з потоками шляхом їхнього дослідження за допомогою інструментарію сторонніх виробників, а також керуючись описами програмних реалізацій, які наведені автором у посібнику.
- *Третій розділ* присвячений опису взаємодії процесів (interprocess communication — IPC). Воно має місце як між потоками одного процесу, так і між процесами різних комп'ютерів. Тут (із залученням інструментарію сторонніх виробників, а також у процесі розроблення додатків) розглянутий обмін

даними: повідомлення Windows, мейлслоти й іменовані канали. Не обійдені увагою й такі специфічні механізми взаємодії процесів, як "файли, відображувані у пам'ять", і протокол Component Object Model (COM).

- *У четвертому розділі* описаний процес вирішення завдання нестачі фізичної пам'яті для виконання користувальницьких процесів й операційної системи. Використовуючи інструментарій сторонніх виробників, показано, як Windows вирішує цю проблему, використовуючи механізм "файли, відображувані у пам'ять". Виклад матеріалу супроводжується описом процесу розроблення програми. Це дозволяє показати, що не тільки розроблювачі ОС, але й будь-який програміст може використати віртуальну пам'ять.
- *П'ятий розділ* присвячений опису низкорівневої структури диска, організації розподілу на ньому даних. За допомогою інструментарію сторонніх виробників проводиться детальний аналіз структур даних дисків. Показано, як, використовуючи Win 32 API визначити тип файлової системи, параметри низкорівневої структури диска.

## Як працювати з навчальним посібником

Практичні заняття, наведені у посібнику, строго не регламентовані. Це дозволяє викладачеві організовувати практичні заняття за зручною для нього схемою, орієнтуючись на апаратні ресурси, якими він володіє в аудиторії. Крім того, керуючись міркуваннями політики прав стосовно апаратних ресурсів навчальної аудиторії, можна будувати практичні заняття як творче обговорення й захисти результатів самостійної роботи, отриманих за допомогою інструментарію сторонніх розробників. В аудиторії можна ознайомлюватися з фундаментальними концепціями сучасних операційних систем "зсередини", керуючись описами програмних реалізацій, які наведені автором у посібнику.

## Програмне середовище

Іноді відтворюваність читачем прикладів посібника залежить від програмного середовища. У посібнику середовище — це операційна система Windows XP, набір утиліт та розроблених в середовищі Delphi 6 програм. Вибір Delphi 6 не випадковий. По-перше, Delphi 6 не ставить підвищених вимог до ресурсів комп'ютера. По-друге, курс "Операційні системи" не вимагає глибоких пізнань в об'єктоорієнтованих мовах.

## Підтримка

Посилання на використовувані у посібнику утиліти, вихідні коди Delphi-програм (з описом процесу їхнього розроблення) і відкомпільовані модулі ви знайдете на сайті автора — **http://chekalov.sumdu.edu.ua**.

# 1

# Взаємодія операційної системи з її оточенням

Показано, як програми передають керування операційній системі для виконання найбільш загальних операцій: роботи з диском, монітором тощо. Виклад матеріалу супроводжується практичним ознайомленням із програмними перериваннями й Win32 API.

Розділ 1

# Взаємодія операційної системи з її оточенням

Ключові компоненти архітектури Windows можна схематично зобразити, як показано на рис. 1.1. [1]

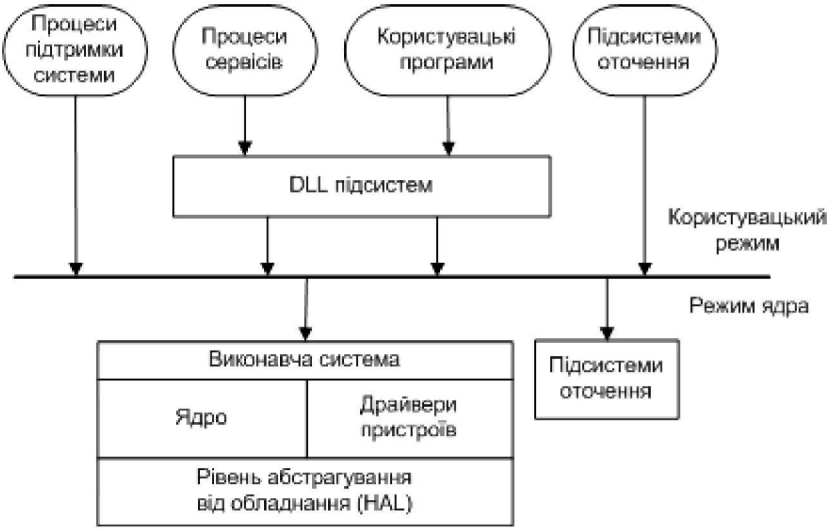



Рисунок 1.1 — Спрощена схема архітектури Windows

На рис. 1.1 наведено чотири типи процесів користувацького режиму: процеси підтримки системи (наприклад, диспетчер сеансів), процеси сервісів (наприклад, менеджер завдань — Task Scheduler), користувацькі програми (наприклад, Win32, MS-DOS) і підсистеми оточення — сервіси.

Ви можете подивитися, які модулі Windows функціонують у режимі ядра (рис 1.2).

Для цього клацніть правою клавішею миші на значку **Мой компьютер**, потім виберіть вкладку **Оборудование/Диспетчер устройств/Компьютер/Драйвер/Сведения**.

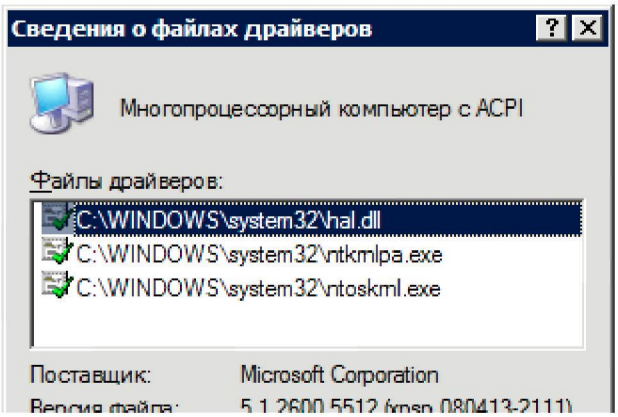



Рисунок 1.2 — Модулі ядра і HAL

***Таблиця 1.1*** *— Модулі Windows, що функціонують у режимі ядра*

| Ім'я файла | Компоненти |
|---|---|
| hal.dll | Рівень абстрагування від пристроїв |
| ntkrnlpa.exe | Виконуюча система і ядро з підтримкою механізму Physical Address Extension (PAE) |
| ntoskrnl.exe | Виконуюча система і ядро |

## 1.1 API-функції як системний виклик

На рис. 1.1 показано, що користувацькі програми викликають модулі Windows, які функціонують у режимі ядра за допомогою звертання до DLL-підсистем. Ці DLL надають документований інтерфейс між програмами й викликуваною ними підсистемою. Так, DLL-підсистеми Win32 (Kernel32.dll, Advapi32.dll, User32.dll і Gdi32.dll) реалізують функції Win32 API.

Ви можете подивитися, які Win32 API-функції реалізовані в DLL-підсистемі Win32 (рис 1.3).

Для перегляду реалізацій DLL-підсистеми Win32 вам достатньо запустити програму Dependency Walker [2] та відкрити будь-який файл, використовуючи команду **Open** меню **File**.

Ознайомимося на практиці з використанням Win32 API-функції.

Нехай необхідно визначити логічні пристрої комп'ютера. Для цього можна використати функцію `GetLogicalDrives`, що повертає побітову маску. Кожен біт цієї маски показує наявність або відсутність логічного пристрою. Так одиничне значення нульового біта означає наявність у системі диска A, одиничне значення першого біта — наявність диска B і т. д.

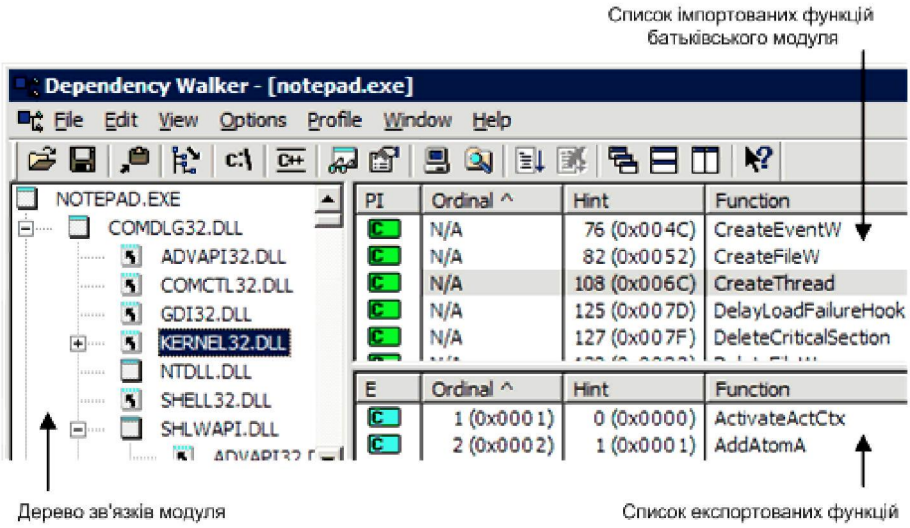



Рисунок 1.3 — Реалізація DLL-підсистеми Win32

Оброблення бітів реалізуємо функцією `DriveExists`. Визначення того факту, що біт установлений, використаємо стандартну процедуру IN мови Object Pascal, реалізовану в Delphi.

```
function DriveExists(Drive: Byte): Boolean;
var
  LogDrives: set of 0..25;
begin
  Integer (LogDrives):=GetLogicalDrives;
  Result:=Drive in LogDrives;
end;
```

Щоб довідатися, які з 26 можливих логічних пристроїв установлені, необхідно викликати функцію `DriveExists` у циклі. Нижче по-

казано, як заповнити вміст списку іменами встановлених логічних пристроїв:

```
procedure TForm1.FormCreate(Sender: TObject);
var
     D: Byte;
begin
  For D := 0 to 25 do  { Для всіх пристроїв }
  If DriveExists(D) Then  { Якщо існує }
    ListBox1.Items.Add(Chr(D+$41));  { Додати до списку }
end;
```

Для реалізації програми:

- помістіть на форму компонент `TListBox1`;
- клацніть двічі мишею на формі й допишіть код процедури й функції;
- скомпілюйте програму.

Приклад результату визначення логічних пристроїв комп'ютера наведений на рис. 1.4.

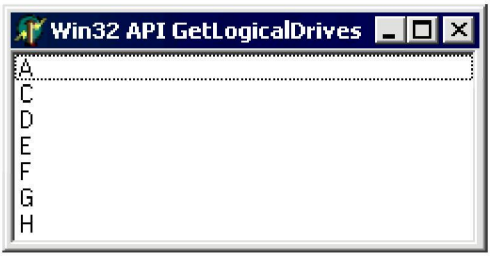



Рисунок 1.4 — Список логічних пристроїв комп'ютера

Код проекту ви можете завантажити із сайта автора [3].

## 1.2 Переривання

*Переривання — це стан операційної системи, що змушує процесор почати виконання коду не належному поточному потоку команд*. Воно може наставати з ініціативи пристроїв або програм. При виявленні переривання процесор припиняє виконувати поточний потік команд і передає керування в особливе місце пам'я-

ті за адресою коду, що обробляє виниклий стан. У Windows цей код називають *оброблювачем пастки* (trap handler).

У системах типу x86 переривання, пов'язані із зовнішніми пристроями, надходять по одній з ліній контролера переривань. Контролер, у свою чергу, пов'язаний із процесором єдиною лінією, по якій і повідомляє про переривання. Як тільки процесор переривається, він вимагає від контролера запиту переривання (interrupt request, IRQ). Контролер транслює IRQ у номер переривання, використовуваний як індекс у структурі, називаною таблицею диспетчеризації переривань (interrupt dispatch table, IDT), і передає керування відповідній процедурі. При завантаженні Windows заносить в IDT покажчики на процедури ядра, що обробляють кожне переривання.

Ви можете подивитися, які IRQ зіставлені контролером зовнішніх пристроїв. Для цього:

- клацніть правою клавішею миші на значку **Мой компьютер**;
- виберіть у меню, що випадає, пункт **Свойства**, а потім вкладку **Оборудование**;
- на вкладці **Оборудование** клацніть на кнопці **Диспетчер устройств**;
- виберіть пристрій і клацніть на ньому правою клавішею миші;
- у меню, що з'явиться, виберіть пункт **Свойства**, а потім вкладку **Ресурсы** (рис 1.5).

Тепер подивимося вміст IDT, включаючи відомості про оброблювачів пасток, які Windows призначила перериванням. Для цього використаємо налагоджувач ядра:

```
kd> !idt

Dumping IDT:

30:      806b14c0 hal!HalpClockInterrupt
31:      8a39dc3c i8042prt!I8042KeyboardInterruptService
                        (KINTERRUPT 8a39dc00)
```

```
34:      8a436dd4 serial!SerialCIsrSw (KINTERRUPT
8a436d98)
```

Команда `!idt` показує номера переривань, які зіставлені з адресами поза модулем Ntoskrnl.exe. Так номеру переривання 31 зіставлена адреса 8a39dc3c і програма обробки переривання i8042prt.

Налагоджувач ядра можна завантажити із сайта корпорації Microsoft [4]. Там само дані докладні інструкції з інсталяції налагоджувача.

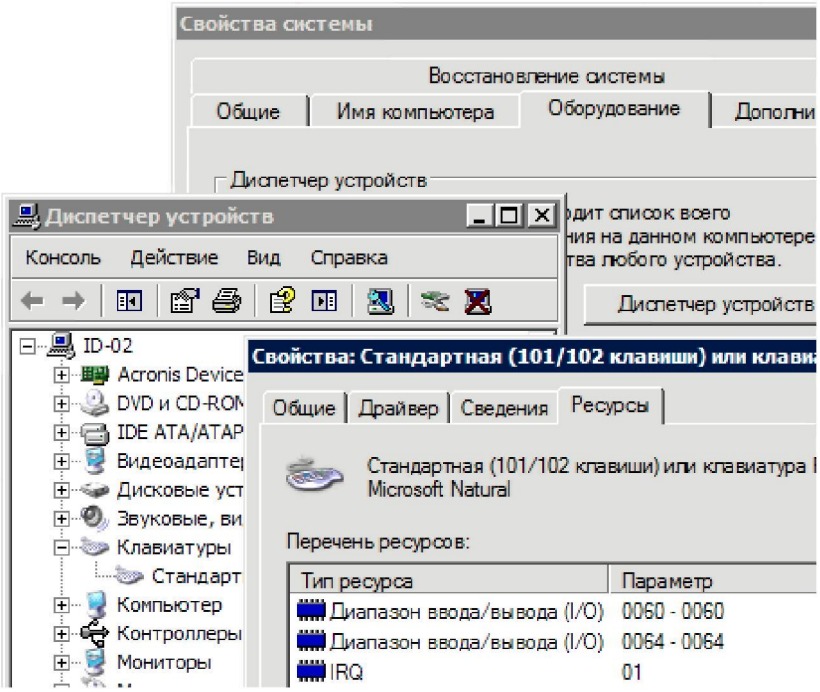


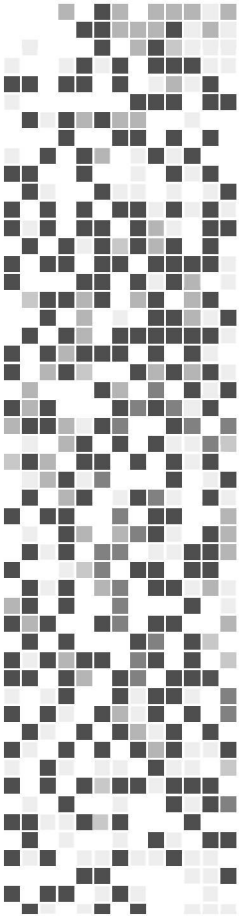

**Рисунок 1.5 —** IRQ клавіатури

## Запитання і завдання до розділу

1. Визначте, які IRQ зіставлені контролерам таких зовнішніх пристроїв: відеоадаптерів, IDE, USB, мережних плат і хост контролерів.

2. Виберіть з таблиці за номером вашого варіанта API-функцію й опишіть її.

| Номер варіанта | API-функція | Номер варіанта | API-функція |
|---|---|---|---|
| 1 | GetDriveType | 9 | GetVersionEx |
| 2 | GetVolumeInformation | 10 | PostMessage |
| 3 | GetDiskFreeSpace | 11 | GetActiveWindow |
| 4 | GetSystemDirectory | 12 | GetDriveType |
| 5 | GetWindowsDirectory | 13 | GetVolumeInformation |
| 6 | GetTempFileName | 14 | GetSystemInfo |
| 7 | GetSystemInfo | 15 | GetWindowsDirectory |
| 8 | GlobalMemoryStatus | 16 | GetACP |



# 2

# Процеси

Розділ присвячений фундаментальному поняттю операційних систем — "процесу". Це основний динамічний об'єкт, над яким системи виконують певні дії. Розглянуто проблему "перегони", що виникає за відсутності синхронізації потоків. Поняття "критична секція" та "мьютекс" у контексті вирішення завдання синхронізації роботи процесів. Виклад матеріалу супроводжується практичним ознайомленням з потоками шляхом їхнього дослідження за допомогою інструментарію сторонніх виробників, а також програмних розробок.

Розділ 2

# Процеси

Процес — власник системних ресурсів, коду та даних виконуваної програми. Процес не виконується процесором. Виконується потік процесу. Процесів може бути декілька й тоді виникає проблема синхронізації їхніх дій. "Побачити" процеси, потоки та вирішення проблеми їхньої синхронізації — це завдання цього розділу.

## 2.1 Поняття "процесу"

На найвищому рівні абстракції процес (process) є власником такого:

- коду та даних виконуваної програми;
- закритого адресного простору (address space), тобто набору адресів віртуальної пам'яті, що може використати процес;
- системних ресурсів (семафори, комунікаційні порти й файли), які виділяються ОС процесу під час виконання програми.

У Windows процес, щоб він міг бути виконаний, повинен включати четвертий елемент — як мінімум один потік (thread of execution).

Використовуючи утиліту Process Explorer [5], ви можете подивитися список процесів (рис 2.1).

Process Explorer в основному вікні дозволяє подивитися ідентифікатор процесу (PID), зайнятість процесора (CPU), кількість потоків у процесі (threads) і т. д.

Для детального перегляду інформації про процес досить клацнути правою клавішею миші на цьому потоці й потім вибрати з контекстного меню **Properties** (рис 2.2).

Тут можна побачити базовий пріоритет потоку (Priority), час виконання потоку в режимі ядра Windows (Kernel Time), користуваць-

кому режимі (User Time), використовувану віртуальну й фізичну пам'ять та інші дані.

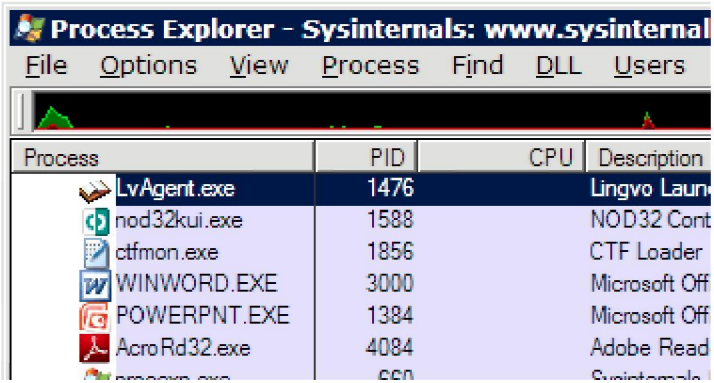



**Рисунок 2.1** — Список процесів

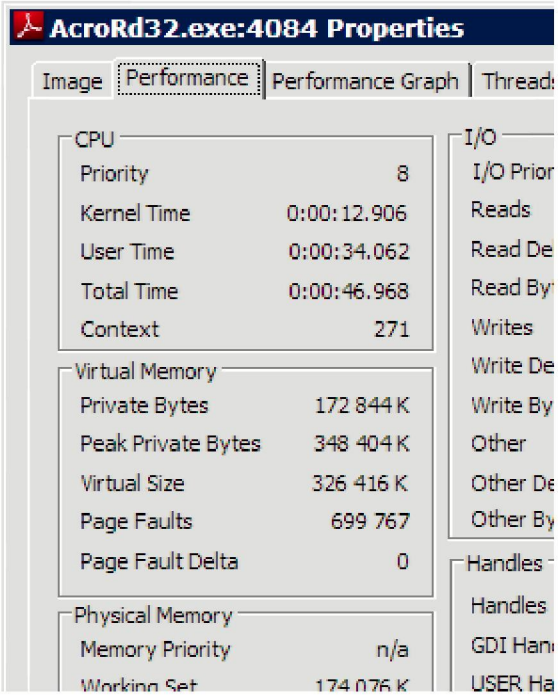



**Рисунок 2.2** — Властивості потоку

## 2.2 Поняття "потоку"

Потоки можуть перебувати в трьох станах (рис. 2.3).


**Рисунок 2.3** — Стани потоків

Опишемо стани потоків. По-перше, потік може виконуватися (running), коли йому виділене процесорний час. По-друге, він може бути у готовності (ready) очікувати, коли йому "виділять" процесор. Є ще третій стан — "блокування" (waiting). Звичайно процес блокують, чекаючи певної події. При виникненні цієї події потік автоматично переводиться зі стану блокування у стан готовність. Наприклад, якщо один потік виконує обчислення, а іншій повинен чекати результатів, щоб зберегти їх на диску.

Можна подивитися стани потоків, використовуючи програму Process Explorer (рис 2.4).

Для перегляду станів потоків можна використати будь-який процес. Але оскільки не всякий процес допускає прозоре керування станами його потоків, то ми розробимо програму, яка в одному потоці дозволить редагувати дані, а в іншому — відображати випадкове число.

- Створіть новий Delphi-проект і збережіть Unit1 як Main.pas.

### Додавання потоку до Delphi-проекту

Для додавання потоку успадковуємо новий клас TThread від базового класу TThread, використовуючи Object Repository:

- Скомандуйте **File/New/Other**.
- Виберіть значок **Thread Object** у діалоговому вікні, що з'явило-

ся, і клацніть на кнопці **OK** (рис. 2.5).

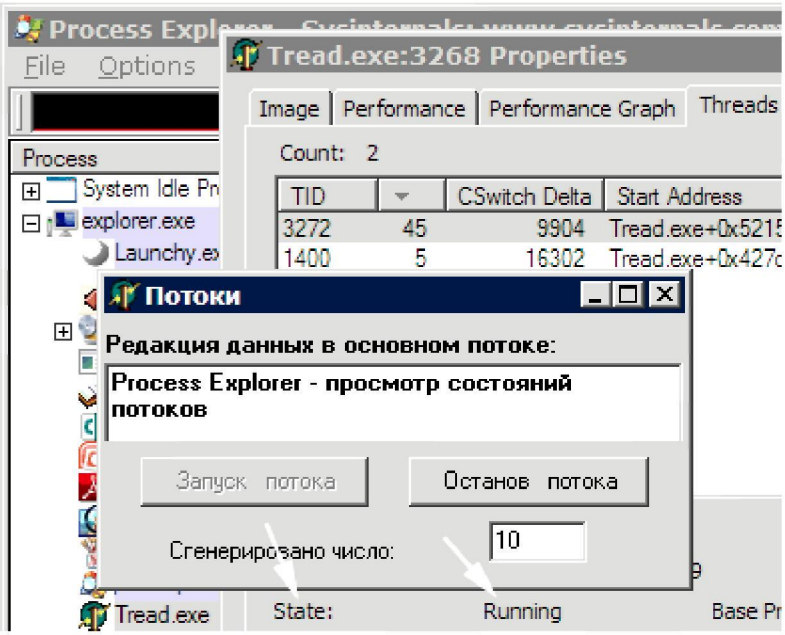



**Рисунок 2.4** — Перегляд станів потоків програмою Process Explorer

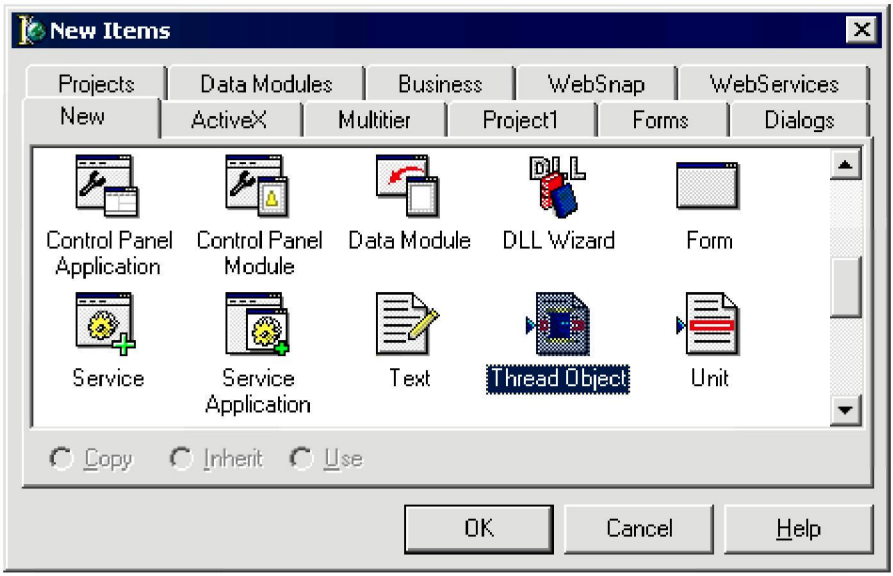



Рисунок 2.5 — Object Repository

- Введите имя `TPainterThread` нащадка класу `TThread` у вікні **New Thread Object** (рис. 2.6).


**Рисунок 2.6 —** Вікно **New Thread Object**

- Збережіть Unit1 як ThrdU.pas.

### Запуск потоку

Після оголошення потоку для його запуску викликають його конструктор `Create`. Конструктор класу типу `TThread` має такий синтаксис:

```
Create(Suspended : Boolean);
```

У цьому випадку при виклику конструктора йому передається один Boolean параметр — `Suspended`, який показує, що потік створений у припиненому стані. Звичайно передають значення `False`, щоб потік почав виконуватися негайно.

- Додайте на форму кнопку й змініть її властивість **Caption** на **Запуск потока**, а **Name** на **btStart**.
- Поставте їй у відповідність такий оброблювач подій:

```
procedure TForm1.btStartClick(Sender: TObject);
begin
    btStart.Enabled := False;
    btStop.Enabled := True;
    PT := TTestThread.Create(False);
end;
```

Тут `PT` — `private` змінна класу `TForm`. Її тип `TTestThread`.

### Зупин потоку

- Додайте на форму кнопку й змініть її властивість **Caption** на **Останов потока**, а **Name** на **btStop**.
- Поставте їй у відповідність такий оброблювач подій:

```
procedure TForm1.btStopClick(Sender: TObject);
begin
    btStart.Enabled := True;
    btStop.Enabled := False;
    PT.Free;
end;
```

### Розміщення коду для фонової задачі

Клас `TThread` визначає віртуальний абстрактний метод `Execute`. Його код виконується після запуску потоку. Конструктор `TThread` викликає `Execute` негайно після створення потоку. Тому немає необхідності запускати його явно. Коли `Execute` виконано, то потік завершується й установлює повертає значення, яке можна запросити, використовуючи властивість `ReturnValue`.

Код, що ви розміщуєте в `Execute`, може виконувати певне фонове завдання, наприклад, обробку великого масиву даних. Якщо код перебуває в циклі, цикл повинен містити деяку визначену умову, що перериває цикл, якщо умова істинна (`True`). Наприклад, якщо певну кількість повторів уже зроблено, ви повинні також помістити перевірку усередині циклу, щоб бачити, чи встановлена властивість Terminated у `True`. Якщо так, ваш код повинен негайно завершувати метод `Execute`.

- Додайте метод `Execute`:

```
procedure TTestThread.Execute;
begin
   Randomize;
   while (not Terminated) do begin
     X := Random(128);
      end;
```

```
end;;
```

Цей код дозволяє випадково генерувати число X у межах 128.

### Використання VCL-компонента усередині потоку

Після того, як число X згенеровано, його можна відобразити на формі таким чином:

- Помістіть на форму компонент TEdit.
- Додайте до модуля потоку визначення:
  - `private` змінної `X` як `Integer`;
  - процедури `DispX` в об'єкті `TTestThread` (слідом за процедурою `Execute`).
- Опишіть у методі `DispX` відображення числа `X` таким кодом:

```
procedure TTestThread.DispX;
begin
    Form1.Edit1.Text := InttoStr(X);
end;
```

- Забезпечте потоко-безпечний спосіб запуску методу `DispX`:

```
procedure TTestThread.Execute;
begin
   Randomize;
   while (not Terminated) do begin
     X := Random(128);
     Synchronize(DispX);
  end;
end;
```

Зміни, які необхідно внести до методу Execute, відображені курсивом.

У методі `Execute` об'єкта `Thread`, цикл `while` повторюється до того часу, доти метод `Terminated` не набуде значення `False`. Усередині циклу запит зроблений до методу `DispX`. Потім цей потік повинен модернізувати зображення компонента `Form`. Щоб

зробити це, він викликає `Synchronize` і передає `DispX` як аргумент.

На закінчення розробки програми:

- помістіть на форму компонент `TMemo` і додайте в секцію модуля `Main` — **ThrdU**, а в секцію `implementation` модуля `ThrdU` — **uses Main, SysUtils**.

Код проекту ви можете завантажити із сайта автора [3].

Тепер можна поекспериментувати із програмою: запустити фоновий потік і ввести текст у "вікні" основного потоку. Зупинити фоновий потік. У процесі цих маніпуляцій ви можете спостерігати за станами потоків (рис. 2.7).

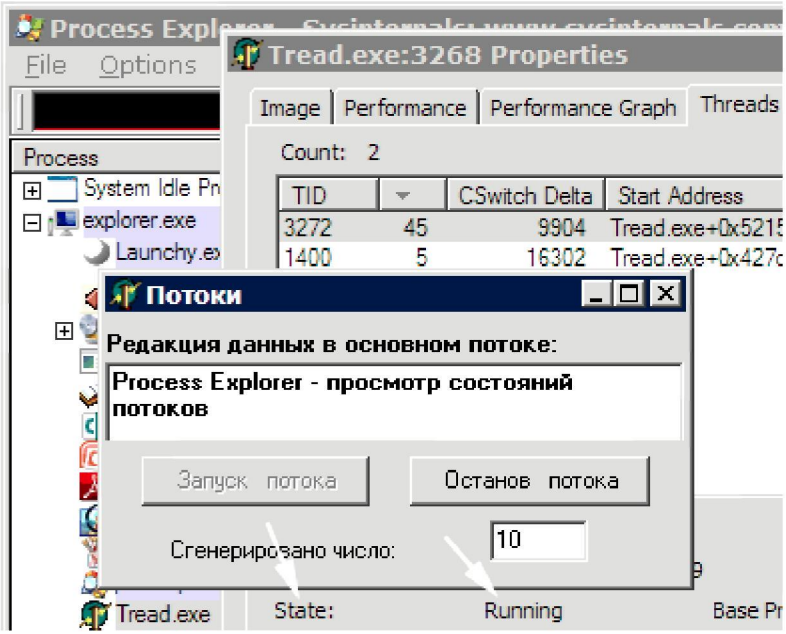



**Рисунок 2.7 —** Перегляд станів потоків програмою Process Explorer

## 2.3 Засоби синхронізації потоків

Розробляючи багатопотокову програму в попередньому параграфі, ми використали метод `Synchronize`, що дозволяє уникати конфліктів конкуренції потоків.

Покажемо, що конфлікти конкуренції потоків ("перегони") приводять до непередбачених результатів роботи програм.

Нехай необхідно ініціювати масив `GlobalArray[1..128]` у першому потоці `TFooThread` цілими числами від 1 до 128 із кроком 1. Після ініціалізації ще раз ініціювати цей масив у другому потоці числами від 129 до 256.

- Створіть новий Delphi-проект.
- Слідом за оголошенням нащадка класу `TForm` створіть новий екземпляр потоку:

```
TFooThread = class(TThread)
protected
  procedure Execute; override;
end;
```

- Помістіть на форму кнопку й поставте їй у відповідність такий оброблювач подій:

```
procedure TForm1.CreateThrd(Sender: TObject);
begin
    TFooThread.Create(False);
    TFooThread.Create(False);
end;
```

- Опишіть у методі `Execute` фоновий потік таким кодом:

```
procedure TFooThread.Execute;
var
  i: Integer;
begin
 { після виконання потоку, перед його завершенням необ-
хідно виконати процедуру }
  OnTerminate := Form1.DispGlblArr;
  for i := 1 to MaxSize do
  begin
    GlobalArray[i] := GetNextNumber;
    Sleep(5); { призупинити виконання потоку на
 5 мілісекунд }
```

```
  end;
end;
```

- Визначте функцію `GetNextNumber`:

```
function GetNextNumber: Integer;
begin
  Inc(NextNumber);
  Result := NextNumber;
end;
```

- Визначте константу й змінні так:

```
const
  MaxSize = 128;
var
  NextNumber: Integer = 0;
  DoneFlags: Integer = 0;
  GlobalArray: array[1..MaxSize] of Integer;
```

І, нарешті, остання дія другого потоку перед його завершенням — відображення за допомогою компонента `TListBox` результату заповнення масиву `GlobalArray`:

```
procedure TForm1.DispGlblArr(Sender: TObject);
var
  i: Integer;
begin
  Inc(DoneFlags);
  if DoneFlags = 2 then { виконується тільки перед завер-
шенням другого потоку }
    for i := 1 to MaxSize do
 { заповнення ListBox значеннями масиву GlobalArray[i]}
      ListBox1.Items.Add(IntToStr(GlobalArray[i]));
end;
```

> **Попередження:** Не забудьте оголосити процедуру `DispGlblArr` як `private`-змінну об'єкта `TForm1`.

- Збережіть проект під ім'ям hurry.dpr, відкомпілюйте й запустіть його.

Код проекту ви можете завантажити із сайта автора [3].

Якби програма працювала так, як ми її задумували, то результат був би таким (рис. 2.8).

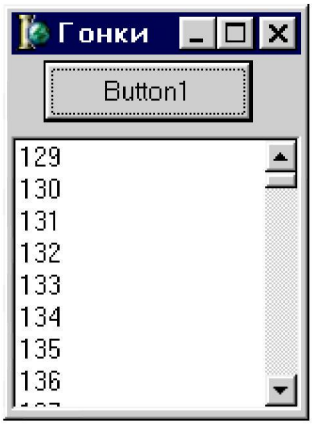



Рисунок 2.8 — Планований результат роботи програми

Однак він виглядає інакше (рис. 2.9):

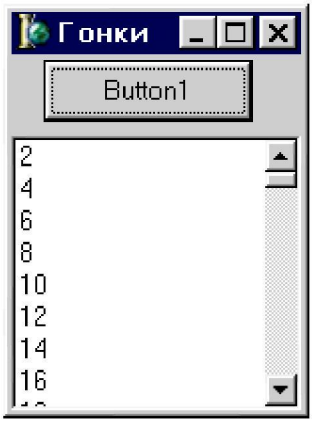



Рисунок 2.9 — Результат роботи програми

### 2.3.1 Критична секція

Для того щоб уникнути "гонок", необхідно дозволити одночасно тільки одному процесу виконувати його критичну секцію коду, тобто секцію коду, що здійснює доступ до послідовно використовуваного ресурсу.

У попередньому параграфі послідовно використовуваний ресурс — це масив `GlobalArray`. Подивимося, як ініціалізувати масив `GlobalArray[1..128]` у першому потоці `TFooThread` цілими числами від 1 до 128 із кроком 1. Після ініціалізації ще раз ініціювати цей масив у другому потоці числами від 129 до 256, використовуючи механізм "критична секція коду".

- Відкрийте Delphi-проект hurry.dpr.
- Ініціюйте в процедурі створення потоків критичну секцію функцією Win32 API `InitializeCriticalSection(CS)`. Нижче вона виділена курсивом:

```
procedure TForm1.CreateThrd(Sender: TObject);
begin
    InitializeCriticalSection(CS);
    TFooThread.Create(False);
    TFooThread.Create(False);
end;
```

- Додайте в оголошення змінних параметр `CS`, що являє собою запис типу `TRTLCriticalSection`:

```
var
   . . .
  GlobalArray: array[1..MaxSize] of Integer;
 CS: TRTLCriticalSection;
```

Запис `CS` передається за посиланням.

А тепер саме головне — оголошення початку й кінця критичної секції:

```
procedure TFooThread.Execute;
var
  i: Integer;
begin
 { після виконання потоку перед його завершенням необхідно виконати процедуру }
  OnTerminate := Form1.DispGlblArr;
 EnterCriticalSection(CS); { оголошення початку критичної секції }
  for i := 1 to MaxSize do
  begin
    GlobalArray[i] := GetNextNumber;
    Sleep(5); {призупинити виконання потоку на 5 мілісекунд}
  end;
 LeaveCriticalSection(CS);{ оголошення кінця критичної секції }
end;
```

Зрозуміло, що запис `CS` заповнюється функцією Win32 API `InitializeCriticalSection`.

- І, нарешті, коли запис `CS` не потрібна, необхідно звільнити займану нею пам'ять:

```
procedure TForm1.DispGlblArr(Sender: TObject);
var
  i: Integer;
begin
  Inc(DoneFlags);
  if DoneFlags = 2 then {виконується тільки перед завершенням другого потоку}
    for i := 1 to MaxSize do
 {заповнення ListBox значеннями масиву GlobalArray[i]}
     ListBox1.Items.Add(IntToStr(GlobalArray[i]));
   DeleteCriticalSection(CS);
end;
```

- Збережіть програму під ім'ям `CritSec`, запустіть її і ви побачите, що вона працює так, як ми її задумували (рис. 2.10):

Код проекту ви можете завантажити із сайта автора [3].

### 2.3.2 Мьютекс

Щоб результати роботи потоків "не переплутувалися", можна використати альтернативу критичної секції — "спрощений" семафор. Його також називають мьютекс (MUTual EXclusion — взаємовиключення). По суті, — це ресурс, яким можуть володіти два потоки, які намагаються одночасно одержати доступ до одного розділеного ресурсу. Семафори використають для вирішення проблеми доступу до розділеного ресурсу великою кількістю процесів.

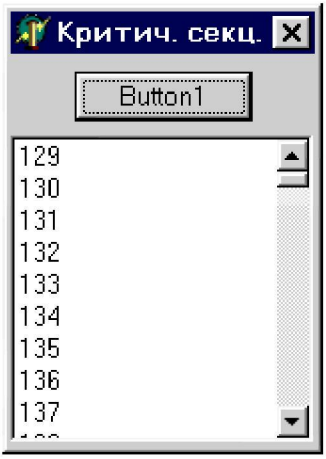



Рисунок 2.10 — Результат роботи програми

Мьютекс створюють Win32 API- функцією:

```
HANDLE CreateMutex(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  BOOL bInitialOwner,
  LPCTSTR lpName
);
```

параметри якої: `lpMutexAttributes` — покажчик на атрибут безпеки, `bInitialOwner` — флаг ініціалізації мьютекса і `lpName` — покажчик на ім'я мьютекс-об'єкта.

### Розроблення програми

- Відкрийте Delphi-проект CritSec.dpr.
- Створіть мьютекс як показано курсивом у процедурі створення потоків:

```
procedure TForm1.CreateThrd(Sender: TObject);
begin
    hMutex := CreateMutex(nil, False, nil);
    TFooThread.Create(False);
    TFooThread.Create(False);
end;
```

Тут як значення параметра `lpMutexAttributes` передано значення `nil`, тому будуть використані атрибуту захисту, які діють за замовчуванням. Параметр `bInitialOwner` визначений як `False`, тобто створений мьютекс не буде мати власника. Значення параметра `lpName` не визначено, отже, створений мьютекс буде безіменним.

- Додайте в оголошення змінних параметр `hMutex`:

```
var
   . . .
  GlobalArray: array[1..MaxSize] of Integer;
hMutex: THandle = 0;
```

Запис `CS` передається за посиланням.

- А тепер саме головне — захоплення й звільнення мьютекса:

```
procedure TFooThread.Execute;
var
  i: Integer;
begin
 { після виконання потоку перед його завершенням необхідно виконати процедуру }
  OnTerminate := Form1.DispGlblArr;
 if WaitForSingleObject(hMutex, INFINITE) =
WAIT_OBJECT_0
```

```
 begin
   for i := 1 to MaxSize do
   begin
      GlobalArray[i] := GetNextNumber;
      Sleep(5); { призупинити виконання потоку на
 5 мілісекунд }
    end;
  end;
ReleaseMutex(hMutex);
end;
```

Для керування входом потоків у блок синхронізації використаємо функцію `WaitForSingleObject(hMutex, INFINITE)`. Ця функція переводить поточний об'єкт у стан очікування, до того часу, доти не стане доступним мьютекс, що визначений параметром `hMutex`. При цьому стан очікування визначає другий параметр функції `WaitForSingleObject`. Він визначений як `INFINITE`, тобто очікувати до того часу, доти об'єкт не стане доступним. Інакше кажучи, оскільки створений мьютекс не мав власника, то перший же потік, що викликає функцію `WaitForSingleObject`, стає його власником до того часу, доти він не буде звільнений функцією `ReleaseMutex`.

- І, нарешті, коли мьютекс не потрібний, необхідно закрити його, використовуючи функцію `CloseHandle`:

```
procedure TForm1.DispGlblArr(Sender: TObject);
var
  i: Integer;
begin
  Inc(DoneFlags);
  if DoneFlags = 2 then {виконується тільки перед завер-
шенням другого потоку}
    for i := 1 to MaxSize do
 {заповнення ListBox значеннями масиву GlobalArray[i]}
      ListBox1.Items.Add(IntToStr(GlobalArray[i]));
  CloseHandle(hMutex);
end;
```

- Запустіть програму, і ви побачите, що вона працює так, як ми її задумували (рис. 2.11).

Код проекту ви можете завантажити із сайта автора [3].

## Запитання і завдання до розділу

1. Опишіть події, які можна спостерігати на вкладці **Performance Graph** контекстного меню **Properties** програми Process Explorer до й після закриття документів у програмі MS Word.
2. Поясніть, чому результат роботи програми "перегони" такий, як зображений на рис. 2.9. Можете скористатися підказуваннями, які наведені на сайті автора [3].

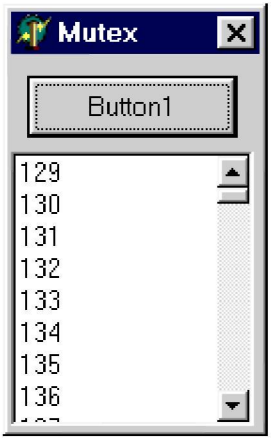


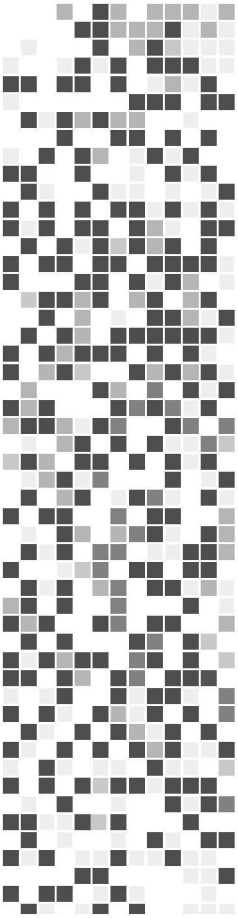

Рисунок 2.11 — Результат роботи програми

3. Чим керує планувальник ОС Windows:
   а) threads;
   б) fibers;
   в) processes?
4. До класу яких систем відносять ОС Unix:
   а) багатозадачних;
   б) персональних;
   в) багатокористувацьких?

5. Який тип багатозадачності використовується в ОС Linux:
   а) що витісняє;
   б) що не витісняє?

6. Виберіть з таблиці за номером вашого варіанта API-функцію й опишіть її.

| Номер варіанта | API-функція | Номер варіанта | API-функція |
|---|---|---|---|
| 1 | InitializeCriticalSection | 9 | CloseHandle |
| 2 | EnterCriticalSection | 10 | FindClose |
| 3 | DeleteCriticalSection | 11 | CreateThread |
| 4 | LeaveCriticalSection | 12 | GetThreadPriority |
| 5 | TryEnterCriticalSection | 13 | SetThreadPriority |
| 6 | CreateMutex | 14 | ExitThread |
| 7 | WaitForSingleObject | 15 | GetExitCodeThread |
| 8 | ReleaseMutex | 16 | ExitProcess |



# 3

# Взаємодія процесів

Описано взаємодію процесів (Inter-Process Communication — IPC). Вона має місце як між процесами одного комп'ютера, так і між процесами на різних комп'ютерах. Тут (із залученням інструментарію сторонніх виробників, а також у процесі розроблення програм) розглянутий обмін даними: повідомлення Windows, мейлслоти та іменовані канали. Не обійдені увагою й такі специфічні механізми взаємодії процесів, як "файли, що проектуються в пам'ять" і протокол Component Object Model (COM).

Розділ 3

# Взаємодія процесів

Взаємодія процесів (Inter-Process Communication — IPC) має місце як між процесами одного комп'ютера, наприклад, для обміну даними між нитями процесів, так і між процесами на різних комп'ютерах, наприклад, системи баз даних.

У табл. 3.1 наведені механізми взаємодії Windows процесів, які розглядаються у даному розділі.

***Таблиця 3.1*** *— Механізми IPC, передбачені у Windows*

| Механізм IPC | Опис |
|---|---|
| Системні повідомлення | Кращий спосіб пересилання даних від одного процесу до іншого |
| Мейлслоти | Корисні для організації зв'язку одного процесу з багатьма на одному комп'ютері або в мережі |
| Іменовані канали (Named pipes) | Корисні для організації двобічного зв'язку між двома процесами на одному комп'ютері або в мережі |
| Файли, що проектуються в пам'ять | Забезпечують одночасний доступ до об'єктів файла відображення з декількох процесів |
| COM (Component Object Model) | Є вбудовані функції для пересилання даних через межі процесів |

## 3.1 Повідомлення Windows

Windows для керування процесами використовує модель повідомлень. Вони генеруються щораз, коли відбувається якась подія. Наприклад, користувач натискує клавішу клавіатури або поміщає дані в буфер. Повідомлення міститься в так звану чергу повідомлень (message queue). Активний потік постійно перевіряє свою чергу й витягує з її повідомлення, що надійшли (рис. 3.1).

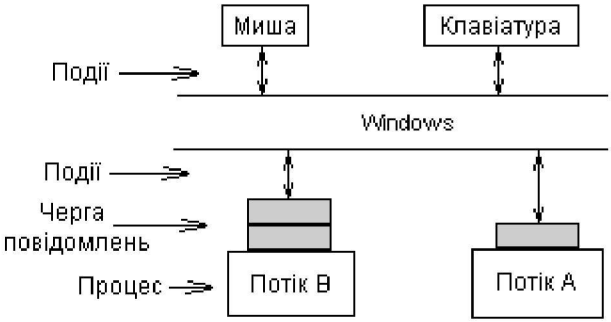



Рисунок 3.1 — Обробка повідомлень

Повідомлення являє собою спеціальний запис, переданий додатку системою Windows. Повідомлення складається із чотирьох частин. Ось приклад посилання повідомлення додатку про необхідність згорнути його:

```
SendMessage(OtherAppHandle, WM_SYSCOMMAND,
                        SC_MINIMIZE, 0);
```

Перша частина повідомлення — це дескриптор вікна, що повинне його одержати. Windows зберігає дескриптори більшості візуальних об'єктів так, що вона може звернутися до будь-якого об'єкта керування, хоча у вищенаведеному прикладі (посилання команди згорнути) це дескриптор форми. Другий елемент — це команда, що буде послана. Треті й четверті частини повідомлення — це параметри повідомлення типу `LParam` і `WParam`. Деякі повідомлення не використають ці параметри (наприклад, `WM_CLOSE` — завершення програми), у той час як інші використають один або обидва параметри, щоб передати додаткову інформацію.

Є три методи надсилання Windows-повідомлення, але тільки два з них використаються для IPC — це `SendMessage()` і `PostMessage()`. Вони є API-функціями Windows і, по суті, однакові. Однак SendMessage() синхронна (надсилає повідомлення й очікує до закінчення його обробки), а PostMessage() — асинхронна (надсилає повідомлення в чергу повідомлень Windows і повертається, не очікуючи закінчення його обробки). Яку з них використати, залежить від розв'язуваного завдання.

## Системні повідомлення

Як приклад розглянемо процедуру обробки зміни часу, коли Windows оновлює системний годинник для переходу на літній час. Щоб оповістити нас про це, програма повинна перехопити повідомлення `WM_TIMECHANGE`.

Спочатку додамо в секцію `private` розділу `interface` форми таку декларацію процедури обробки повідомлення:

```
procedure WMTimeChange(var Msg: TWMTimeChange);
   message WM_TIMECHANGE;
```

Код процедури (лістинг 3.1) потрібно додати до розділу `implementation` форми. Вона відображує повідомлення всякий раз, коли змінюється системний час. Переконайтеся в цьому, змінюючи час у третє.

**Лістинг 3.1.** Відображення діалогового вікна у разі зміни системного часу

```
procedure TForm1.WMTimeChange(var Msg: TWMTimeChange);
begin
  // Ця процедура виконується, якщо Windows надсилає
  // повідомлення WM_TIMECHANGE, що показує, що
  // системний час змінився
  inherited;
  if (Msg.Msg = WM_TIMECHANGE) then
    begin
      ShowMessage('Увага! Системний час змінено');
      // См. help: "Програма повинна повернути нуль,
      // якщо вона обробляє повідомлення"
      Msg.Result := 0;
    end;
end;
```

Використання директиви `inherited` у цій процедурі дозволяє передати повідомлення обробника повідомлення, який належить класу-предкові. У нашому випадку це обробник повідомлення `WM_TIMECHANGE` класу `TForm`. Якщо він повертає повідомлення

WM_TIMECHANGE, тоді генерується форма з повідомленням 'Увага! Системний час змінено'.

### Приватні повідомлення

Окрім розглянутої обробки стандартних Windows-повідомлень, можна надіслати свої повідомлення.

Для цього досить пам'ятати, що Windows створює повідомлення, розміщує їх у цикл повідомлень (message loop), а потім "розсилає" їх потокам. Тому якщо ви хочете надіслати повідомлення іншого потоку, то заздалегідь його слід помістити у цикл повідомлень Windows. Для цього необхідно оголосити константу, аналогічну стандартним константам. Наприклад,

```
const
  WM_MyDelphiMessage = WM_USER + 100;
```

і відправити повідомлення, використовуючи API-функції SendMessage() або PostMessage(), наприклад,

```
PostMessage(Form1.Handle, WM_MyDelphiMessage, 0, 0);
```

Для того щоб після відправлення повідомлення потік, якому воно адресоване, перехопив його, створимо метод

```
procedure WMMyDelphiMessage(var Msg: TMessage);
   message WM_MyDelphiMessage;
```

Ви можете ознайомитися з деталями проекту Messages на сайті автора [3].

Зверніть увагу на використання типу TMessage. Немає такого типу, як TWMMyDelphiMessage, так що краще передати об'єкт TMessage, який являє собою повідомлення Windows.

Є вірогідність того, що окрім процесу, якому ви надсилаєте повідомлення, є й інші процеси, що перехоплюють це повідомлення, але ви про них не знаєте. Щоб запобігти цьому, використовуйте RegisterWindowMessage. Win32 Developer Reference описує її як функцію, що "визначає нове гарантовано унікальне віконне повідомлення в рамках системи". Просто визначте змінну типу

LongWord (або UINT, щоб використати С-структуру) і присвойте їй результат функції RegisterWindowMessage:

```
Var
  iMyDelphiMessageID: LongWord;
  ...
  iMyDelphiMessageID :=
    RegisterWindowMessage(PChar('Моє Delphi
                                повідомлення'));
```

Ви можете ознайомитися з деталями проекту RegisterWindowMessage на сайті автора [3].

Єдине обмеження розглянутих механізмів передачі повідомлень — це те, що вони передаються в рамках однієї Windows-сесії, іншими словами, їх не можна використовувати для зв'язку між процесами в мережі. Зараз ми розглянемо Windows API, які знімають ці обмеження.

### Спілкування в мережі

Windows API дозволяє надіслати повідомлення інакше: використовуючи функцію BroadcastSystemMessage. Один з параметрів цієї функції — Recipients. У табл. 3.2 наведено список значень, які він може взяти (див. довідку по Windows SDK). Одне з них — BSM_APPLICATIONS — дозволяє відправити повідомлення всім комп'ютерам мережі. Це аналог параметра HWND_TOPMOST функції PostMessage().

***Таблиця 3.2*** *— Значення параметрів* *Recipients*

| Значення параметра | Опис |
|---|---|
| BSM_ALLCOMPONENTS | Передача повідомлень всім компонентам системи |
| BSM_ALLDESKTOPS | Тільки для Windows NT: передача повідомлень всім комп'ютерам мережі. Вимагає SE_TCB_NAME привілею |
| BSM_APPLICATIONS | Передача повідомлень всім комп'ютерам мережі |

Альтернативний параметр BSM_ALLDESKTOPS дозволяє надіслати повідомлення в рамках мережі. Однак його можна використати тільки на платформі Windows NT за наявності в процесу привілеїв. Як же надіслати повідомлення від не Windows NT-комп'ютера іншим комп'ютерам у мережі? Відповідь одна — це поштові слоти (мейлслоти).

## 3.2 Мейлслоти

Мейлслоти (поштові слоти) — це механізм однобічного зв'язку процесів.

Мейлслоти реалізують в архітектурі клієнт-сервер. Один комп'ютер, що називають мейлслот-сервером, створює мейлслот й очікує повідомлення. Другий комп'ютер — мейлслот-клієнт — може створювати повідомлення й зберігати його у мейлслоті сервера. Поки процес володіє ім'ям мейлслота, він може помістити в нього повідомлення. Перевага мейлслота — його здатність передати повідомлення у межах домену. Клієнт може вибрати між надсиланням повідомлення конкретному комп'ютеру-домену або кожному комп'ютеру-домену. Кожен комп'ютер домену, що створив мейлслот з даним ім'ям (тобто мейлслот-сервер), здатний читати повідомлення у мейлслоті.

Мейлслот може бути корисним, наприклад, для модифікацій стану потоку. Наприклад, потік відстежує присутність службовців на робочому місці. Як тільки службовець поміщає свій жетон у турнікет, потік записує час і його номер у базу даних. Потік також надсилає повідомлення мейлслоту, названому "presence". Якщо мейлслот-клієнт знає ім'я мейлслота-сервера, то він може звернутися до нього безпосередньо:

```
Var
  HndMailSlot: THandle;
  HndMailSlot :=
    CreateFile(PChar('\\OtherComputerName\mailslot\
         presence'), GENERIC_WRITE, FILE_SHARE_READ, nil,
           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
```

Формати імен мейлслотів наведені у табл. 3.3.

***Таблиця 3.3.*** *— Формати імен мейлслотів*

| Формат | Опис |
|---|---|
| \\.\mailslot\name | Для взаємодії із процесом на тому самому комп'ютері |
| \\computername\mailslot\name | Для взаємодії із процесом на віддаленому комп'ютері |
| \\domainname\mailslot\name | Для взаємодії із процесами на віддалених комп'ютерах у рамках домену |
| *\mailslot\name | Для взаємодії із процесами на віддалених комп'ютерах у рамках системного первинного домену |

Відзначимо, що API-функція `CreateFile()` використана для того, щоб мейлслоту призначити дескриптор. Це зроблено тому, що, по суті, *мейлслоти є тимчасовими псевдофайлами*, що розміщуються в ОЗУ й доступні за допомогою стандартних функцій для роботи з файлами. Якщо мейлслоту-клієнту відомий дескриптор, то він використовує функцію `WriteFile()`,щоб надіслати повідомлення стандартним способом. Щоб читати повідомлення з мейлслота, сервер так само використовує стандартну функцію `ReadFile()`.

Для створення мейлслота-сервера використовують майже такий самий код, як і для клієнта, за винятком того, що замість API-функції `CreateFile()` використовують `CreateMailSlot()`:

```
var
  HndMailSlot: THandle;
  HndMailSlot :=
    CreateMailSlot(PChar('\\.\mailslot\presence'),
                           0, 0, nil);
```

Перший параметр — це ім'я мейлслота, але, крім нього, повинне бути наявним й `\\.\mailslot\`. Другий параметр — це максимальна довжина повідомлення (0 означає "необмежено"), за ним іде число мілісекунд, що операція читання "чекає" операцію за-

пису. Заключний параметр — це покажчик на структуру безпеки (SECURITY_ATTRIBUTES), якщо така потрібна.

Функція CreateMailSlot() також повертає дескриптор файла, що використовує сервер для того, щоб витягти повідомлення. Оскільки мейлслоти є файлами, вони не здатні надіслати повідомлення про те, що повідомлення було отримано. Тому сервер звичайно читає мейлслот, використовуючи функцію ReadFile() не відразу, а через заданий GetMailSlotInfo() часовий інтервал (лістинг 3.2). GetMailSlotInfo() відшукує інформацію про мейлслот, визначає, чи містить він повідомлення, що очікує прочитання. Кожне звертання до CreateMailSlot() або CreateFile() повинне мати відповідний виклик CloseHandle(),щоб звільнити мейлслот, коли в ньому більше немає необхідності.

**Лістинг 3.2.** Читання мейлслота в оброблювачі події OnTimer

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
  iMsgSize: DWord;
  iRead: DWord;
  sMsg: string;
begin
  //Параметри GetMailSlotInfo():дескриптор мейлслота,
  //адреса макс. довжини повідомлення, адреса розміру
  //наступного повідомлення, адреса кількості повідомлень
  //й адреса часу очікування. Якщо значення iMsgSize
  //відмінне від MAILSLOT_NO_MESSAGE, то повідомлення є.
   if not (GetMailslotInfo(HndMailSlot, nil,
                           iMsgSize, nil, nil)) then
     raise Exception.Create(
       'Одержати інформацію від мейлслота не вдалося');
   if (iMsgSize <> MAILSLOT_NO_MESSAGE) then
     begin
      SetLength(sMsg, iMsgSize);
      if not (ReadFile(HndMailSlot, sMsg[1],
                       iMsgSize, iRead, nil)) then
         raise Exception.Create('Прочитати
                                мейлслот не вдалося');
```

```
      memMessages.Lines.Append(sMsg);
    end;
end;
```

На сайті автора [3] ви знайдете два проекти: MailSlotServer і MailslotClient, які зв'язуються один з одним. Змінивши перший параметр функції `CreateFile()` у MailslotClientForm.pas з \\.\mailslot\presence на *\mailslot\presence (точку на зірочку), ви можете надсилати повідомлення серверу, що може бути розміщений на іншому комп'ютері у вашому домену. На рис. 3.2 показано спілкування програм — сервера із клієнтом.

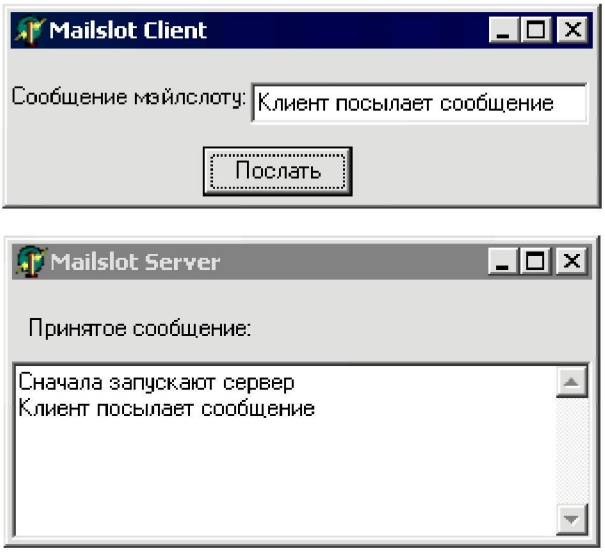



Рисунок 3.2 — Мейлслот-клієнт надсилає повідомлення мейлслот-серверу

Ви можете відстежити момент надсилання повідомлення мейлслот-серверу (рис 3.3), використовуючи програму Process Explorer.

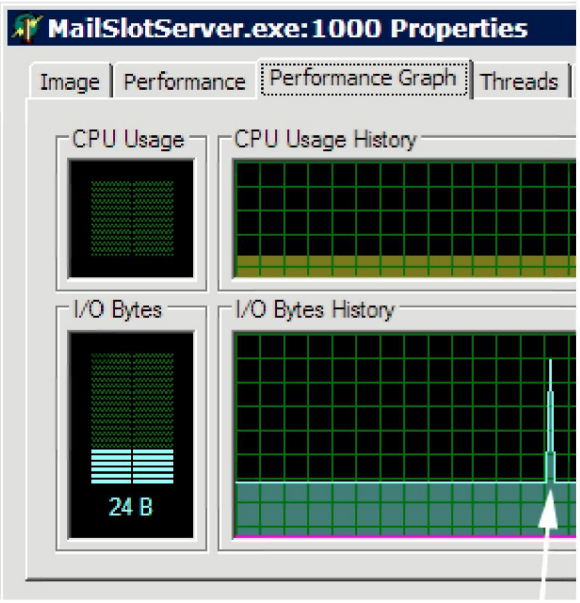



Рисунок 3.3 — Момент надсилання повідомлення мейлслот-серверу

Для цього достатньо:

- Запустити програми MailslotClient.exe та MailSlotServer.exe;
- у вікні Process Explorer вибрати MailSlotServer.exe;
- клацнувши правою клавішею миші, вибрати **Properties**, а потім вкладку **Performance Graph**;
- надіслати повідомлення мейлслот-серверу.

## 3.3 Іменовані канали

При передачі повідомлень на основі мейлслота використовують так звані датаграми — пакети інформації, факт одержання яких не гарантують і не перевіряють.

Якщо потрібна гарантія доставлення інформації або підтвердження цього факту, то Microsoft пропонує альтернативу — іменовані канали. Іменовані канали — це механізм, розроблений для обміну повідомленнями двох процесів.

Подібно мейлслоту сервер іменованого каналу створює канал з унікальним ім'ям, до якого можуть приєднатися клієнти. Кількість одночасних можливих підключень до каналу задається в момент його створення. Основна відмінність мейлслота від іменованого каналу в тім, що перший дозволяє передавати повідомлення комп'ютерам мережі, у той час як останній при цьому використовує блокування й розблокування.

Можна провести аналогію із синхронними й асинхронними повідомленнями. Блокування синхронне: спроба читання або запису з каналу не буде припинена до того часу, доти операція не буде завершена. Якщо буфер каналу порожній, метод `ReadFile` буде чекати, поки туди не помістять дані. Аналогічно якщо буфер повний, то метод `WriteFile` не буде виконаним до того часу, доти не з'явиться досить місця в буфері каналу для того, щоб записати дані. За замовчуванням іменований канал створюють як заблокований.

Заблокований канал, не очікуючи зміни режиму (розблокування), дозволяє методам `ReadFile` й `WriteFile` повернутися негайно. Однак користувач повинен перевірити повернуте `Boolean`-значення методу, щоб упевнитися у цьому. Наприклад, якщо `ReadFile` повернув `False` при читанні порожнього розблокованого каналу, користувач повинен викликати `GetLastError`, що поверне константу `ERROR_NO_DATA`.

Ім'я іменованого каналу на локальному комп'ютері зазначить у форматі `\\.\pipe\pipname`, а на іншій машині мережі — `\\servername\pipe\pipname`. Зверніть увагу на відсутність формату `\\*\pipe\pipname`, що дозволяв мейлслотам передавати повідомлення у рамках домену.

Метод `CreateNamedPipe` більш складний, ніж його аналог для мейлслота. Його вісім параметрів наведені в табл. 4. Повні описи значень цих параметрів можна знайти у розділі "Named Pipe Modes" Win32 Help.

***Таблиця 3.4*** — *Параметри* `CreateNamedPipe`

| Формат | Опис | Значення |
|---|---|---|
| lpName | Покажчик на ім'я каналу | \\.\pipename |
| dwOpenMode | Режим відкриття каналу. Визначає, чи надходять повідомлення від клієнта, сервера або обох | PIPE_ACCESS_INBOUND або PIPE_ACCESS_OUTBOUND або PIPE_ACCESS_DUPLEX |
| dwPipeMode | Визначає режими читання й запису | PIPE_TYPE_BYTE або PIPE_TYPE_MESSAGE PIPE_REAMODE_BYTE або PIPE_READMODE_MESSAGE PIPE_WAIT або PIPE_NOWAIT |
| nMaxInstances | Максимальна кількість екземплярів каналу | Від 1 до PIPE_UNLIMITED_INSTANCES |
| OutBufferSize | Розмір вихідного каналу в байтах | Тип Word |
| nInBufferSize | Розмір приймаючого буфера | Тип Word |
| nDefaultTimeOut | Час очікування в мілісекундах | Від 1 до NMPWAIT_USE_DEFAULT_WAIT |
| lpSecurityAttributes | Покажчик на структуру атрибутів безпеки SECURITY_ATTRIBUTES | nil часто використовується, щоб зазначити заданий за замовчуванням захист |

Створивши іменований канал, клієнт може з'єднатися з ним або викликати `CreateFile`, щоб одержати його дескриптор або використати функцію `CallNamedPipe`. Що використати, залежить від того, як необхідно зв'язатися з каналом. Перша з функцій поверне дескриптор, що дозволить багаторазово звертатися до каналу (читання, запис і т.д.). Остання виконує підключення, читання, запис і від'єднання в одній операції.

Є інші API-функції, застосовні до цієї структури. `WaitNamedPipe` дозволяє клієнтському додатку чекати, поки зазначений іменований канал не стане доступним (`CallNamedPipe` еквівалентний `WaitNamedPipe`, якщо немає необхідності відкрити канал негайно). `ConnectNamedPipe` — метод, що застосуємо до сервера. Він

дозволяє йому чекати, поки процес клієнта не з'єднається з іменованим каналом. `PeekNamedPipe` подібний `ReadFile`, за винятком того, що вміст каналу не вилучено. `TransactNamedPipe` виконує операцію читання й запису каналу, що вже має дескриптор, тобто той, який був створений функцією `CreateFile`. Функція `DisconnectNamedPipe` роз'єднує іменований канал із процесом, що створив сервер (`CreateNamedPipe`). Детальний опис цих функцій можна знайти в розділі "Named Pipe Operations" Win32 Help.

На сайті автора [3] ви можете знайти два проекти: сервер (NamedPipeServer) і клієнт (NamedPipeClient) іменованого каналу. Використовуючи програму Filemon.exe [6], можна спостерігати активність іменованих каналів:

- у меню **Volumes** File Monitor виберіть пункт **Named Pipes**;
- установіть режим спостереження **Options/Filter**, і зніміть прапорець із **Log Errors**;
- запустіть NamedPipeServer.exe й NamedPipeClient.exe;
- пошліть повідомлення серверу, й ви побачите, що останній його прийняв (**READ**).

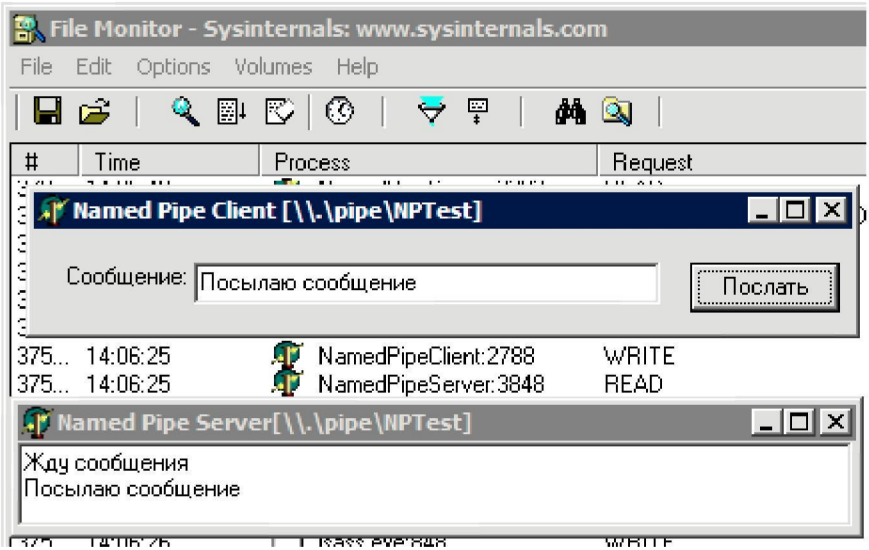



Рисунок 3.4 — Активність іменованого каналу

## 3.4 Файли, відображувані у пам'ять

Необхідність механізму "файли, відображувані у пам'ять", обумовлена потребою маніпулювати даними одного файла декільком клієнтам. У цьому випадку виникає ряд проблем. По-перше, це обмежені розміри фізичної пам'яті (ОЗУ). Отже, перемістити весь файл із ОЗУ у фізичну пам'ять для прискорення процесу маніпуляції його даними можливо не завжди. Залишається виділити частину фізичної пам'яті для маніпуляції необхідними в цей момент даними файла. Отже, відмовитися від думки маніпулювати даними файла у фізичній пам'яті й погодитися маніпулювати ними у самому файлі. Звичайно не безпосередньо, а за допомогою "вікна" у фізичній пам'яті.

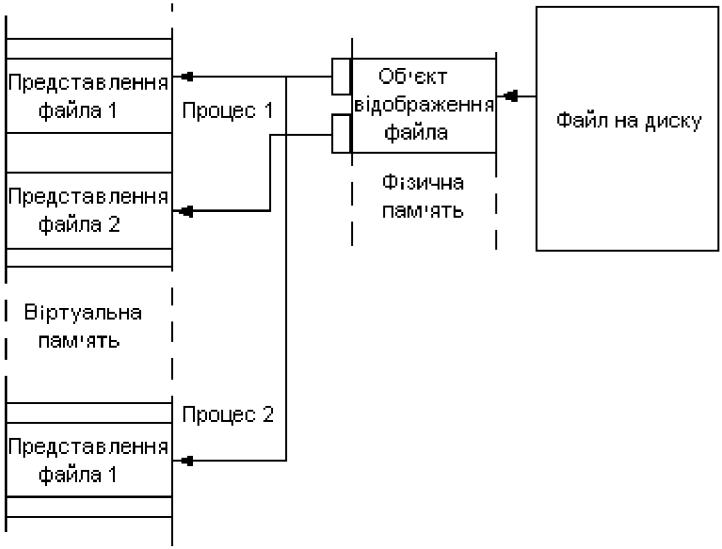



Рисунок 3.5 — Відношення між файлом, його поданням й об'єктом відображення

*Відображення файлів (file mapping)* — це зіставлення вмісту файла із частиною віртуального адресного простору процесу. Для супроводження цього зіставлення операційна система створює *об'єкт відображення файла*.

*Подання файла (file view)* — це частина віртуального адресного простору, що процес використовує для доступу до вмісту файла. Процеси читають і пишуть у подання файла, використовуючи покажчики так само, як і під час роботи з динамічно виділюваною пам'яттю. Зрозуміло, що в цьому випадку ми застраховані від втрати даних.

Win32 API-функції проектування файлів дозволяють процесу створювати об'єкти відображення файлів і подання файлів для простого доступу до файлів і поділу даних між процесами.

Об'єкт відображення може містити весь файл або його частину. Він пов'язаний з файлом на диску таким чином, що коли система вивантажує фізичні сторінки пам'яті об'єкта, те всі зміни, зроблені в об'єкті, зберігаються у файлі. Коли ж сторінки об'єкта знову знадобляться, то вони будуть завантажені з файла.

Процеси можуть маніпулювати файлом за допомогою подань файлів. Процес може створити кілька подань об'єкта відображення файла.

Коли кілька процесів використають той самий об'єкт відображення файла для створення подання локального файла, дані будуть когерентні (ідентичні). Файл не може розміщуватися на вилученому комп'ютері, якщо їхнє проектування полягає в поділі пам'яті між декількома процесами.

Для наочного ознайомлення з механізмом "файли, відображувані у пам'ять", розглянемо дві програми: одна з них створює файл, відображуваний у пам'ять, записує туди координати місця розміщення миші на його формі. Друга (клієнт) читає ці координати й відображує їх на своїй формі із частотою роботи таймера.

- Помістимо на форму Delphi-програми шість компонентів `Label` так, щоб форма виглядала так, як показано на рис. 3.6.

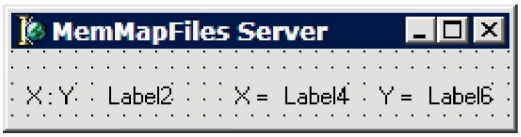



**Рисунок 3.6 —** Форма проекту `MemMap`

- Створимо об'єкт відображення файла в процедурі `FormCreate`:

```
hMapObj := CreateFileMapping(
       $FFFFFFFF, { Дескриптор відкритого файла (в ОЗУ)}
             Nil, { Атрибутів безпеки немає }
  PAGE_READWRITE, { Читання й запис }
               0, { Розмір об'єкта визначається
                    молодшими 32 бітами }
   SizeOf(DWORD), { Розмір об'єкта відображення файла }
 'shared_memory');{ Ім'я об'єкта }
```

Як бачите, для спільного доступу до пам'яті, що не асоційована з файлом, процес повинен використати функцію `CreateFileMapping` і зазначити, як її дескриптор значення `$FFFFFFFF`. Відповідний об'єкт відображення файла в цьому випадку буде пов'язаний із системним файлом підкачування сторінок. У цьому випадку також необхідно зазначити ненульовий розмір об'єкта відображення файла.

- Виділимо пам'ять:

```
GetMem(PMapView, SizeOf(LongInt));
```

- У випадку, якщо об'єкт відображення файла створити вдалося, то одержуємо його дескриптор:

```
PMapView:=(MapViewOfFile(hMapObj,
              FILE_MAP_WRITE,0,0,0));
```

- Збережемо й відобразимо координати миші в процедурі `FormMouseMove`:

```
PMapView^ := X SHL 8 + Y; // Збережемо дані
label2.caption := IntToStr(Hi(PMapView^))
                  + ':'+ IntToStr(LO(PMapView^));
```

```
label4.caption:=IntToStr(X);
label6.caption:=IntToStr(Y);
```

Поділюваний об'єкт відображення файла не буде зруйнований до того часу, доти всі процеси, що використають його, не закриють всі свої дескриптори цього об'єкта, використовуючи функцію `CloseHandle`. Це зроблено в процедурі `FormDestroy`.

Тепер розглянемо проект `Client`. У ньому ми відкриємо об'єкт відображення файла, створений у програмі `MemMap`, і відобразимо його в подання файла додатка `Client`. У такий спосіб у пам'яті додатка `Client` будуть зберігатися координати миші, які зчитує програма `MemMap` зі своєї форми. Їх ми й будемо відображати на формі додатка `Client`.

- Помістимо на форму два компоненти `Label` й один `Time` так, щоб форма виглядала так, як показано на рис. 3.7.


Рисунок 3.7 — Форма програми `Client`

- Відкриємо об'єкт відображення файла `shared_memory`:

```
hMapObj := OpenFileMapping(FILE_MAP_READ,
                           False, 'shared_memory');
```

- Виділимо пам'ять:

```
GetMem(PMapView, SizeOf(LongInt));
```

- У випадку, якщо `MemMap.exe` вами запущений і йому вдалося створити об'єкт відображення файла, то одержимо його дескриптор:

```
PMapView:=(MapViewOfFile(hMapObj,
```

```
                    FILE_MAP_WRITE, 0, 0, 0));
```

- Відобразимо координати миші в процедурі `Timer1Timer`:

```
Label2.Caption := IntToStr(Hi(PMapView^)) + ':' +
                  IntToStr(LO(PMapView^));
```

Завантажити розроблені додатки із сайта автора [3] і переконайтеся наочно, що механізм "файли, відображувані у пам'ять", дозволяє маніпулювати даними одного файла декільком клієнтам.

## 3.5 Стандарт COM

Стандарт COM (Component Object Model) починався з OLE. OLE дозволяло будь-якому OLE-клієнтові зберігати дані й викликати відповідний OLE-сервер, що обробляв їх під керуванням OLE-клієнта. Отже, додаток-клієнт використав функціональність додатка-сервера. Кількість серверів була обмежена (див. їх перелік, наприклад, **Вставка/Об'єкт** у MS Word). Розвиток стандарту COM розширює кількість цих серверів аж до написаних вами. Microsoft, позначаючи нові можливості стандарту COM, назвала ці програми ActiveX.

Тут ми спробуємо "препарувати" COM-технологію, щоб розібратися, як програма-клієнт використовує функціональність (методи) програми-сервера.

### Абстрактні методи

Почнемо ми з розроблення простого калькулятора (лістинг 3.3), використовуючи абстрактний метод.

**Лістинг 3.3.** Використання virtual, abstract и override

```
type
  MCalc = class
  public
    function Sum : integer; virtual; abstract;
  end;
```

```
  MyCalc = class(MCalc)
  private
    fx,fy : integer;
  public
    function Sum : integer;  override;
  end;
```

// Після визначення класів опишемо методи

```
function MyCalc.Sum:integer;
begin
 result:=fx+fy;
end;
```

// Створимо процедуру

```
procedure TForm1.btnCalcClick(Sender: TObject);
var
 Calc: MyCalc;
begin
  Calc := MyCalc.Create;
  Calc.SetOperands(StrToInt(Edit1.Text),
                   StrToInt(Edit2.Text));
  Edit3.Text:=IntToStr(Calc.Sum);
  Calc.Free;
end;
```

Тепер поставимо запитання: "Як іншим користувачам скористатися методом `Sum`, якщо в них у розпорядженні є тільки exe-модуль калькулятора?". Отже, як певній програмі-клієнту використати функціональність вашої програми-сервера?

Для цього програма-клієнт повинна знати повне ім'я програми-сервера, щоб його можна було завантажити з жорсткого диска в оперативну пам'ять. Тепер адресу методу `Sum` можна визначити за таблицею віртуальних методів (VMT), оскільки клас `Mcalc` був оголошений як `virtual`. Аналогічний механізм був використаний для реалізації взаємодії модулів, що виконують у стандарті COM.

COM підтримує модель клієнт-серверних взаємодій — між користувачем сервісів об'єкта (клієнтом), реалізацією цього об'єкта і його сервісів (сервером). Сервер реалізує об'єкт і сервіси, які ідентифікуються GUID в COM Library (TLB-файл). Клієнт, розміщуючи COM Library, створює COM-об'єкт (1 рис. 3.8), що, використовуючи GUID, надає клієнтові "місце розміщення реалізації" сервера (2 на рис. 3.8).

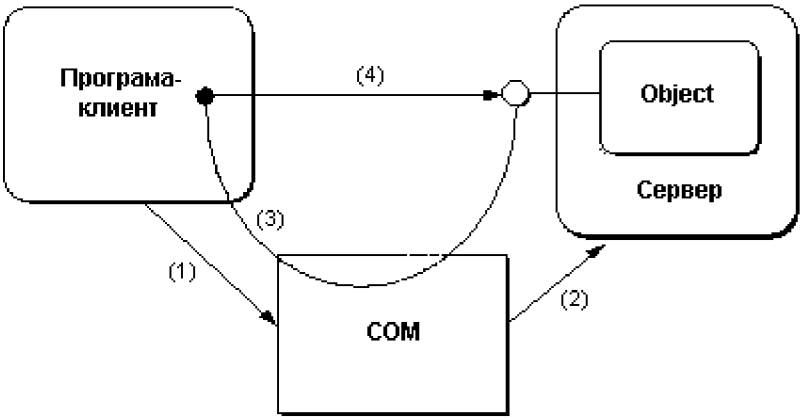



Рисунок 3.8 — Клієнт-серверна модель COM

Клієнт взаємодіє з COM-об'єктом за допомогою інтерфейсів. Інтерфейс пов'язаний із групою функцій, зв'язаних, у свою чергу, з COM-об'єктом, і не містить їхньої реалізації. Клієнт одержує покажчик (3 на рис. 3.8), ідентифікований за допомогою IID (Interface Identifier, власне кажучи, це той самий GUID), і вказує на інший покажчик, що, у свою чергу, указує на таблицю, що містить адреси реалізації кожної функції із цієї групи (рис. 3.9).

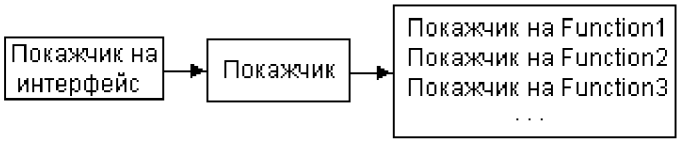



Рисунок 3.9 — Структура інтерфейсу

## "Препарування" стандарту COM

Як реалізоване використання методів програми-сервера клієнтами можемо побачити у процесі розроблення In-Process COM-сервера.

> Примітка. In-Process COM-сервер реалізується як DLL і запускається в адресному просторі клієнтського процесу.

- Розроблення сервера починається з команди **File/New/Other/ActiveX/ActiveX Library**.
- Потім створюється COM-об'єкт (**File/New/Other/ActiveX/Com Object**).
- Задаємо ім'я класу — `DisplaySomething` (рис. 3.10) і клацаємо на кнопці **OK**.

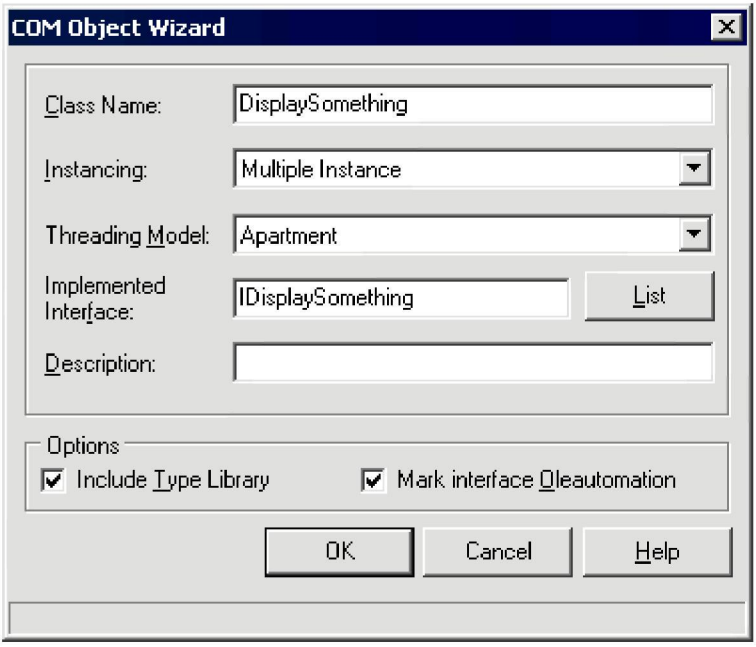



**Рисунок 3.10 —** Вікно майстра створення **Com Object**

- Зберігаємо проект під ім'ям `DisplaySomething.dpr`, а модуль — `DisplaySomethingU.pas`.

Якщо подивитися модулі проекту, то, крім модуля з вихідним кодом нового класу `DisplaySomethingU.pas`, там є й бібліотека типів `DisplaySomething_TLB.pas`. Це бінарна реалізація стандарту COM.

Тепер необхідно оголосити метод інтерфейсу IDisplaySomething. Для цього використовується модуль DisplaySomething_TLB.pas, оскільки цей інтерфейс повинен бути загальним і для сервера, і для клієнта.

- Додамо його слідом за GUID.

Оголошення інтерфейсу з точністю до GUID тепер буде виглядати так (лістинг 3.4).

**Лістинг 3.4.** Оголошення інтерфейсу IDisplaySomething

```
IDisplaySomething = interface (IUnknown)
  ['{F2DF5380-3BDE-4535-9787-E7A59E7E02C5}']
  procedure DisplayMessage(S : string); stdcall;
end;
```

- Реалізуємо процедуру DisplaySomething(S: string) у модулі COM-об'єкта (виділена курсивом у лістингі 3.5).

**Лістинг 3.5.** Код модуля DisplaySomethingU

```
unit DisplaySomethingU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj,
         DisplaySomethin_TLB, StdVcl, Dialogs;

type
  // Інтерфейс зазначається у списку спадкування!
```

```
  TDisplaySomething = class(TTypedComObject,
                            IDisplaySomething)
  protected
    procedure DisplayMessage(S : string); stdcall;
  end;

implementation

uses ComServ;

procedure TDisplaySomething.DisplayMessage(S: string);
begin
  ShowMessage(S);
end;

initialization
  TTypedComObjectFactory.Create(ComServer,
        TDisplaySomething, Class_DisplaySomething,
        ciMultiInstance, tmApartment);
end.
```

- Додамо в секцію `uses` модуль `Dialogs`, оскільки ми використовуємо процедуру `ShowMessage`.

  Примітка. Код проекту COM-сервера DisplaySomething ви можете знайти на сайті автора [3].

- Використовуючи команду **Project/Build All Projects** створюємо DLL COM-сервер.
- Зареєструємо COM-сервер DisplaySomething.dll у Windows (**Run/Register ActiveX**).

Якщо реєстрація COM-сервера пройшла успішно, то ми побачимо повідомлення: "Successfully Registered ActiveX Server ...". У розглянутому випадку GUID дорівнює DE765490-C18F-4B40-A29F-221E9E3ADD03. Запустимо Regedit і подивимося його опис (рис. 3.11).

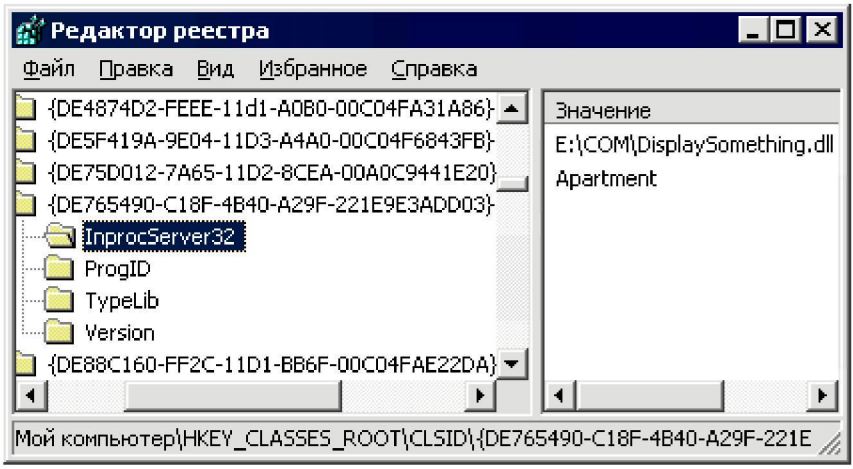



Рисунок 3.11 — Опис GUID COM-сервера в реєстрі Windows

Тепер реалізуємо програму-клієнт так, щоб вона ініціювала тільки що створений COM-об'єкт.

- Створимо нову Delphi-програму.
- Додамо в секцію `uses` імена модулів `DisplaySomething_TLB`, `ComObj` й `ActiveX`.
- Помістимо на форму кнопку, і поставимо у відповідність її оброблювачеві події `OnClick` таку процедуру:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FMyComObject : IDisplaySomething;
begin
  OleCheck(CoCreateInstance(Class_DisplaySomething,
                            nil,
                            CLSCTX_ALL,
                            IDisplaySomething,
                            FMyComObject));
  FMyComObject.DisplayMessage('Метод
                        DisplayMessage знайдений!');
end;
```

Примітка. Код програми-клієнта ви можете знайти на сайті автора [3].

Тепер поставимо запитання: "Як програма-клієнт користується методом `DisplayMessage`, що реалізований в COM-об'єкті, якщо у клієнта в розпорядженні є тільки DLL-програма цього об'єкта?".

Для відповіді на це запитання використовуємо код модуля DisplaySomething_TLB.pas. Клієнт "читає" його, оскільки він оголошений у його секції `uses`. За значенням {GUID} CLASS_DisplaySomething у реєстрі клієнт у розділі HKEY_CLASSES_ROOT\CLSID\{GUID} знаходить шлях до сервера (рис. 3.12) та ініціює створення COM-об'єкта операційною системою Windows.

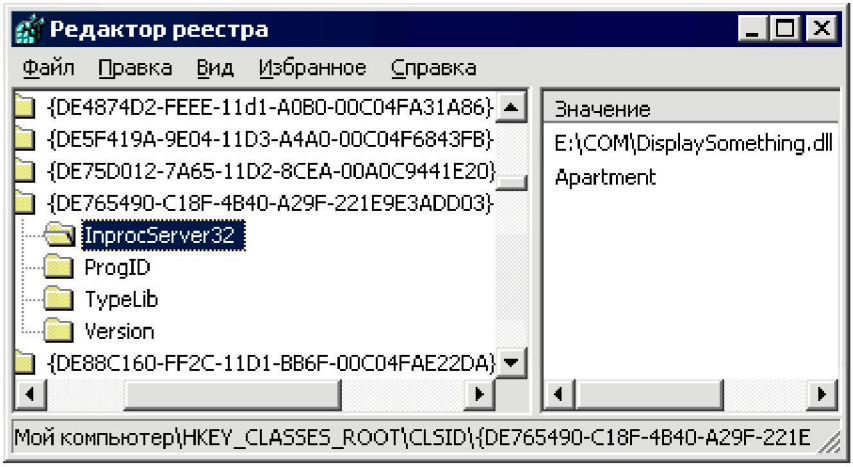



Рисунок 3.12 — Опис GUID COM-сервера у реєстрі Windows

Тепер залишається тільки експортувати інтерфейс і за ім'ям в ньому методу `DisplayMessage` знайти його адресу. Оскільки цей метод один (читай перший), то його адреса визначається в реєстрі за {GUID} IID_IDisplaySomething. Якби методів було два, то наступний був би через 4 байти (розмір покажчика).

Ви можете запустити COM_client.exe, переконатися, що він знаходить метод `DisplayMessage` (рис 3.13).

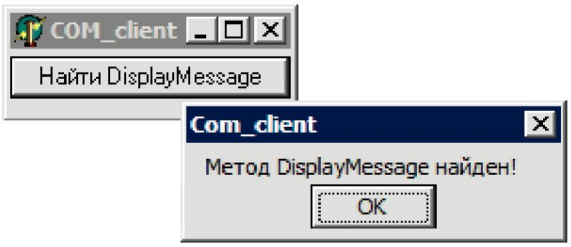



Рисунок 3.13 — Знаходження методу `DisplayMessage`

Крім того, ви можете переконатися, що COM_client.exe до знаходження методу `DisplayMessage` створює COM-об'єкт і завантажує DisplaySomething.dll:

- запустіть COM_client.exe;
- активізуйте Диспетчер задач: **<Ctrl>+<Alt>+<Delete>** (рис. 3.14);

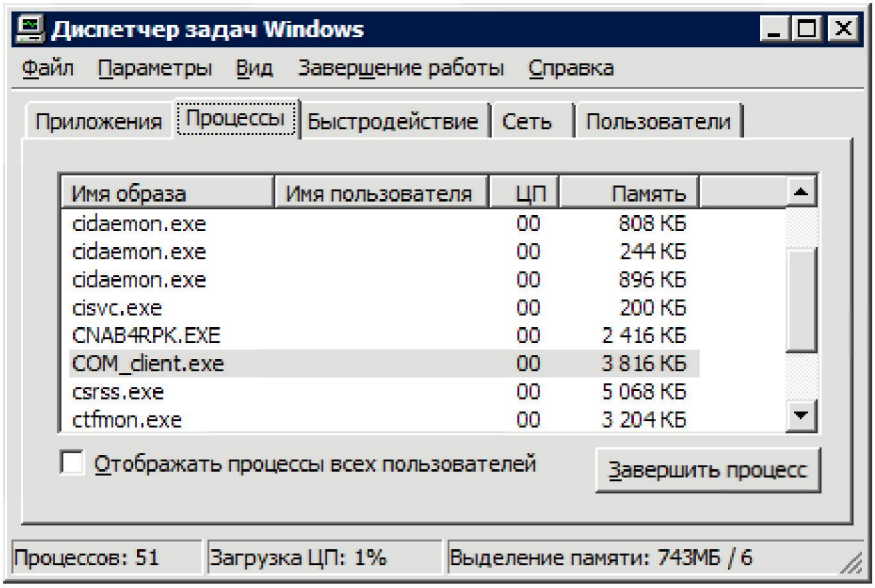



Рисунок 3.14 — Знаходження методу DisplayMessage

- клацніть мишею на кнопці **Найти DisplayMessage;**
- перегляньте вікно Диспетчера задач. Ви не побачите там нового процес DisplaySomething.dll, але пам'ять, використовувана

COM_client.exe збільшиться на величину, порівнянну з розміром DisplaySomething.dll.

## Запитання і завдання до розділу

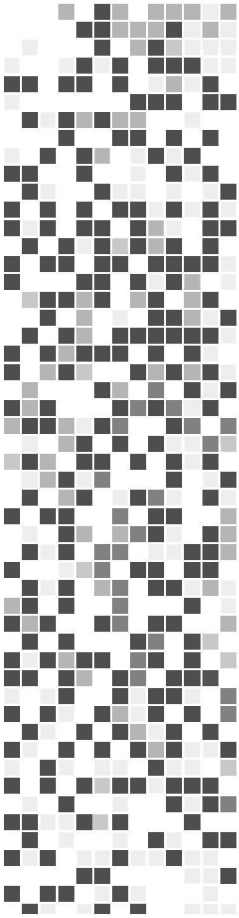
1. Використовуючи таблицю 3.3, створіть мейлслот для передачі повідомлень декільком комп'ютерам.
2. Рис. 3.3 відображує момент посилання повідомлення мейлслот-серверу. Скільки байтів у цьому повідомленні?
3. Використовуючи програму Filemon.exe, визначіть, іменовані канали якої служби використовує програма Internet Explorer у момент закриття:
   а) lsass;
   б) lsarpc;
   в) jqs.
4. Опишіть події, які можна спостерігати на вкладці **Performance Graph** контекстного меню **Properties** програми Process Explorer до й після закриття документів у програмі MS Word.
5. Запустіть COM_client.exe й, використовуючи програму Process Explorer, покажіть, що клацання мишею на кнопці **Найти DisplayMessage** ініціює процес завантаження бібліотеки DisplaySomething.dll
6. Дайте визначення таким поняттям COM-стандарту: інтерфейс, GUID, IID , CLSID, Type Library, COM-сервер, COM-клієнт, COM-об'єкт.
7. Виберіть у таблиці за номером вашого варіанта системне повідомлення або API-функцію й опишіть її.

***Таблиця 3.5*** *— Варіанти завдань*

| Номер варіанта | Системне повідомлення | Номер варіанта | API- функція |
|---|---|---|---|
| 1 | WM_CLOSE | 9 | BroadcastSystemMessage |
| 2 | WM_SYSCOMMAND | 10 | RegisterWindowMessage |
| 3 | WM_TIMECHANGE | 11 | PostMessage |
| 4 | WM_USER | 12 | SendMessage |

*Продовження таблиці 3.5*

| Номер варіанта | Системне повідомлення | Номер варіанта | API- функція |
|---|---|---|---|
| 5 | WM_ACTIVATE | 13 | CreateFileMapping |
| 6 | WM_CLEAR | 14 | MapViewOfFile |
| 7 | WM_COMMAND | 15 | OpenFileMapping |
| 8 | WM_COPYDATA | 16 | UnMapViewOfFile |



# 4

# Керування пам'яттю

Описано процес розв’язання задачі нестачі фізичної пам'яті для виконання користувацьких процесів й операційної системи. Використовуючи інструментарій сторонніх виробників, показано, як Windows вирішує цю проблему, використовуючи механізм "файли, відображувані у пам'ять". Виклад матеріалу супроводжується описом процесу розроблення програми. Це дозволяє показати, що не тільки розроблювачі ОС, але й будь-який програміст може використати віртуальну пам'ять.

Розділ 4

# Керування пам'яттю

Найчастіше користувач запускає стільки програм, що потрібний їм обсяг оперативної пам'яті перевищує розмір фізичного адресного простору комп'ютера.

Скільки всього фізичної пам'яті у вашого комп'ютера можна побачити на вкладці **Быстродействие** диспетчера завдань (рис. 4.2).

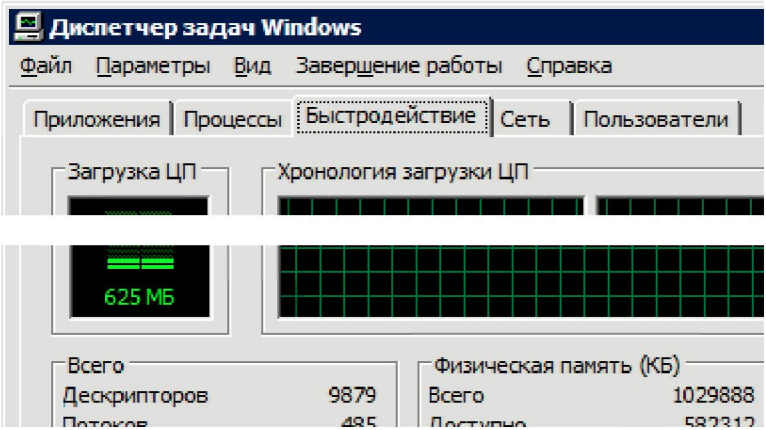



Рисунок 4.2 — Об'єм фізичної пам'яті

Ви можете самостійно розробити додаток для одержання інформації про загальне число кілобайтів фізичної пам'яті вашого комп'ютера. Для цього досить використати Win32 API-функцію `GlobalMemoryStatus`, у поле `dwTotalPhys` якої записуються дані про цей параметр.

Досить поставити у відповідність оброблювачеві події `FormCreate` форми такий код:

```
procedure TfmMain.FormCreate(Sender: TObject);
var
 MS: TMemoryStatus;
begin
```

```
  MS.dwLength:=SizeOf(MS);  {зазначаємо розмір структури}
  GlobalMemoryStatus(MS);  {одержуємо стан пам'яті в
                                момент виклику функції}

  ListBox1.AddItem('Загальна фізична пам'ять: '
     + FloatToStr(Round(MS.dwTotalPhys/1024)) + ' Кб',
nil);
end;
```

і ваша програма (з точністю до цифрових значень) відобразить число кілобайтів фізичної пам'яті вашого комп'ютера (рис. 4.3).

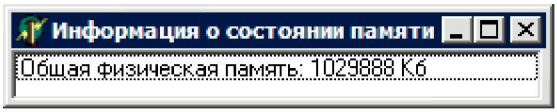



Рисунок 4.3 — Результат роботи програми

Ви можете ознайомитися з деталями проекту TotalPhys на сайті автора [3].

## 4.1 Організація пам'яті

ОС Windows вирішує проблему нестачі фізичної пам'яті, використовуючи ряд механізмів (рис. 4.4).

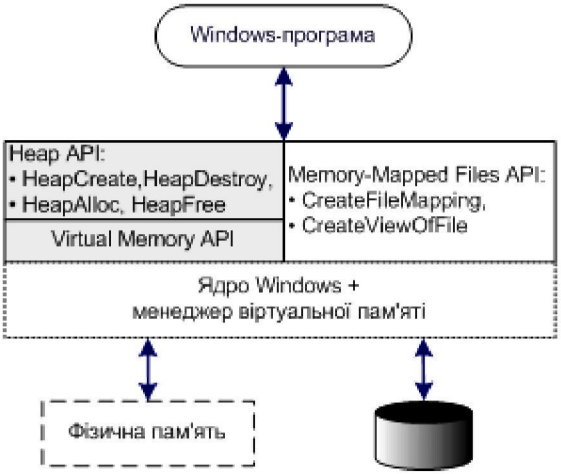



Рисунок 4.4 — Архітектура Windows API керування пам'яттю

Ці механізми дозволяють керувати пам'яттю програми, розділяючи її між фізичною й віртуальною пам'яттю. У якому обсязі виділяється фізична й віртуальна пам'ять для кожного конкретного додатка, можна побачити за допомогою Диспетчера задач:

- запустіть Диспетчер задач;
- перейдіть на вкладку **Процессы**;
- використовуючи **Вид/Выбрать столбцы**, виберіть **Память-использование** й **Объем виртуальной памяти** (рис. 4.5).

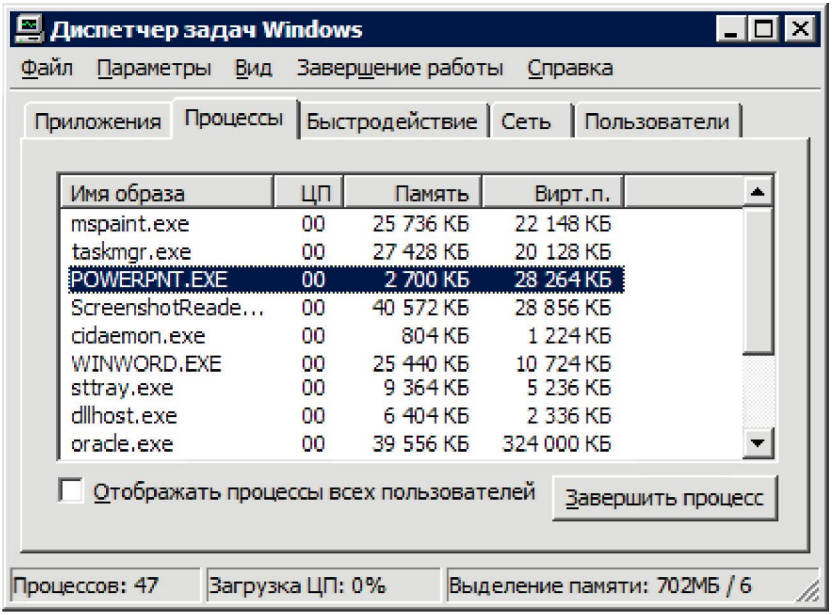



Рисунок 4.5 — Використання фізичної й віртуальної пам'яті процесами

## 4.2 Віртуальна пам'ять

Відомо багато варіантів визначення поняття "віртуальна пам'ять". Наприклад [7], "віртуальна пам'ять — схема адресації пам'яті комп'ютера, при якій пам'ять представляється програмному забезпеченню безперервною й однорідною, у той час як у реальності для фактичного зберігання даних використаються окремі (розривні) області різних видів пам'яті, включаючи короткочасну

(оперативну) і довгострокову (жорсткі диски, твердотільни накопичувачі)".

Тут наведене це визначення віртуальної пам'яті, щоб підкреслити, що цей термін має на увазі, що процесор для маніпуляції даними жорстких дисків використовує механізм "файли, відображувані у пам'ять " (див. розділ 3).

Ви можете переконатися в цьому, переглянувши відображені у пам'ять файли для певного процесу:

- запустіть програму Process Explorer;
- настройте нижню частину вікна на режим відображення DLL: **View/Lower Pane View/DLLs**;
- виберіть певний процес (рис. 4.6);

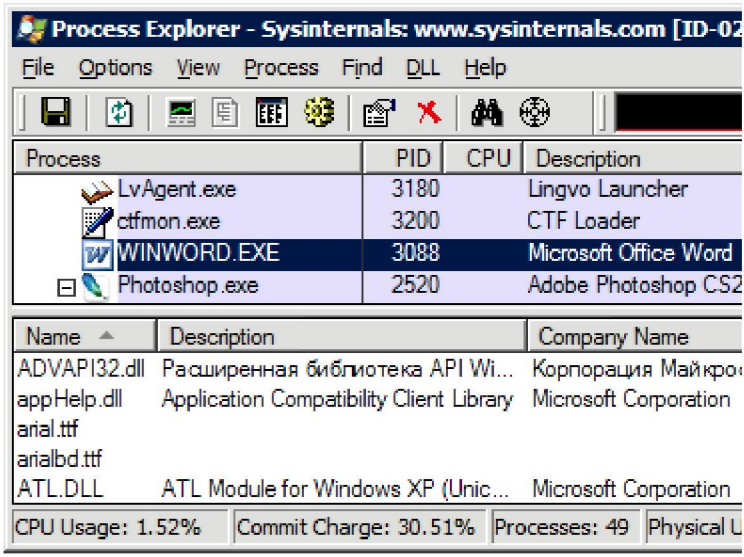



Рисунок 4.6 — Використання фізичної і віртуальної пам'яті процесами

Зверніть увагу, що, крім файлів бібліотек (DLL), тут показані й інші відображені у пам'ять файли: що виконуються (EXE) і файли даних. Наприклад, файл шрифтів MS Word — arial.ttf.

Механізм "файли, що відображені у пам'ять", крім того, що стирає для програміста грань між фізичною й дисковою пам'яттю (як показано у розділі 3, не тільки розроблювачі ОС, але й будь-який програміст може використати віртуальну пам'ять), ще й заощаджує фізичну пам'ять за рахунок того, що, наприклад, з одним екземпляром бібліотеки DLL може працювати кілька програм.

Переконайтеся в цьому:

- запустіть Process Explorer;
- знайдіть всі програми, які спільно використовують бібліотеку DLL kernel32.dll: **Find/Find Handle or DLL** (рис. 4.7).

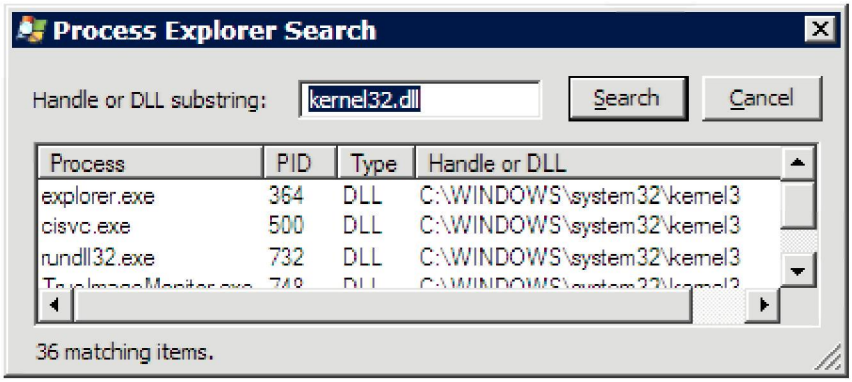



Рисунок 4.7 — Використання фізичної й віртуальної пам'яті процесами

## Запитання і завдання до розділу

1. Який обсяг фізичної пам'яті комп'ютера, для якого вкладка **Быстродействие** Диспетчера задач відображена на рис. 4.8?
2. На рис. 4.8 у полі **Выделение памяти** зазначено **Всего 1430736 Кб** — це:
   а) віртуальна пам'ять процесів й ядра ОС;
   б) фізична пам'ять процесів й ядра ОС.
3. Виберіть з таблиці за номером вашого варіанта поле API-функції GlobalMemoryStatus і опишіть його.

| Номер варіанта | Параметр |
|---|---|
| 1 | dwAvailPhys |
| 2 | dwTotalPhys |
| 3 | dwTotalPageFile |
| 4 | dwAvailPageFile |
| 5 | dwAvailVirtual |
| 6 | dwTotalVirtual |
| 7 | dwMemoryLoad |

4. Виберіть з таблиці за номером вашого варіанта поле API-функції GlobalMemoryStatus й опишіть його.

| Номер варіанта | Значення flProtect повідомлення |
|---|---|
| 8 | PAGE_READONLY |
| 9 | PAGE_READWRITE |
| 10 | PAGE_EXECUTE |
| 11 | PAGE_EXECUTE_READ |
| 12 | PAGE_EXECUTE_READWRITE |
| 13 | PAGE_GUARD |
| 14 | PAGE_NOACCESS |
| 15 | PAGE_NOCACHE |

5. Виберіть з таблиці за номером вашого варіанта програми й визначте, які бібліотеки DLL вони спільно використовують.

| Номер варіанта | Програми | Номер варіанта | Програми |
|---|---|---|---|
| 1 | Word, PowerPoint | 9 | msPaint, Calc |
| 2 | Excel, Word | 10 | Word, lsass |
| 3 | ProcExp, PowerPoint | 11 | Excel, svchost |
| 4 | Depends, Excel | 12 | msPaint, lsass |
| 5 | Word, Filemon | 13 | Filemon, PowerPoint |
| 6 | Explorer, Excel | 14 | Explorer, Internet Explorer |
| 7 | PowerPoint, svchost | 15 | Calc, svchost |
| 8 | lsass, svchost | 16 | Internet Explorer, Word |

Як визначити, які бібліотеки DLL використовують спільно кілька додатків, покажемо на прикладі MS Word й MS PowerPoint:

- запустіть MS Word й MS PowerPoint;
- збережіть у текстових файлах вміст нижньої частини вікна Process Explorer для обох процесів: **File/Save As**;

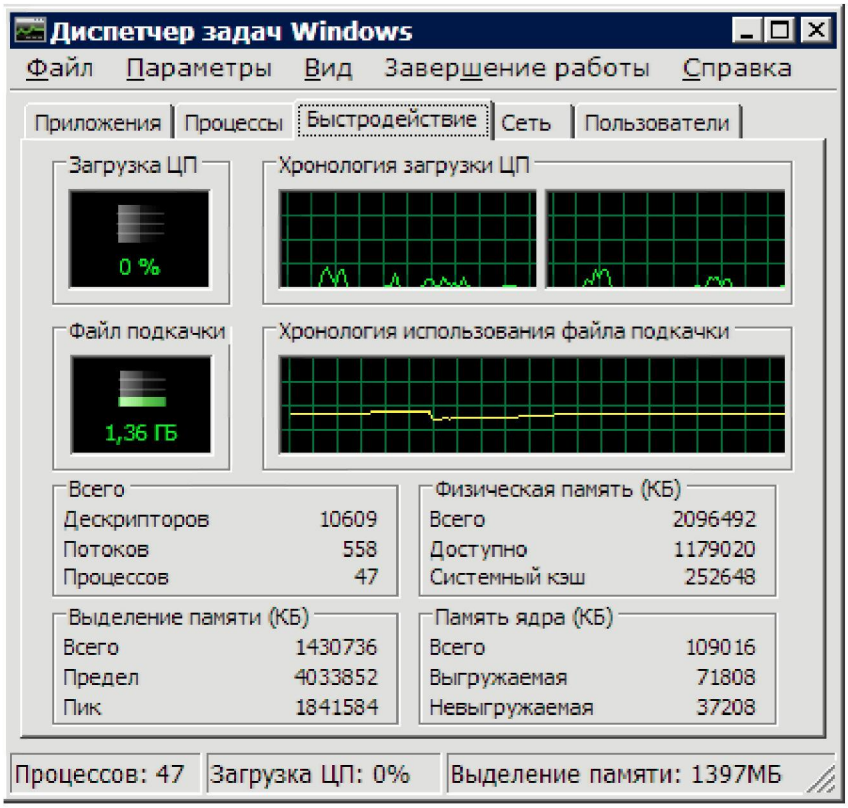



Рисунок 4.8 — Вкладка **Быстродействие** диспетчера завдань

- порівняйте текстові файли й знайдіть загальні бібліотеки DLL (рис. 4.9).

6. Чи можна видалити спільно використовувані бібліотеки DLL?

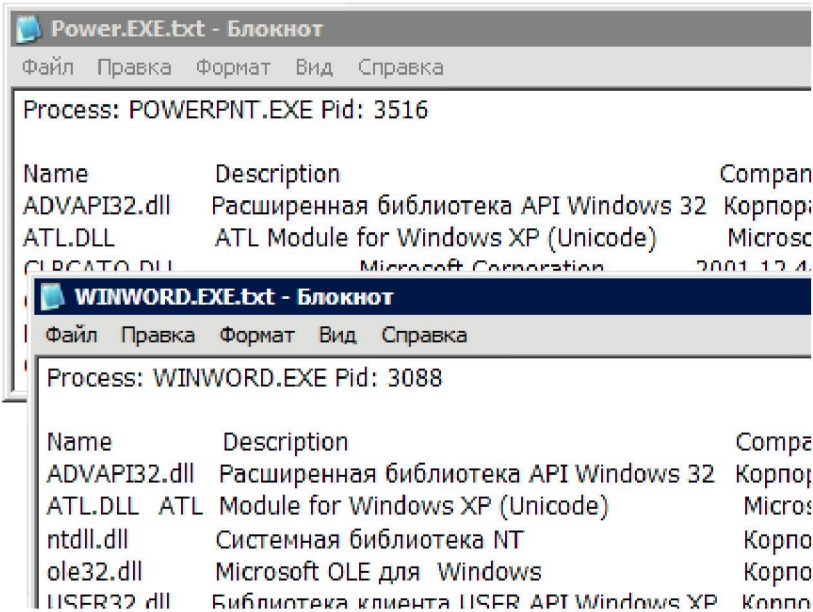
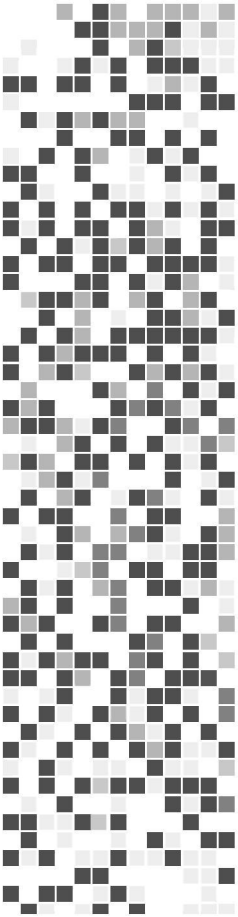



Рисунок 4.9 — Список загальних бібліотек MS Word й MS PowerPoint



# 5

# Файлові системи

Описана низькорівнева структура диска, організація зберігання на ньому даних. За допомогою інструментарію сторонніх виробників детально проаналізована структура даних дисків. Показано, як, використовуючи Win 32 API, визначити тип файлової системи, параметри низькорівневої структури диска.

Розділ 5

# Файлові системи

## 5.1 Низькорівнева структура диска

Інформація на дисках зберігається на концентрично розміщених *доріжках* (track). Доріжки нумеруються, починаючи із зовнішньої, номер якої — 0.

Кожна доріжка розбивається на сектори, що містять мінімальні блоки інформації (звичайно 512 байтів), які можуть бути записані на диск або зчитані з його. Сектори поєднують у кластери — мінімальні одиниці простору накопичувача, що адресуються файловою системою.

Накопичувачі (у тому числі й флеш-диски) можна уявити так (рис 5.1):

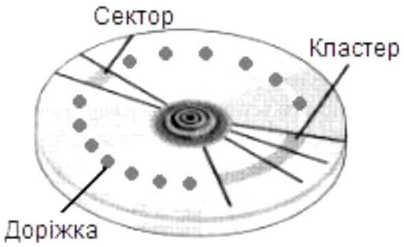



Рисунок 5.1 — Форматування накопичувача

Ви можете визначити кількість секторів у кластері, використовуючи API-функцію `GetDiskFreeSpace` у такий спосіб:

```
procedure TForm1.Button1Click(Sender: TObject);
type TDiskInfo = record
     RootPath: DWORD;
     SectorsPerCluster: DWORD;
     BytesPerSector: DWORD;
     FreeClusters: DWORD;
     TotalClusters: DWORD;
     end;
```

```
var
  DiskInfo: TDiskInfo;
begin
  With DiskInfo do
  begin
    GetDisFreeSpace(PChar('C:\'), SectorsPerCluster,
                    BytesPerSector, FreeClusters,
TotalClusters);
    ListBox1.Items.Add(format('Секторів у кластері:%d',
                              [SectorsPerCluster]));
    end;
end;
```

З деталями проекту SecClr можна ознайомитися на сайті автора [3].

## 5.2 Структура даних диска

Після включення комп'ютера або його перезавантаження керування одержує BIOS (Base Input/Output System — базова система уведення/виведення). BIOS ініціалізує й тестує апаратуру комп'ютера, після чого зчитує й завантажує програмний код, записаний у першому секторі завантажувального дискового пристрою (сектор називається Master Boot Record, MBR — головний завантажувальний запис), і передає йому керування. Звичайно MBR виконує такі дії:

- відшукує в таблиці розділів перший розділ, позначений як активний;
- намагається завантажити у пам'ять перший сектор знайденого розділу, такий сектор розділу називається завантажувальним;
- передає керування завантажувальному сектору (Boot Sector).

Завантажувальний сектор звичайно містить код, що завантажує з розділу операційну систему. Після завантаження операційна система маніпулює файлами даних, використовуючи для цього таблицю розміщення файлів. У такий спосіб узагальнена схема структури даних диска може бути зображена так (рис. 5.2):

| Завантажувальний сектор | Таблиця розміщення файлів | Каталоги й файли |
|---|---|---|

Рисунок 5.2 — Узагальнена схема розміщення інформації на накопичувачі

У випадку якщо ви використаєте ОС Windows і таблицю розміщення файлів FAT, то структура даних диска виглядає так (рис. 5.3):

| Завантажувальний сектор | FAT | FAT | Кореневий каталог і файли | Решта каталогів і файлів |
|---|---|---|---|---|

Рисунок 5.3 — Розміщення інформації на накопичувачі

Ви можете переконатися, що структура даних на диску саме така, як подана на рис. 5.3.

Щоб не ризикувати даними, візьмемо флеш-диск і перемістимо їх на жорсткий диск.

Використаємо інструментарій Acronis Disk Director Server [8]:

- запустіть Acronis DiskEditor і виберіть флеш-диск (рис. 5.4);

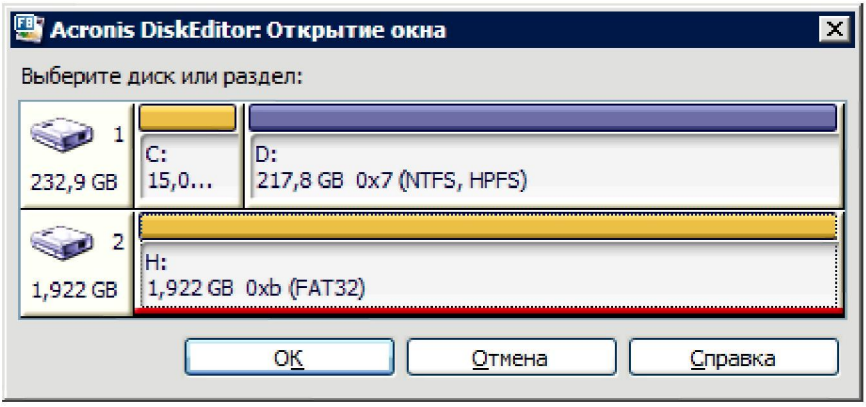



Рисунок 5.4 — Вибір накопичувача

DiskEditor автоматично позиціюється на завантажувальному секторі розділу, й головне вікно за замовчуванням відкриється в режимі перегляду **Загрузочный сектор FAT32** (рис. 5.5).

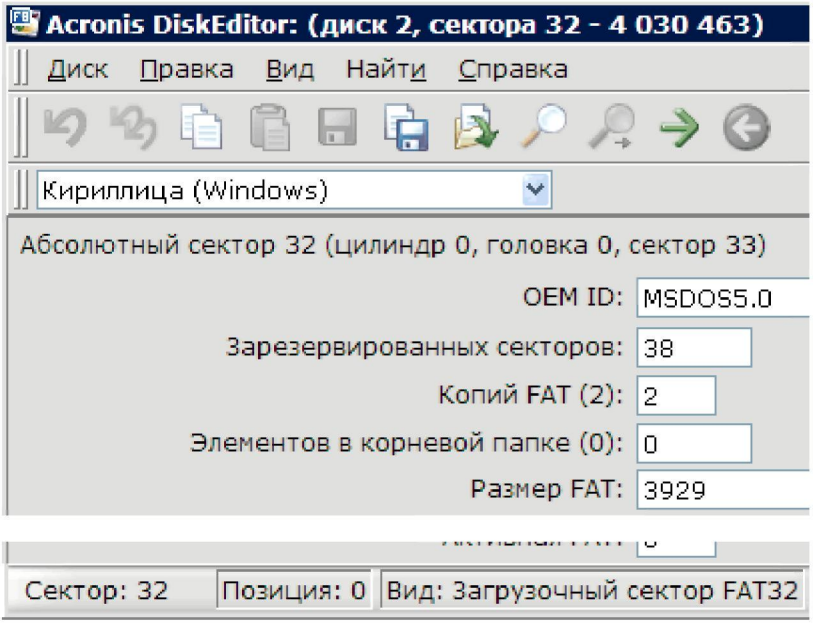



Рисунок 5.5 — BOOT-сектор

Можна переглянути *завантажувальний сектор* у шістнадцятковому вигляді (рис. 5.6). Для цього використайте меню **Вид/Шестнадцатеричный**.

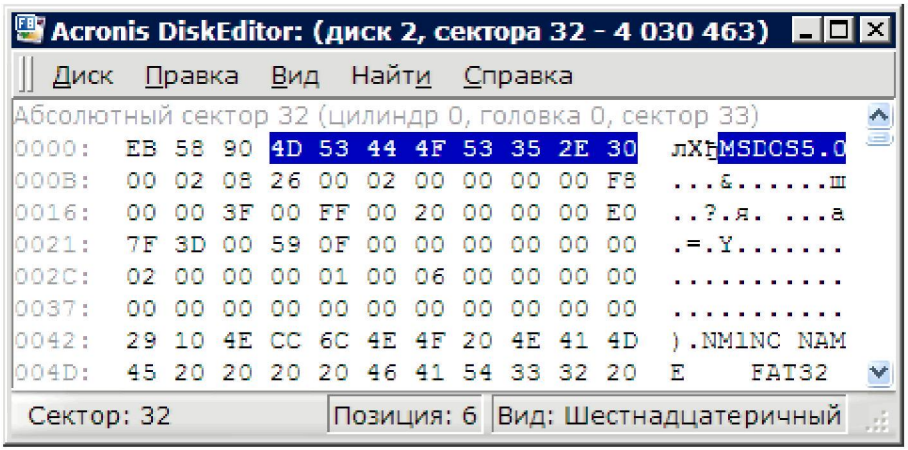



Рисунок 5.6 — BOOT-сектор у шістнадцятковому вигляді

Побачити *FAT* не складно. Подивіться на рис. 5.5 і ви зрозумієте, що він розміщений в абсолютному секторі 33.

Тепер знайдемо *кореневий каталог* диска. Для цього скористаємося формулою:

```
StartRoot = Зарезервирован.сектора +
                Загрузоч.сектор + 2*Размер FAT
```

Наприклад, якщо число `Загрузоч.сектор` дорівнює 32, а `Зарезервирован.сектора` дорівнює 38, а `Размер FAT` — 3929 (рис. 5.5), тоді `StartDATA = 7928`, тобто коренева папка розміщена починаючи із сектора 7928.

Щоб ми переконалися, що ми бачимо саме кореневий каталог, видалимо всі дані з диска (це можна зробити командою FORMAT), і потім запишемо на диск файл DATA.txt.

- Виберіть в меню **Найти/Перейти** й у вікні діалогу **Перейти к сектору** в поле **Абсолютный сектор**, уведіть розраховане число 7928.
- Переключіть головне вікно в режим перегляду **Вид/Папка FAT** (рис. 5.7);

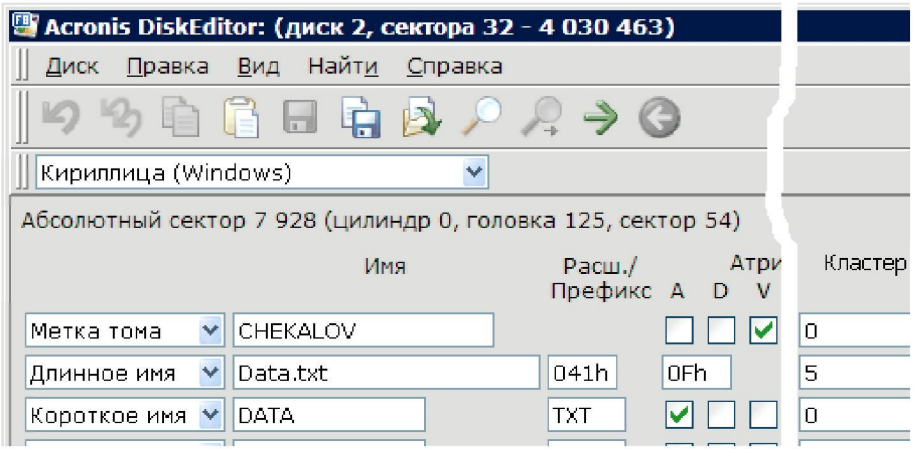



Рисунок 5.7 — Область кореневої папки

Знайдемо сектор, з якого починається *область даних* накопичувача:

- ініціюйте діалог **Найти**, й у поле **Текст** введіть початкові символи, які зберігаються в першому файлі області даних;
- змініть вид подання результатів пошуку — **Вид/Шестнадцатеричный** (рис. 5.8).

Наприклад, DATA.txt починається з 1111. Це зручно, тому що цифри в будь-якому кодуванні — це цифри.

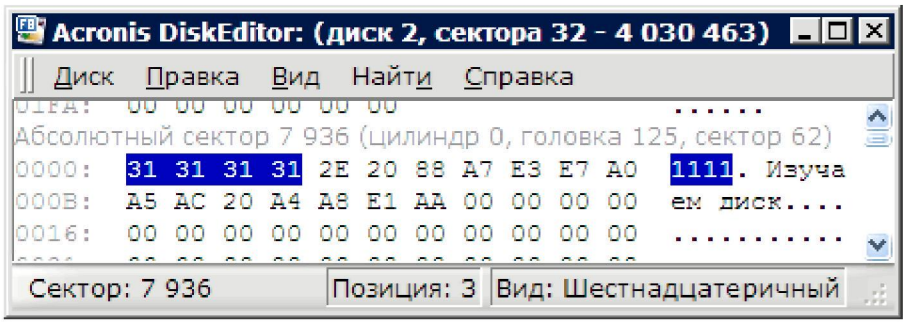



Рисунок 5.8 — Перегляд області кореневої папки

## 5.3 Файлові системи Windows

Накопичувачі бувають різних типів: CD й DVD-ROM, USB і жорсткі диски. Конструктивні відмінності накопичувачів накладають відбитки на файлові системи.

Ви можете визначити тип файлової системи, використовуючи API-функцію GetVolumeInformation().

- Помістіть на Delphi-форму такі компоненти: `TListBox` й `TButton`.
- Клацніть двічі на кнопці й усередині операторних дужок `begin … end`, додайте наступний код обробника події:

```
if GetVolumeInformation(PChar(combobox1.Text), Nil, 0,
        Nil, NoMatter, NoMatter, FileSysName,
                          Sizeof(FileSysName)) then
   with Form1.ListBox1 do
```

```
  begin
    Clear;
    Items.Add('Файлова система: ' + FileSysName);
  end
end;
```

Опишіть змінну `FileSysName` як `Array[0..Max_Path] of Char`.

Можливі значення параметра `FileSysName`

| | |
|---|---|
| FAT | File Allocation Table |
| FAT32 | Virtual File Allocation Table |
| HPFS | High Performance File System |
| NTFS | NT File System |
| CDFS | CD-ROM File System |

Ви можете ознайомитися з деталями проекту `GetFileSysName` на сайті автора [3].

## 5.4 Файлова система FAT32

Основною файловою системою Windows є NTFS. Її будова є закритою, хоча аналіз за допомогою DiskEditor можливий. Ми цього робити не будемо через обмеженість годин, відведених на вивчення курсу "Операційні системи".

Проведемо дослідження FAT32.

Щоб записати файл, операційна система повинна виконати таку послідовність дій. У вільному елементі папки створюється опис файла, потім шукається вільний елемент FAT32, і посилання на нього розміщується у записі папки (поле **Кластер** на рис. 5.7). Займається перший кластер, описуваний знайденим елементом FAT32. У цей елемент FAT32 розміщується номер наступного кластера або ознака останнього кластера в ланцюжку.

Знайдемо ланцюжок для файла DATA.txt у дампі FAT32 (рис. 5.9).

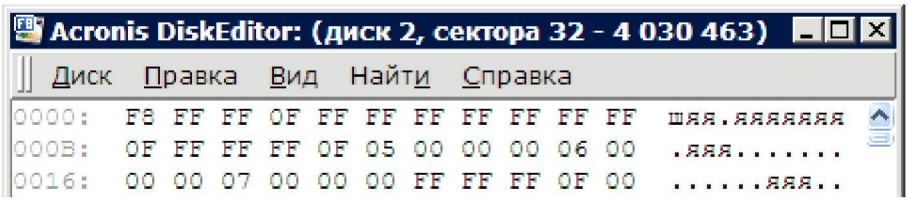



Рисунок. 5.9 — Дамп FAT32

Для адресації даних на диску у файловій системі FAT32 використовуються адреси в шістнадцятковій формі. Оскільки розрядність файлової системи 32 біти, то для адреси використовується 4 байти. Кожен байт задається групою з двох символів і у FAT32 може мати одне з таких значень:

- 00 00 00 00 — вільний кластер;
- 02 00 00 00 – EF FF FF 0F — номер наступного кластера файла;
- F0 FF 00 00 - F6 FF FF 0F — зарезервовані значення;
- F7 FF 00 00 — дефектний кластер;
- FF FF FF 0F — останній кластер у ланцюжку.

Таким чином, бачимо, що дамп, наведений на рис. 5.9, містить таку інформацію: F8 FF FF 0F — блок, якій задає ідентифікатор FAT (медіадескриптор); FF FF FF FF — зарезервований блок, FF FF FF 0F — кінець першого файла (розмір <= 1 кластер); FF FF FF 0F — кінець другого файла; 05 00 00 00 — покажчик на кластер 05 (початок файла DATA.txt), 06 00 00 00 — покажчик на кластер 06 (наступний кластер файла DATA.txt), 07 00 00 00 — покажчик на кластер 07 і FF FF FF 0F — кінець файла DATA.txt.

## Запитання і завдання до розділу

1. Виберіть у таблиці 5.1 за номером вашого варіанта параметр API-функції `GetDiskFreeSpace()` або `GetVolumeInformation()` й опишіть його.

***Таблиця 5.1*** — *Варіанти завдань*

| Номер варіанта | Параметр | Номер варіанта | Параметр |
|---|---|---|---|
| 1 | lpRootPathName | 8 | nVolumeNameSize |
| 2 | lpSectorsPerCluster | 9 | lpVolumeSerialNumber |
| 3 | lpBytesPerSector | 10 | lpRootPathName |
| 4 | lpNumberOfFreeClusters | 11 | lpMaximumComponentLength |
| 5 | lpTotalNumberOfClusters | 12 | lpFileSystemFlags |
| 6 | lpRootPathName | 13 | lpFileSystemNameBuffer |
| 7 | lpVolumeNameBuffer | 14 | nFileSystemNameSize |

2. Знаходження кореневої папки флеш-диска ми визначали за формулою
   StartDATA = Загрузоч.сектор +Зарезервирован.сектора
   + 2*Размер FAT.
   Чому `Размер FAT` необхідно множити на 2?
3. Запишіть ім'я кореневої папки жорсткого диску.
4. Скільки файлів й якого розміру зберігається на диску, FAT32 якого зображений на рис. 5.10?

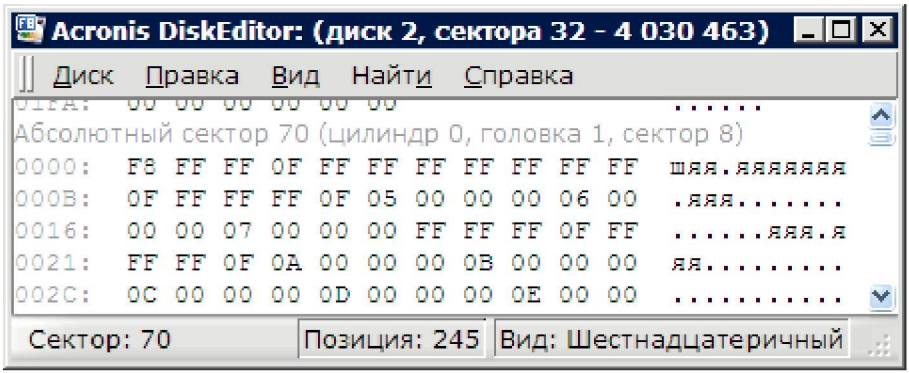



Рисунок 5.10 — Перегляд області кореневої папки

5. Використовуючи DiskEditor, знайдіть другий FAT і вкажіть його відмінності від першого FAT.
6. Дані якого розділу накопичувача зображені на рис. 5.11?
7. Використовуючи DiskEditor, з'єднайте фрагменти двох різних

файлів, модифікуючи FAT.

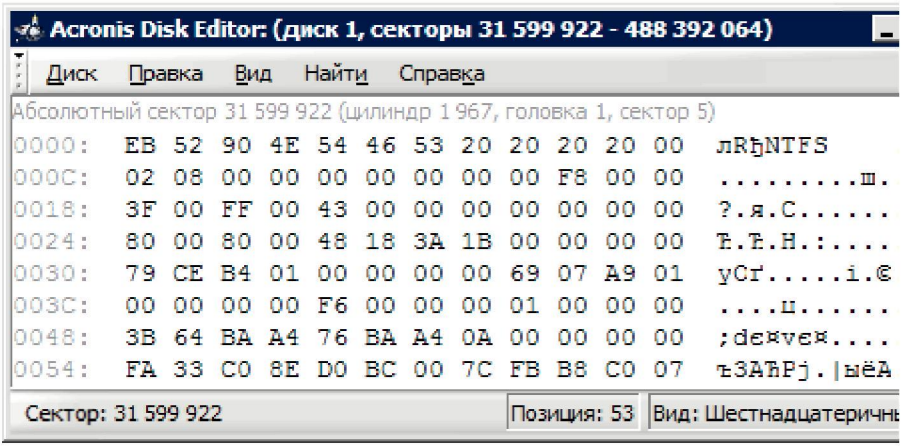



Рисунок 5.11 — Дані диска

# Список літератури та електронні ресурси


1 Кастер Хелен. Основы Windows NT и NTFS / Кастер Хелен ; *пер. с англ.* — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd.", 1996. — 440с. — ISBN 5-7502-0023-X.

2 Dependency Walker: http://dependencywalker.com/. — Назва з екрану

3 Чекалов А. Основы функционирования операционных систем: практикум: http://chekalov.sumdu.edu.ua/os.09/ index.htm. — Назва з екрану.

4 LiveKd: http://technet.microsoft.com/ru-ru/sysinternals/bb897415.aspx. — Назва з екрану.

5 Process Explorer: http://www.sysinternals.com/Utilities/ProcessExplorer.html. — Назва з екрану.

6 Filemon: http://technet.microsoft.com/ru-ru/sysinternals/bb896642.aspx. — Назва з екрану.

7 Виртуальная память: http://ru.wikipedia.org/wiki/Виртуальная_память. — Назва з екрану.

8 Acronis Disk Director Server: http://www.acronis.ru/enterprise/products/diskdirector/. — Назва з екрану.

9 Олифер В.Г. Сетевые операционные системы : учебник [для ВУЗов] / Олифер В.Г., Олифер Н.А. — СПб: Питер, 2003. - 544 с. — ISBN 5-272-00120-6.

10 Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Рихтер Дж. - 4-е изд. — СПб: Питер; М.: Издательскоторговый дом «Русская Редакция», 2004. - 749 с. — ISBN 5-272-00384-5.

11 Руссинович М. Внутреннее устройство Microsoft Windows:

Windows Server 2003, Windows XP и Windows 2000. Мастер-класс / Руссинович М., Соломон Д. - 4-е изд. — СПб.: Питер; М.: Издательско-торговый дом «Русская Редакция», 2005. - 992 с. — ISBN 5-7502-0085-X.

12 Шеховцов В.А. Операційні системи./ Шеховцов В.А. — К.: Видавнича група BHV, 2005. – 576 с. — ISBN 966-552-157-8.

13 Карпов В.Е. Основы операционных систем : курс лекций / Карпов В.Е., Коньков К.А. - 2-е изд. — М.: Изд-во "Интернет-университет информационных технологий - ИНТУИТ.ру", 2005. - 536 с. — ISBN 5-9556-0044-2.

Навчальне видання

**Чекалов** Олександр Петрович

# ОСНОВИ ФУНКЦІОНУВАННЯ ОПЕРАЦІЙНИХ СИСТЕМ. ПРАКТИКУМ

Навчальний посібник

Дизайн обкладинки О.П. Чекалова
Редактор М.Я. Сагун
Комп'ютерне верстання О.П. Чекалова

Формат 60х84/16. Ум. друк. арк. 4,25. Обл.-вид. арк. Тираж 300 пр. Зам. №

Видавець і виготовлювач Сумський державний університет,
вул. Римського-Корсакова, 2, м.Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.